Optimized Representations for the Acceleration of Display- and Collision Queries

Elmar Eisemann

UNIVERSITÉ JOSEPH FOURIER

THÈSE

pour obtenir le grade de DOCTEUR DE L'UJF

préparée au laboratoire Jean Kuntzmann dans le cadre de l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique

préparée et soutenue publiquement par

Elmar Eisemann

le 16 Septembre 2008

Titre :

Optimized Representations for the Acceleration of Display- and Collision Queries

JURY

Docteur Xavier Décoret	Co-Directeur de Thèse
Professeur François Sillion	Directeur de Thèse
Professeur Georges-Pierre Bonneau	Président du Jury
Professeur Wolfgang Heidrich	Rapporteur
Professeur Michael Wimmer	Rapporteur
Professeur John C. Hart	Examinateur

Acknowledgments

Many people deserve thanks for helping me. I want to mention my family first, my parents Freya, Hans and my siblings Martin and Almuth Eisemann, for coping, supporting and helping my throughout all these years. A special thanks goes to Hedlena Maria de Almeida Bezerra (to be precise). She had to bare with most of my stress, the many voyages, the long evenings in front of the computer... she read most parts of this dissertation and really is a constant source of my happiness. I am very lucky to have her in my life and cannot thank her enough.

I would like to thank my advisor Xavier Décoret for his constant advice and sympathy. He is a great researchers and a fantastic person. It is hard to imagine a better choice for an advisor and I will always be thankful for his support, advice, even friendship. Without him, I might not even be near where I am today!

I would also like to acknowledge my second advisor François Sillion. He really made me feel at home and invited me to ARTIS group meetings even before I was a member of the team. He was always available and a help whenever there was need. This also applies to Nicolas Holzschuch. He does a good job in guiding our team and gracefully handled the difficult task of following in François' footsteps.

I would like to thank especially John C. Hart for the fantastic opportunities that he opened up for me. I am very thankful that John accepted me as one of his students and for the confidence, motivation and advice he gave me. His passion for graphics and his deep knowledge and capacity of dealing with work will be a constant inspiration.

I am enlightened that John came to my thesis defense, which I consider a high honor. This also holds for Georges-Pierre Bonneau, Michael Wimmer and Wolfgang Heidrich, who were all part of this committee. I am really thankful that such high-ranked researchers accepted to come to Grenoble. I would like to thank particularly Wolfgang Heidrich and Michael Wimmer for agreeing to evaluate my dissertation, which was a considerable amount of extra work and both gave me very insightful comments and more than helpful advice.

I would like to thank Matei Stroila and his wife Elena Leonte who both made my stay at UIUC especially nice, but also want to mention the other students in the graphics group. In Seattle, Holger Winnemöller and David Salesin deserve special thanks for making my stay at Adobe productive, pleasant and an unforgettable experience. In Sweden, Ulf Assarsson and Erik Sintorn for the fantastic collaboration. It was great to have had the opportunity to work with all these brilliant researchers.

Further, I would like to thank the many friends that helped me during the duration of my PhD and stay in France. This includes people from ENS like Anas Jarjour, Mathias Kobylko, Martial Hue, Guillaume Moroz to name a few, many researchers at INRIA, and, of course, all the (Ex-)ARTIS team members, who I would like to thank for their support: David Roger, *Alex* Orzan (for the many invitations and good food), Pierre-Edouard Landes, Cyril Crassin, Lionel Baboud, Adrien Bousseau, Thierry Stein, Lionel Atty, Pascal Barla, David Vanderhaeghe, Kartic Subr (inventor of the Subr-Transform), Aurélien Martinet, Florent Moulin and Laurence Boisseux (also for the help with models), Sylvain Paris (for the many deep discussions on graphics, and of course the game nights), Stéphane Grabli, Samuel Hornus, Sylvain Lefebvre (all for many early, helpful discussions and introduction to the *nightlife grenoblois*), Gilles Debunnes (for more than *just* QGLViewer).

Emmanuel Turquin deserves to be mentioned separately because he had to cope with me the longest... we met even before we became students in Grenoble. Ever since our Master we were colleagues, neighbors, directors of movies, lost on the way to Yosemite, at concerts and especially good friends. There is no way any short paragraph could thank him enough.

Also, I want to underline my gratitude towards the permanent researchers: Jean-Dominique Gascuel (who more than once repaired my Linux installation already back in 2002 and with whom I had many insightful discussions concerning the GPU), Joëlle Thollot (for her motivating and friendly way and for sharing her vision of graphics), Fabrice Neyret (for sharing his very deep understanding of computer graphics and many interesting pointers), and Cyril Soler (especially for reading a larger chunk of this dissertation, but also much of what I know about graphics has its roots back in the time of my first internship with him).

Frédo Durand deserves a very special acknowledgment. His invitation to MIT was what made me decide to focus on graphics. I cannot thank him enough for putting me on this track and sharing his deep knowledge with me. He is an inspiration and I am more than thankful for his constant support.

I would also point out another very important person: Daniel Huybrechts. He was my professor at the University of Cologne. Not only was he a great mentor, he was also the one who believed in me and pushed me to apply for the École Normale Supérieure. Without his support this fantastic voyage I took during these last years would have never happened.

Thanks also go to Stacie Slotnick for taking a look at grammar and syntax.

I usually do not like to write acknowledgements because it is easy to forget someone, especially over such a long period of time. So, I decided to leave out at least one person on purpose. You, the special one, I know that you know that I mean you...

. . . to all that could have been. . .

Abstract

A rapidly growing computer graphics community has contributed to dramatic increase in complexity with respect to geometry as well as physical phenomena. Simulating, approximating and visualizing geometry consisting of tens of millions of polygons simultaneously tested for collision or visibility is becoming increasingly common. Further, recent technological innovations from graphics card vendors has given impetus to achieving these results at very high frame rates. Despite tremendous developments in graphics hardware, capturing the complete surrounding environment poses a significant challenge. Given the added time constraint for real-time or interactive rates, simplified representations and suitable approximations of physical effects are of key importance.

This dissertation focuses on simplified representations and computations to achieve real-time performance for complex tasks and concentrates on a variety of topics including simplification, visibility, soft shadows and voxelization.

Résumé

Le domaine de l'informatique graphique a reçu un formidable essor. L'incrément de la complexité de la géométrie et des phénomènes physiques traités est impressionnant. Simuler, approximer et visualiser une scène de plusieurs dizaines de millions de polygones, de surcroit simultanément testés pour déterminer collisions et visibilité, devient très commode nos jours. Les innovations récentes des fabricants des cartes graphiques nous motivent pour obtenir des résultats convaincants en temps interactif ou même temps-réel. Pourtant, même avec l'avancé de la technologie, la capture et simulation complète d'un environnement reste un défi énorme. Etant donnée la contrainte temporelle, des représentations simplifiées ainsi que des approximations adaptées des effets physiques sont d'une importance majeure.

Cette dissertation se concentre sur les représentations alternatives et des calculs adaptés pour arriver à une haute performance pour des tâches complexes qui apparaissent fréquemment en informatique graphique. Citons par exemple: la simplification de modèles, la visibilité, les ombres douces, les collisions ou encore la voxelisation.

First Words

A dissertation represents a large amount of information gathered over several years and presented as one single entity such as this document. Even though it is a scientific document, in my eyes, it is wrong that the same rules apply.

We published several scientific papers and for these we followed the strict standards. A dissertation is much more personal and so I took the liberty to sometimes ramble a bit, give anecdotes, background not directly related, or facts that (I personally) find interesting, but that might not be crucial for the understanding. Nevertheless, I hope that this gives more insight into how I personally see and understand computer graphics, how I believe elements are connected and why computer graphics is such an exciting field. Talking about excitement, I decided to rely on a layout that hopefully motivates to read and helps to easily navigate through the content, although I admit it might not be traditional.

This said, I hope you will enjoy reading...

Contents

Part I

Introduction

"It's a small world, but I wouldn't want to paint it."

Steven Wright

Chapter 1

Overview and Thesis statement

Thesis statement: *There is no ultimate solution to represent data. There are too many different types of complexity (data size, algorithmic complexity, ...) and too many different tasks (display, illumination, animation, ...). Our statement is that you need a variety of adapted solutions following a classification into Preprocessing, Transformation, Representation, Structuring and Restatement.*

Fig. 1.1 : Movie Characters

from left to right: Toy Story (1996, Pixar/Disney) , Shrek (2001, Dreamworks) , Ice Age (2002, Blue Sky/20th Century Fox) , Finding Nemo (2003, Pixar/Disney) , Wall-E (2008, Pixar/Disney)

Computer graphics is a large field of computer science that focuses on research related to the creation and manipulation of visual content. This includes the images, as well as geometry that undergoes a displaying process (also called rendering). Considered born in the 60s with the pionneering work of Sutherland [Sut63], this field has delivered rapid and overwhelming results despite its young age. Computer generated images, along with the plethora of tools for Digital Content Creation and Digital Image Processing, have reached a quality level that is more than impressive.

Even though computer graphics is often associated to a limited scope of applications, its impact is actually many fold and touches various fields.

- **Games** - with 9,5 billion dollars income (2007), it is one of the currently biggest markets in the entertainment industry. Models, effects and interaction becomes progressively more challenging and complex.
- **Movies** - special effects become increasingly important and fully computer generated feature films are common (Figure 1.1 shows some images from different movie productions). Even movies including life action contain large amounts of virtual environments. Star Wars

- Episode 3 contains over 90% of virtual content that needs a realistic appearance to merge seamlessly with the filmed elements.

- **Architecture/Design** - previsualization plays a more and more important role. Projects like the creation of a new stadium might involve hundreds of millions of dollars. A realistic (possibly animated/interactive) preview ensures that the visions match up. It also helps in the development process to make decisions with respect to the aesthetics, economics or design of the building.

- **Remote Surgery** - it is helpful to be supported by a simulation with augmented view and precision and have a robot carry out the actual surgery eventually at a distance. The challenges here include interaction, haptic feedback, as well as improved/enhanced display, often with measured data that need to be digitized on the fly.

- **Medical Data Visualization** - gigantic volumes of data exist that currently allow no free navigation or interaction. Further, it is very interesting to see that algorithms in computer graphics can have an important impact on other research fields. One example is Liu et al.'s work [LTF*05] on motion magnification. It recently helped in visualizing findings on the hearing mechanism [GAF07].

- **Training** - simulated events are often better than performing them in reality from an economic and safety standpoint. Many factors need consideration ranging from interaction to realistic visual output.

- **Biology** - animals adapt to their environment and their behavior in captivity changes significantly from their behavior in the wild. Recently, the use of virtual reality equipment allowed to persuade the animals of actual freedom and it became possible to investigate their natural behavior in a controlled environment (A recent study at the Max-Planck-Institute for Ornithology found out that birds use only half of their brain for long distance flying and sleep with the other. This result was revealed using a simulator. Projectors delivered realistic images (day and night changes, including an exact reproduction of the stars in the sky which are of importance for the orientation) and wind machines kept the birds in a fixed position. `http://www.orn.mpg.de/`). Various data is captured from sportlers to analyze their movement and improve training.

- **Tools** - content creation becomes increasingly important. Level design for games, image, and video development for homepages or communities. Providing solutions that allow the production of complex content in easy ways is of importance.

All of these fields (and many other related ones) have one problem in common which is that complexity is a limiting factor. As a simple introductory example, consider the obvious gap between the scenes in the real world - such as the one photographed by Miller (reproduced in Figure 1.2) and the scenes we are currently able to describe and render in a digital world. Nature is beautiful, and beauty can be complex! The interplay of light between the leafs of a tree, the shadows projected on the ground, translucency of the foliage in combination with caustics and refractions created by a small drop of water, are common phenomena. Nevertheless, the physical models that describe them are complicated, and their transcription in the digital world challenging. The creation of truly photo-realistic images is currently still an open problem in complicated scenes, even when relying on enormous computational power. It will probably remain an issue for a long time, if ever it can be solved at all.

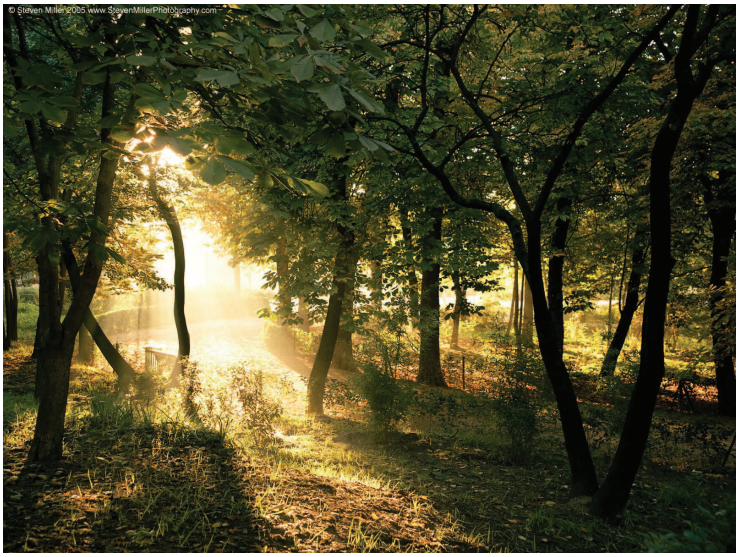


Fig. 1.2 : *Nature Is Complex* *(StevenMillerPhotography.com)*

Tasks become even more challenging when images have to be produced in real-time from a virtual scene, that is, 60 times per second on a screen with millions of points (not to mention the tendency to higher resolution screens or projectors and the necessity to supersample information to avoid artifacts on the screen). To make this situation even worse, real-life applications often combine several computations simultaneously, thus requiring each to be produced at much higher framerates. This is typically neglected in research work. The claim, a technique is ready to be applied in practice, has to be used carefully. In modern games, only 5-10 ms are allocated to shadow computations, thus imposing 200 *fps* (frames per second), currently disqualifying many algorithms on this topic. Even some of the current techniques we present might not yet be applicable in practice or at least within the next five years. Nevertheless, they are indicators for actual future solutions and efficient with respect to their scope.

Complexity is omnipresent; the number of characters in games increases (especially with the dawn of online games), graphical details appear, effects become more realistic, data sets in medicine/biology/geology represent terra bytes of information, interaction happens between millions of entities in a scene, images are made out of giga pixels... In contemporary computer graphics, all these concerns need to be addressed. The Holy Grail would be a representation that addresses all complexity issues at the same time, but, in our eyes, this is impossible. Thus, our first claim in this dissertation is that *there is no ultimate solution to address complexity.*

A Word on Display Rates *Real-time rendering is often related to numbers like 30 Hz (Hertz) and movies rely on 24 images per second, thus 24 Hz. The frequency at which the human brain interprets image sequences as continuous animation is around 20Hz, which is mostly enough for passive viewing. Nevertheless, oscillations at 24 Hz are still perceivable as flickering up to a rate of* > 84 *Hz. For this reason, Television works at higher rates (PAL: 50 Hz, NTSC: 60 Hz, this is enough due to screen latency). One resulting challenge is the mapping of 24 cinema frames onto the television rate. For NTSC a doubling and tripling alternation can be used. For PAL, frames are usually doubled. This makes movies shorter in Europe and the sound is just sped up accordingly. During this doubling process, information is spread: each displayed image shows only half an image (every second row: interlacing), the movements are still at 24 Hz. Nevertheless, television productions, like sport events, are shot at the rate of the TV with half images, to increase animation precision. This is also needed for large screens (IMax cinemas use 60 Hz full screen).*

Active viewing is different. If an input arrives between two frames (no matter the display frequency), the interaction is visualized only in the next frame. Thus 60 *Hz react with* 30 *Hz in the worst case. Further, if the refresh rate of the screen is insufficent, the screen update can collide with the refresh rate of the simulation. This results in another delay or shearing (vsync). In consequence, perceptual experiments showed that subjects could detect differences in behavior even beyond 90 Hz [LRC*03]. Interestingly, even though perceivable, above* 60*Hz, the refresh rate no longer improved the subject's performance. Hence, the suggested high rates for modern games.*

1 Sources of Complexity

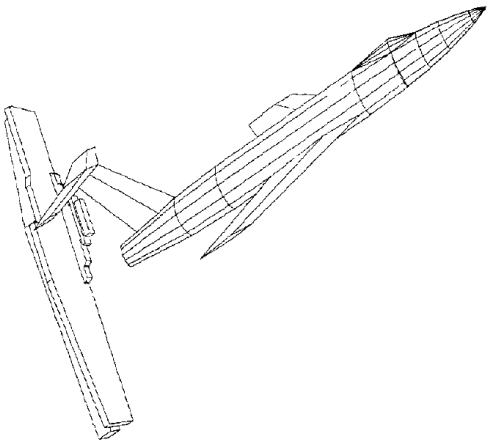


Fig. 1.3 : *A Complex Scene in 1967*

Before going any further, we must discuss what we mean by *complexity*. In general, complexity is not perfectly well defined. Seth Lloyd [Llo06] gave no less than 32 different definitions of complexity, and the list is probably not exhaustive. The general common denominator is that it refers to a system with many components. In computer science, complexity is often defined as *time/memory complexity*. This usually involves the Bachmann-Landau (Big Oh-) notation, originally introduced in mathematics, to describe the limiting behavior of a function with respect to its arguments. It thus reflects how the input size affects computation time and memory usage. Further, it often makes sense to limit the input to certain types, to find results in a worst-case, average-case or best-case scenario. As it is describes a limit value, the actual run-time might differ from the theoretical consumption. In particular, constants do not affect the complexity class an algorithm belongs to.

In the context of computer graphics, constants and actual performance on practical data sets are of crucial importance. To increase confusion, the definition of *a practical data set* might change from one generation of computers to the next. A scene that is described with a million polygons can be considered complex these days, but when reading older articles, a complex scene might just have contained a couple of thousands of triangles: In [App67], an efficient algorithm needed 84.6 seconds(!) to display 190 surfaces as a wireframe (see Figure 1.3). Currently, a million polygons sounds complex, but one should know that, on a newer graphics card, even this high amount of primitives can be processed at rates of a hundred Hertz. Typical games, such as the one shown in Figure 1.4, make use of hundred thousands of polygons per view, and use more involved computations that include, e.g., shading, material properties (including texture), shadows and even post-processing of the image, while aiming at a frame rate of around 30-60 Hz. For offline productions like computer generated movies, on the other hand, a million polygons are a small amount and millions of polygons per character are common. The notion becomes ambiguous.

Fig. 1.4 : *Assassin's Creed (2008, UbiSoft)*

Besides this algorithmic view, there are other situations which can be classified as complex. In a general sense, a task can be complex because it requires many steps (e.g., building a house); or because there are many particular cases to treat, some of them being impossible to anticipate. It can be complex because it implies many successive choices with a combinatorial explosion of the possible outcomes; or because the output itself is much larger than the original input. Finally, it

can even be complex because the current representation does not allow to easily infer the needed result, or because a solution can only be found after many iterations over the entire data set.

It thus makes sense to define complexity with respect to a task. A task is complex when there is no straightforward solution to realize it, or when the solution does not yield an acceptable performance behavior.

Since we claim that there is no ultimate solution to address complexity, we aim at providing a set of useful strategies to tackle it. And, even though, complexity goes hand in hand with tasks and contexts, we can classify some possible generic origins for complexity.

We distinguish the following sources:

- **Input/Storage** - The input or data representation can have a significant impact on the complexity of a task. If many or complicated components are involved, it is often costly and complicated to apply certain computations. For example, for a digital 3D model of a very large city with a lot of details and an explicit description via a static list of polygons, the storage cost can become prohibitive. This causes complex issues of memory and bandwidth management. On the contrary, a procedural model created on the fly according to the current view would be much lighter.
- **Manipulation** - Tasks can sometimes be repetitive and would allow a transfer from one entity in the scene to another. This is impossible if the scene does not provide information about similarity of components. For example, in a city represented as a soup of polygons, it is not possible to make global changes such as "every cylinder used to model chimneys should now be more tesselated". Content creation and edition is consequently complex. Using instances instead, one could edit a template and the change will be reflected by the other instances. Another example based on a city model would be the creation itself. This can be very demanding, but if the final shot is only done in one street of a city, many parts remain invisible. Having access to this knowledge gives the possibility to focus on those parts that will actually have be seen. As we will see, visibility is complicated and is usually precomputed, which is not easily possible when a scene is evolving constantly. The unpredictable scene modifications on the fly make such a problem particularly difficult to solve.
- **Interaction** - in many scenarios, elements enter into interaction with the scene (collision, light transfer, ...). The more elements need combination, the more costly this process becomes. Many autonomous characters exploring a city and guided by their field of view would imply costly tests of what they can see. This is even more challenging if interreflections of windows are taken into account. Just as for a mirror, the field of view would potentially be even larger. If each individual further interacts with the others, the potential complexity is enormous. A different example are raindrops falling on the houses. Millions of drops need to be treated. Accumulating in basins and creating small rivers and lakes.
- **Evaluation** - Data needs to be processed according to several tasks. In computer graphics, this usually means displaying it on the screen, or performing computations like feature line extraction (for line art), or perspective deformations according to a specific camera model. For example, the rendering of a polygonal building implies projection, rasterization, and shading for each polygon, which is complex, whilst not necessary. Indeed, for buildings in the distance, this should be much less complex and can visually be equivalent to render a fake facade represented by an image. Especially, shading is a means that provides us with

much richer appearance while avoiding some of the cost that would arise from a geometric representation.

- **Output** There are several situations were the output is the actual bottleneck in the process. Any combinatorial investigation can quickly lead to a very high resulting cost, but similar problems arise in different scenarios too. For example, the view of a facade can be generated with a 1024×1024 image. This can be complex to transfer over a network, and complex to decipher, e.g., for driving instructions. Instead, a clip-art or vectorial rendering can proof more adapted because it is less complex and more easily understandable.

2 Dealing With Complexity

Theoretically, complexity in itself is not an issue. Mathematically, the problem and its solution are usually well defined. The problem arises when trying to put such a solution *actually* to work.

The key in this context is to address the problems in an adapted manner. We need to develop specific solutions for specific types of complexity and the specific aspects of the task. Even though this might sound very vague, we have found that every existing approach seems to be based on, at least, one of five principles, and we believe that thinking in these categories is a good process to develop new methods.

1. **Preprocessing:** One way to deal with tight time constraints is to perform computations in advance. In this context, processing time is of less significance, as it takes place before the actual time-constrained evaluation.

   For example, finding the area of the surface of a polygonal model takes some time (linear traversal faces, cross-products, sum, ...). It is much more efficient to compute it once and to store it alongside with the polygon instead of evaluating it every time it is needed.

   In this document, we will see much more elaborated preprocessing that requires heavy reformulation of the computations. One particular example is the simplification of models with respect to a predefined viewing area.

2. **Representation:** The way data is represented is important. In some cases, one might need to rely on the original data; in some other cases, it might be possible to find a more appropriate solution.

   One representation might be more adapted for a certain shape than others. A centered unit sphere can be described by the simple equation $||x|| = 1$ defining the exact shape. An accurate shape description using triangles is impossible. This is of importance for approximate collision detections. For a sphere represented by its center and radius, it is much easier to perform than with a sphere represented by an approximated triangulation.

   If one wants to compute differential properties (curvature, tangent directions, ...) on a surface, it is difficult to find an appropriate formulation for polygonal meshes (discrete differential geometry), whereas if the data is given as an implicit function, these values are well-defined.

   In this dissertation, we investigate this topic by providing an algorithm to produce vector representations. We explore many different data representations and the construction of

vector art. The resulting clip art output can further undergo a stylization process. Such a vectorial representation is scale invariant, which proves often more efficient and user-friendly than a pixel-equivalent.

3. **Transformation:** In many situations, the result of a certain task is easier if a particular type of representation is used. In these circumstances, a transformation into an adapted form is of interest. This operation has to be performed rapidly as otherwise the benefit from having an appropriate description is lost.

   For example, shadow computations need to test for every point in the scene (or on the screen) whether the corresponding 3D point is lit or shadowed. It is therefore natural to have an optimized data structure for this query. Unfortunately, we cannot easily preprocess and store this information at sufficient precision (we cannot consider all possible light positions in advance). It is thus interesting to have a quick way to create a representation that allows to efficiently process these queries. In this example, it could be a shadow map [Wil78], that encodes the first encountered surface as seen from the light (thus the ones that are lit) and can be obtained very quickly by drawing the scene once from the source.

   Another example is to choose adapted geometric representations according to the current point of view. Many of the details that are present in the original model might not be necessary when projecting the object on a small area on the screen.

   In this document, we address the question of how to transform a triangulated representation into a GPU (Graphics Processing Unit) adapted data structure that enables rapid queries concerning the presence of matter or containment, as well as surface attributes like normals.

4. **Structuring:** A very common way to deal with complexity is to treat a problem hierarchically. This task is simple if the right structure is already provided, but, if not, the question of how to structure a scene is important. It can involve technical considerations such as creating uniformly sized groups of nearby objects. It can also repose on more semantic definitions like finding all instances of a model, or regroup elements sharing a similar material. The latter is often interesting in an editing context.

   In this dissertation, we present a system that drives stylization based on structural information which is derived from the scene.

5. **Restatement:** This word reflects the idea of reconsidering the actual problem to solve. This can either lead to an equivalent solution with less complex computations, or a restatement of the original goal with less complex requirements. The idea is to change the queries that evaluate a result.

   For example, light transport in a scene during a walkthrough only makes sense up to the intensity contrast that is still perceivable on the screen. If the light transport in a general scene is too costly, maybe a restriction to a certain type of material (for example diffuse) significantly simplifies the task. If only direct illumination is of interest, light-back facing elements can be directly excluded from any computation because they will not receive any energy.

   The central question in this context is how to avoid unnecessary computation and what consequences reformulations have. We apply this principle in several algorithms to produce information about visibility in a scene. In particular, we developed approaches to produce soft shadows, which is a major challenge in real-time rendering. Our spectrum varies from

highly accurate methods at lower frame-rates to more approximate and thus highly efficient solutions.

To make the idea of these points clearer, let's illustrate them with an example from outside of computer graphics. A classic problem of computer science is sorting. There are many different algorithms, the simplest (iteratively looking for the smallest remaining element and moving it to the front) has a quadratic complexity. Nevertheless, one can do better and there are solutions that have an $O(nlog(n))$ run-time behavior, where n is the number of input elements. The way a problem is addressed thus significantly influences its complexity. One important aspect in this context is the way data is represented. If the elements do not allow a random access, but need to be always traversed in order (as is the case for linked lists), some approaches might no longer be efficient. Conversion steps can be employed to change data into a different representation that then enables a more efficient evaluation. This transformation, of course, needs to be fast and should not dominate computation time because it will, otherwise, eliminate the advantage. If data is really large, it might not be possible to maintain all in memory. In a preprocess, already existent data might be arranged in such a way that efficient insertion remains possible, for example, by splitting the data into blocks. Finally, it could also be possible to relax the constraints of sorting and modify the original task to a classification process. Imagine one wants to sort exams according to the grade. In this case, a complete sorting process is unnecessary. Instead, the exams can be thrown in buckets according to the grade making the process extremely fast and simple. The result is equivalent to the real sorting procedure, but the complexity of the problem is much lower.

As you might have seen from this example already, in practice, the aforementioned possibilities are not independent. Usually, an algorithm combines aspects of several categories, but most allow a rather clear association. We believe it is advantageous to think according to these categories. It helps discover limitations and provides indications on how to improve a solution. Further, it helped drive our research work in these last years.

Of course, it is difficult or even impossible to prove the completeness of this set. One way to validate these categories is by verifying their compatibility with respect to previous work. In this document, we provide a large overview of related work and all mentioned articles are coherent with respect to our nomenclature.

By revisiting classical issues (display, visibility and its related topic: shadows - in particular soft shadows, collision detection, ...) and investigating them under the aforementioned aspects, we found the inspiration that helped us in developing entirely new directions or improvements over previous work.

In many cases, the categorization allowed us to address the problem in an appropriate way, which resulted in simple, efficient, and practical work. This question of practicality is a crucial aspect. Some research areas provide methods that are close to impossible to realize in practice. They are however important contributions for a scientific community as they might prepare the way to other insights. The problem is that they might be very difficult to implement or the theoretical gain only appears for highly unrealistic data sizes.

Reusing sorting for illustration purposes, it is known that quicksort has a bad behavior with respect to its upper performance bound, which can be quadratic. In practice, it performs very well because it has an expected running time of $O(nlog(n))$ and this with very small constants. Most theoretically more efficient solutions can thus be slower in practice. Nevertheless, for a

standard person, even quicksort might already be complex to use (only few people are sorting their documents this way by hand).

Of course, this is a simplified scenario because the implementation of quicksort is not a significant challenge for a computer scientist, but it shows that a solution needs a certain simplicity to be accepted and useful. In the industry, this is even more important because maintenance is an essential issue and difficult when a method is complex.

Our goal was to provide efficient and practical solutions with controlled and predictable behavior that can be used in practice. Effectively, all the results in this dissertation go along with working implementations. Some are a little more involved, but most are implementable by a graduate computer graphics scientist.

3 What Makes Graphics Difficult?

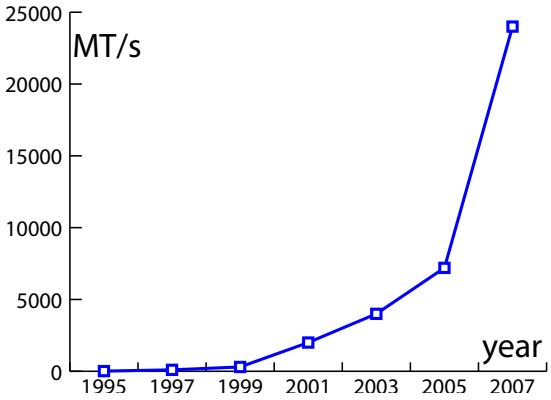


Fig. 1.5 : *GPU Texture Fill*

Computer graphics advances at an incredible pace these last decades and achievements in computer science do not only result from algorithmic solutions, but also (or even especially) from advances in hardware. It is a symbiotic relationship that is particularly true for graphics. So far, we got a glimpse of the sources of complexity and the categories we introduced to address problems that suffer from high complexity. One valid question is whether this complexity is not a problem that will disappear in the future with improvements of hardware and without further algorithmic research. This impression results especially from the quality of modern graphics applications and the performance of current computers. In fact, from the moment, the first hardware accelerated solutions to graphics hit the market in 1995, a success story started for many companies and an unforeseeable step-up in computational power resulted.

Currently, the performance of graphics hardware almost doubles every 6 months [AMH02]. Despite Moore's law, which stated that complexity of minimum component costs roughly doubles every two years, which was often connoted with a performance doubling on a two years basis (This observation remained valid into 2007 for CPUs (central processing units)). In Figure 1.5, we show only one aspect (texture fill rate) to illustrate this trend[1]. This strong amelioration of computational power in combination with improvements resulting from physical explanations and algorithmic considerations lead us to the exciting field of computer graphics that we know today. It enables the use of very rich and fascinating effects.

Nevertheless, the impression that the pure addition of transistors is sufficient to achieve the ultimate level of visual quality is a wrong assertion. It is true that our eyes have a limited

[1]The data represents the strongest architecture of that year independent of the price based on the technical reference information of NVidia cards. The latest G9 X2 achieves incredible 76.8 GigaTexels. But two cards need to be combined (SLI), this possibility explains the strong increase towards the end. It is thus likely that in the near future the speed-up will be weaker. Furthermore, fill-rate is no longer considered equivalent to global performance, since shading usually induces higher costs now.

resolution in space and time and, in practice, we see no more than about a million points spread out in a non-uniform way on our retina at around 80Hz. It is actually our brain who is responsible for the interpolation and illusion of a much better visual system. It thus seems natural that there exists some limit and that the absolute detail level will shift into reach. Unfortunately, many physical phenomena of our environment are independent of this resolution and thus, despite the aforementioned technical advances, a complete capture will not be possible in the near future.

Computer graphics is more than just hardware or coding issues and goes beyond entertainment. As simple as it sounds, even when restricted to the task of image creation, it is not easy. Many people believe that artists have drawn images for centuries and thus computers could just rely on this knowledge and reproduce the known artistic techniques. The first question is whether a painter, whose working domain is usually two dimension, is the appropriate comparison or whether a sculptor is not closer to the idea of creating a virtual 3D environment. But even if we stick to a painter, it is very difficult to transfer knowledge because computers need to rely on exact computations, where artists make use of intuition. Another important point is that even artists failed many times in history. Drawing images is not as simple as taking a brush and smearing color on a canvas.

Fig. 1.6 : *Master of the Sophien Cathedral of Ohrid, 11th Century*

It happened, that Cezanne thought for more than three hours before placing a single additional stroke on his canvas or consider perspective drawing which was only discovered relatively late in the 15^{th} century (see Figure 1.6 for an example from the 11^{th} century). Of course, perspective also played and does play the role of an artistic means, but especially medieval work often needed to be hide shortcomings and imperfections by choosing particular views or placing obscuring objects in the view of the observer. This intriguing idea is actually still used today. Especially computer games rely on a lot of hand tweaking and particular restrictions to achieve a convincing effect. It is a tedious and challenging process that often limits the freedom of an observer, the materials in the scene, or the interaction with the environment.

Contrary to nature, where we perceive a constant level of detail, it is a grand challenge for image synthesis to allow the same experience for an observer who has the freedom to move in an artificial environment. Usually, details are lost the closer we get to a virtual object, whereas details tend to appear in the real world. Even worse, also light interaction of sub-resolution elements can have visible effects on the macro scale appearance. Assuring a coherent appearance at all scales is a difficult task.

We are far from solutions that reproduce a fully convincing and completely interactive reality in a general scene. Much products give the false impressions that many problems are already sufficiently solved. The truth is that, even in offline rendering, many situations, if not all, need strong approximations because the realistic solution is not feasible.

One argument that one encounters often in this context is that artists would not want realistic rendering anyway, as it might take away their freedom of manipulating content. This is only partly true. Artists usually do like the richness and details of a realistically simulated process [Tes]. It is often only a limited control over the realism that is wanted. For example, indications on a global smoke movement, but no one wants to be obliged to manipulate every single particle to obtain a realistic appearance. The idea is to only guide reality. After all, a photographer usually creates, by definition, a photo-realistic result, but whoever saw the setup that is involved for a professional shot and the number of fill-, head- and supplementary lights, might start to wonder about this apparent realism. There is some level of artificiality, but in a realistic context. The fact, that the situation is physically realistic results in a convincing image despite the unnatural setup.

For many phenomena, we are quite advanced on a theoretical level and often a good mathematical description of the physical situation exists. It accurately defines what needs to be done to solve a particular problem, but the direct solution is often impractical.

Fig. 1.7 : *16.7 Million Triangles* *(Deussen et al. [DHL*98])*

For example, even good models to describe light transfer in a scene are mathematically relatively simple, but practically problematic. They are too expensive to employ and often rely on indirect definitions where the input is defined with respect to the searched output, resulting in a fix point equation. To illustrate this problem, let's consider light reflected from each leaf of a tree to others. This is of interest if we want to drive a plant growth simulation because the reception of light is strongly correlated with the development of branches. The typical combinatorial cost of such a situation is $O(n^2)$ light paths if there are n leafs in our scene. This is usually considered impractical because of the combinational cost (To visualize this relationship, one can imagine a soccer stadium filled with people as input and the current population of China as output). Scenes like the one in Figure 1.7 have no less than 16.7 Million triangles (and roughly the same number of leafs), the terrain is static and quite limited in extent. Even worse, in the situation of light transport, it is necessary to know by what amount the two leafs are visible. In this case, each leaf combination further involves tests against all other leafs. Describing accurately visibility relationships between n leafs leads to an $O(n^4)$ complexity and, ultimately, light bounces several times.

Costs like these show why the pure computational increase on a processor level is not enough to sufficiently address the issue of complexity. It becomes thus necessary to deliver methods that provide a (possibly approximate but sufficiently accurate) solution. In this context a good understanding of the initial mathematical formulation and the establishment of error bounds on approximate considerations is essential.

4 How to Represent Virtual Objects?

Besides pure computational and algorithmic decisions, the problem is actually deeper. It already starts with the input. We talked about complex scenes, but even the representation of shape on a computer is not direct. Mathematically, well-defined models, as well as algorithms, are necessary to enable the depiction on a machine. While some constructs can be expressed in a mathematical framework, real-world objects rarely obey a simple description. To approximate their appearance, a detailed formulation becomes necessary increasing directly the cost for any particular usage. All little imperfections that are very common in nature need to be encoded.

Today, the basic representation of shape are triangles. The main reason probably being that this representation is the simplest description resulting in a piecewise flat surface in space. This makes it especially simple to test against rays, which are at the basis of ray tracing. One other important fact is that each entity defined by three points in space remains linear under perspective projection. Even though this property was well-known for a long time, it has an important implication which is that the extremities can be projected independently before filling (rasterizing) the shape defined by the three projected points, allowing a parallelized process. It is at the basis of the graphics hardware pipeline that we will quickly review in Chapter 2.

On the other hand, the piecewise nature of triangles also has disadvantages. Representing curved or smooth data is particularly difficult. These are often better approximated by other mathematical constructs. Interestingly, this was one of the reasons, that NVidia's first attempt for graphics acceleration (NV1) was based on quadratic patches (NURBS). Unfortunately texturing and especially collision detection, was all but trivial and Microsoft just released the DirectX toolkit that was based on triangles. This destroyed all hopes for this architecture. Other examples for different representations include symmetric or rotationally invariant models with respect to an axis. In these cases, a section through the shape might more efficiently describe its form. For fog, smoke, or clouds, density functions are more appropriate because they do not allow any obvious representation via triangles.

In our work, we thus also considered other representations, like for example, implicit surfaces (defined by a variety of different basis functions), where the object is defined as the set of points obeying an equation of the form $F(x) = 0$. Algebraic shapes often permit a much simpler and especially exact description, where a triangular representation would only be approximate.

Also, there is more than just the actual shape of an object. Following our claim that no ultimate description exists, many objects need several representations. For collision detection it is often useful to provide some information about proximity. In these cases, it is possible to increase simulation precision when objects are getting closer, or adapt the repulsion forces depending on the intersected distance. This makes a distance field or an implicit formulation as $F(x) = 0$ more appropriate.

Today, there are basically three different ways to produce these models:

- **Hand-crafted** - created by a human using some tools.
- **Measured** - obtained from scanning reality (CRT, laser, LIDAR, satellite images, stereo vision, shape from shading etc.).
- **Derived** - procedural, or obtained from an input model (e.g., simplification, implicit approximations).

One prominent detailed scan is the model of the David statue by Michelangelo [SU]. The statue has been digitized in a tremendous effort by a team of 30 scientists from Stanford University and the University of Washington in the context of the Digital Michelangelo Project. The resulting model consists of 56 Million triangles and encodes the original masterpiece at a resolution of 1.0 mm (see Figure 1.8). This amount of data is far beyond what we can display rapidly on current hardware and it will take a couple of years before we will reach this point. Even then, for a virtual museum with several such models, there is still a long way to go.

Fig. 1.8 : *left: Original, right: Scan (Digital Michelangelo Project, Stanford University)*

In many cases, when models are scanned, the input is somewhat predefined. Nevertheless, the majority of models is actually hand-crafted and this ratio is increasing. Many commercial and especially free tools (Blender, Fibermesh, SketchUp, ...) appear on the market that grant an easy construction of shapes even for untrained users. Ignorant of any performance issues, the resulting shapes often contain redundancies or unnecessary tessellations. In any application, data is thus usually simplified, which involves a more trained user. In fact, in the game industry, polygonal level of details are usually done by hand because the quality is supposedly higher than when relying on current automatic solutions.

This motivated us to provide solutions that enables to involve aspects of complexity directly in the design process. In Chapter 5, we will introduce a tool to predict approximation errors induced by a simplified model with respect to the original. Besides the error-bound, it further predicts where the error occurs and from what point of view this error becomes most visible. The second contribution in this direction is presented in Chapter 12 where we compute and visualize visibility in a scene. This information can then directly be used in the design process. In a game, simple geometry is often added with the goal of blocking the view from more complex models. Typically, this involves many people who test the program for performance issues and then the scene is modified accordingly before the next test run is launched. The solution we provide addresses this issue and such considerations can be directly integrated in the design process. Even though very rudimentary, this represents an interesting way of tackling complexity at its roots.

5 What Major Challenges Are Addressed in the Dissertation?

Up to here, we have described the various aspects of complexity and categorized the solutions. We claimed before that an ultimate solution for all tasks does not exist and the strong entanglement between task and complexity will not allow us to provide a single multi-task representation. Nevertheless, when looking closer, one realizes that one of the most important aspects of complexity is arising from the unbalanced cost of global and local computations.

The very high performance of graphics hardware is currently correlated with the fact that streaming processors treat elements independently. Computations are performed in parallel and the process is sped up (compare our Chapter 2 on the graphics pipeline). This architecture treats lo-

cal computations increasingly efficient. Phenomena like shading or textures and bump- or normal mapping can be applied, as well as complex material models provide astounding realism. These possibilities lead to the gigantic quality leap in these last years because these local techniques significantly increase image quality (in fact, the image 1.4 does particularly benefit from nicely handcrafted textures that even contain illumination cues).

Non-local effects, such as realistic shadows, refractions, translucency, or global illumination, remain challenging. Interaction with the environment is a local effect, but becomes global the moment that a large variety of individuals need testing. Even in the latest computer games and movie productions, global effects are rarely based on an accurate physical model and often rely on artists (sometimes this even means to post process the result by hand) or heuristics to assure a convincing appearance. This solution can be time-consuming for the creator, expensive or inaccurate and even if humans can be fooled to a very high degree, it might lead to obvious inconsistencies. As a result, tedious hand-tweaking and supplementary restrictions are added.

Fig. 1.9 : *Crysis (2007, Crytek)*

An example for the limited extent of global effects is depicted in Figure 1.9. The scene is from the game Crysis by Crytek which currently represents the state-of-the-art in real-time rendering and generally delivers amazing visuals. Even though the image does look quite good, even this simple scene reveals several problems because the illumination of the explosion does not interact with the environment. It does not cast shadows, which would need to be very smooth due to the large extent of the fire, it does not influence the shadows on the ground, and the fire does not really affect the environment except for some lighting that is locally applied around the center of explosion. What makes this effect so difficult is the problem that the appearance of each point in the scene relies on global information. This information is not present, if each element of the scene is treated independently, which is the case for the graphics pipeline.

This holds for other phenomena we mentioned before, the complexity of light transfer and shadows are related to the interaction between many elements. Only knowledge of the scene allows us to locally determine whether a point is shadowed. This observation was the main driving factor of the work in this dissertation. We want to provide and exploit global information for scenes. In this way, we reduce complexity by providing alternative ways of representing, capturing and querying a scene.

The challenge for fast solutions is that today, this often means benefitting from the fast processing on graphics hardware - but it is not as general as a CPU. Much of our work has been influenced by its possibilities. However, we strived to not simply map CPU to GPU implementations, but, instead, to propose algorithms and data structures that really suit the nature of the GPU without being attached to the actual hardware. This avoids that our contributions will quickly be invalidated by future hardware generations. This is of major importance, in a time where technology changes rapidly and we find ourselves at the verge of merging directions (streaming hardware and multicore processors). Intel recently announced the Larrabee processor, which will

be multicore and address graphics applications. The trend here is to leave the original graphics pipeline towards more general processing units. It is thus highly important to understand how tightly a work is bound to the current hardware and whether it can be transported to future generations. For instance, it is currently unclear how data will be processed by future hardware. Will each processor only treat a single floating point value, as is the case for current graphics stream processors, or will the development move towards vector based processors, that can treat several such values in parallel (e.g., a color RGB(A)-tuple, which was the case only one generation of hardware ago)? So the question is whether each processor will be SISD (single instruction single data) or SIMD (single instruction multiple data) in a parallelized architecture. It is important to use smart coding of current hardware if it helps illustrating the advantage of a method, but if the only contribution is a smart mapping onto these very hardware specific properties, the work will probably have a lesser impact at long term. Only in the rare case, that such a technique results in a major advantage with respect to any previous achievement, it is of interest. As such, it might even influence future hardware design choices.

In this document, we will show several algorithms with the goal to decrease complexity of tasks and recover global besides solely local information. Their realm varies from CPU over hybrid to full graphics card implementations. We make use of recent features that are unlikely to disappear in the near future. Nevertheless, we usually illustrate that our solution also works on older hardware.

6 Contributions of This Dissertation

This dissertation contains many fold contributions in various areas of computer graphics and every idea in this document is back-upped by a running program, typically offering a user-interface to explore the influence of various parameters, the trade-offs and special cases.

The trend nowadays is to favor solutions that provide intuitive controls that can be easily understood and manipulated, while using the computer power to hide tedious and repetitive tasks. This can be found in the present dissertation under two forums. First, all algorithms have a very restricted set of parameters. Second, when you have to deal with complexity, why not use the human capabilities of abstraction and simplification?

Our work reflects this important tendency which is to move away from fully automatic methods and to rely on the user for certain specifications. User input is adequate for tasks that are difficult for a machine and simple for a human being. We kept this aspect in mind while working on all our solutions and, in particular, those that involve user interaction like our semantical grouping [BEDT08] and stylization of clip-art [EWHS08].

Computational power on the other hand, can be channeled to help the user in achieving otherwise difficult tasks. This was one motivation in our work in Chapter 12, where we show how instantaneous feedback on visibility can help in designing a scene. Having this information at hand, the artist can include performance considerations in the creation process. Our work on view-dependent object simplification 5 shares this aspect.

In the following, we will quickly review the nomenclature introduced in Section 2 and group our contributions according to these categories. As pointed out in this introduction, we address various kinds of complexity and we will shortly discuss the context and the achieved result with

respect to previous work. Later in this document, when presenting our algorithms, we will give a more profound comparison and discussion of related works to clearly underline the importance and difficulty of all the addressed problems.

PREPROCESSING

Part II concentrates on simplification via a preprocess. Very complex scenes cannot be displayed at real-time rates. To achieve a better performance, it is necessary to remove complexity by simplifying the elements. Despite a wealth of algorithms (we give a short overview in Chapter 4), it seems to be the case that little work has focused on the question of accurate simplification, which allows us to limit the error that is committed during the simplification process. The question we addressed in our work is: given an input model, how much can we deform it into a simpler shape, such that the appearance remains close to the original, as seen from a predefined observing region? Even though the resulting ***exact error bounds*** (and the related *point validity*, which indicates how much a point's position can be altered during simplification) are mostly of theoretical interest, it provides a deep insight into the simplification process and leads to a new geometrical interpretation of the committed error. Furthermore, the result could be used in the context of a validation system to find the strongest deviation of the simplified shape with respect to the original model or the viewpoint that maximizes the apparent error. We mainly focus on the 2D situation, but provide an extension of point validity to the 3D case. Furthermore, we implemented a new ***aggressive simplification algorithm*** as a proof of concept. In this context, we exploit global appearance and combine objects independently. We avoid relying on adjacency information, and create efficient representations that agglomerate features that can be simplified together. Our approach seems to be the first error analysis in such a general setting and is presented in Chapter 5.

The work was published in Computer Graphics Forum:
E. Eisemann and X. Décoret:
On Exact Error-bounds for View-dependent Simplification [ED07a].

REPRESENTATION

On the one hand, *representation* concerns the input, where we leave the ground of polygonal representations and investigate different definitions like implicit functions. On the other hand, it concerns the output, where we aim at the derivation of a vectorial ***clip art*** representation. This output is constructed from feature curves that we extract from the model. The result is a simple illustration of the original input based on stacked filled polygonal regions. These polygons capture the global shape of the initial object, whereas previous work relied on a collection of projected triangles. This is key to facilitate further editing and compacts the representation.

Even though, for triangle meshes, care has to be taken to assure closed regions, the extraction of the needed feature contours builds on previous work. For implicit surface, we describe a novel ***advection scheme*** based on to interrogate the surface by channeling particles towards feature curves. It is based on Langrange Multipliers and allows to capture the intersection between two manifold surfaces. The resulting vectorial output of this method breaks the barrier that is usually

imposed by pixel images. It creates a result that can be well displayed on a variety of output devices and deformations do not result in the typical artifacts that arise in pixelated versions.

We also developed a generic system to support the ***stylization*** of clip art to provide a richer appearance.This addresses the typical problems one encounters when illustrating a document. Pose, lighting, and style are usually inseparably combined in handcrafted work, making it difficult to find matching illustrations. The possibility to create clip art in coherent styles is a new way of navigation in this large space and addresses direct user needs.

Currently, our system aims at the creation of stills. Nevertheless, we investigated the theory behind the tracking of topological events of the silhouette which addresses temporal coherence. This result was mentioned in a single phrase in [SEH08], but no proof was given. We explain how the equations by Plantinga and Vegter [PV06] could accelerate currently existing methods and prove their equivalence in appendix C.

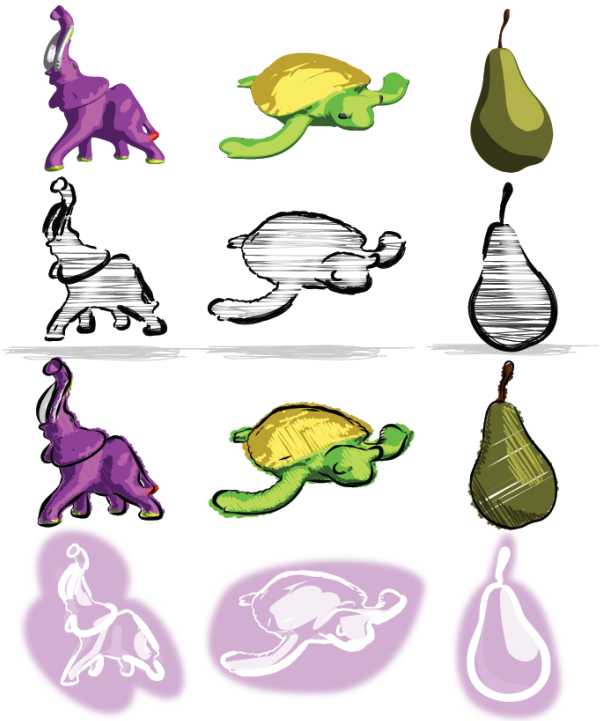


Fig. 1.10 : *Stylized Clip Art*

The results on the clip art creation for iso-surfaces were published in TVCG:
M. Stroila, E. Eisemann and J. C. Hart:
Clip Art Rendering of Smooth Isosurfaces [SEH08].

The clip art construction from meshes and the stylization system were presented at EGSR:
E. Eisemann and H. Winnemöller and J. C. Hart and D. Salesin:
Stylized Vector Art from 3D Models with Region Support [EWHS08].

TRANSFORMATION:

Part III will deal with voxelization which is one example of scene information extraction at runtime. Many tasks require such globally captured information. The simplest data is knowledge about the presence of matter, which is of interest in many visibility related tasks. In fact, for visibility, this is the only major information that is needed and it was what initially motivated our research on this topic. We aimed at a quick capture and transformation of scene data in a GPU adapted structure. More precisely, in Chapter 7, we present an efficient algorithm to ***voxelize*** a scene. It runs entirely on the GPU and avoids any CPU transfers or interaction. In consequence, the voxelization can be obtained on a per frame basis which makes it possible to perform very simple queries against the entire scene in each time step. We further modified our voxelization to derive ***solid interiors***. This very simple solution is particularly impressive, as it outperforms even recently published methods [CLT07] by several orders of magnitude. An

interesting extension produces local density and normal information from these voxel values. We show a variety of applications in Chapter 8 that benefit from the possibility to extract a high resolution ($> 1024^3$) voxel representation from complex meshes at 90 Hz and more. All of our applications improve upon previous work in some aspects.

The results on GPU voxelization were published at I3D:
E. Eisemann and X. Décoret:
Fast scene voxelization and Applications [ED06a].

The paper was reprised as a sketch at SIGGRAPH:
E. Eisemann and X. Décoret:
Fast scene voxelization and Applications [ED06b].

The solid voxelization was published at GI:
E. Eisemann and X. Décoret:
Single-Pass GPU Solid Voxelization and Applications [ED08b].

STRUCTURING

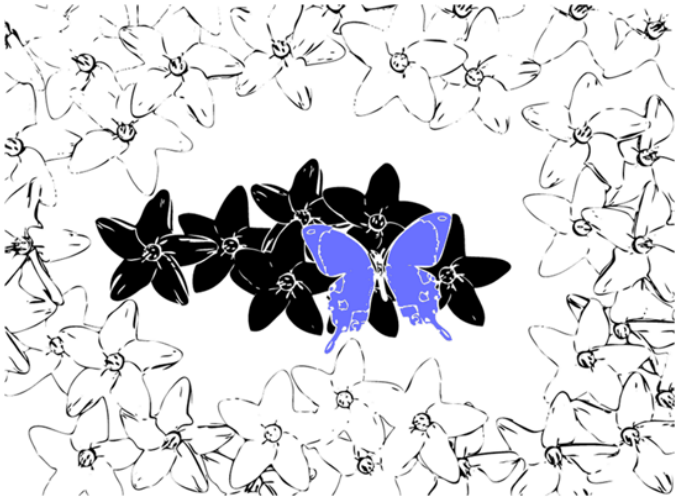


Fig. 1.11 : Attention Is Drawn Towards the Butterfly - *and its importance is spread over the group.*

In many situations, coping with information at different scales is possible by structuring a scene. Bounding-volume hierarchies like those employed in ray tracing are typical examples where hierarchical information is created.

Instead of aiming at an acceleration process we investigate how stylization can benefit from information on scene structure. ***Group stylization*** is a means in art to convey connections between objects and guide the interpretation of an image. We introduce a novel way of establishing groups in a dynamic scene and exploit this information to create non-photorealistic renderings.

The technique provides each entity with non-local information about clustered elements and its relation to other grouped entities. The criteria for this process are user-definable and allow to describe complex behavior with simple interaction. One stylization possibility is to apply a similar abstraction to all members of a group, which can lead to unified and more comprehensive representations.

The achievements in our context are twofold. On the one hand, we developed an interactive system that provides the possibility to use global scene information in the stylization process, on the other hand, the resulting output is often more uniform, as groups become more evident and the final rendering becomes simpler and easier to understand.

The results of this work were published at NPAR:
H. Bezerra, E. Eisemann, X. Décoret and J. Thollot:
3D Dynamic Grouping For Guided Stylization [BEDT08].

RESTATEMENT

Restatement describes the process of adapting or reformulating the problem to achieve an efficient algorithm. Part IV deals with visibility queries. Instead of adapting the model to the task as in the previous chapter, here we adapt the task. This means that we rearrange and modify the computations, to deduce a more efficient way of obtaining a convincing result. As we often project our work on visibility into the related context of soft shadows, we give an overview of existing work in Chapter 10, before presenting our contributions. We worked on the two extremes of this problem and provide a very fast approximation and highly accurate methods to perform visibility computations.

In our work on ***plausible soft shadows*** in Chapter 11, we achieve an approximate solution, that can be obtained very rapidly. At the time of writing, this algorithm is, to our knowledge, still the fastest soft shadow algorithm available that is capable of producing real penumbrae from a local source and this with a performance that is not related to the light's size. This is a remarkable property, as other solutions are either rapidly penalized when the light size increases, or cannot even deal with localized sources and solely support very large sources such that no shadow details remain. We achieve this result by casting the visibility queries into a filtering process, following the work by Soler and Sillion [SS98].

The second contribution in this part concerns ***visibility sampling*** presented in Chapter 12. Here, accurately sampled visibility in triangulated scenes is efficiently computed on the graphics card. The method finds application in the context of accurate light map computation and in visibility-assisted modeling. The simple initial situation of only a single source and receiver enables an efficient restructuring of the queries. Basically, we accumulate consecutively the blocking contribution of each triangle at hand which thus remains compatible with a streaming process. Our application to visibility assisted modeling is particularly interesting as it tackles complexity directly at the root during the construction process. Just like in our work on simplification, the provided information can help the user to make good choices with respect to efficiency during the creation. In other words, instead of taking a scene as given, we provide tools that help design a scene with respect to performance considerations.

The last contribution in Chapter 13 extends the previous algorithm to ***alias-free accurate soft shadows***. The distinction into source, caster, and receiver is lifted. The input is simply a source and a triangulated scene. The resulting approach delivers images equivalent to ray-tracing for volumetric light sources, but usually at much higher speed due to an efficient use of the latest DX10 graphics hardware. The implementation is simple because no hierarchical structures are needed and leads to an alias-free results for hard and soft shadows. One difficult part of the algorithm is the derivation of the ***penumbra region***, the region that needs shadow processing. In practice, we found that an overestimation was sufficient, but we did develop an analytical derivation of the shape for an ellipsoidal light source which we present in appendix A. Finally, we explain how to improve the quality of the shadow boundary based on an image-space approach in appendix B.

The work on plausible soft shadows was first published at SIBGRAPI:
E. Eisemann and X. Décoret:
Plausible Image-based Soft Shadows using Occlusion Textures [ED06c]

An extended version of this work was published in Computer Graphics Forum:
E. Eisemann and X. Décoret:
Occlusion Textures for Plausible Soft Shadows [ED08a]

Our approach for visibility sampling was published at Eurographics:
E. Eisemann and X. Décoret:
Visibility Sampling on GPU and Applications [ED07b]

The work on alias-free soft shadows was published at EGSR:
E. Sintorn, E. Eisemann and U. Assarsson:
Sample Based Visibility for Soft Shadows using Alias-free Shadow Maps [ED07b]

The final part V of this dissertation gives an outlook on future research in this area and concludes the document. To guide the reader through this dissertation, each part is introduced by a short description of its content and a repositioning into the general context. An independent lecture of each part is thus possible.

Concerning the lecture of this document: This dissertation is relatively long, even though we reduced its size significantly. It focuses only on some of our major results developed during the time of this PhD. ***To make this document more focused, we decided to remove all our contributions related to non-photorealistic rendering, as well as most side projects (that were not yet published), and solely present our work in the context of realistic rendering. This also means that we will not discuss the points "Representation" and "Structuring".***

As indicated in this introduction, we see all our contributions to be part of one global theme and we want to underline that they all contribute to one single achievement. Centering the document around realistic rendering allowed us to make it shorter and provide a text that better matches the length of a standard dissertation. Nevertheless, we decided to add some details in the appendix. These are not the major contributions of this thesis, but complete its content. One appendix C is related to our work in [SEH08], but can be read independently. We decided to add it because it provides a simple proof that was previously missing.

Overall, we wanted to make the document reader-friendly. This means that we merged several contributions together to provide a consistent and non-repetitive description of our work. The text contains two major parts on previous work (Chapter 4 and 10) that are not necessary for the comprehension of the document, but give an insight into many previous approaches and prepare for our contributions.

All chapters contain details (such as proofs, implementations for older hardware, or supplementary information in sideboxes and footnotes). These have been provided for the interested reader and to give more insight into the problems. They are important for anyone who wants a more profound understanding or tries to reproduce the techniques. Otherwise they can be omitted.

Finally, each chapter contains (several) summary boxes, that resume the content of the previous pages. On the one hand, this serves as a quick reminder, on the other hand, it gives the possibility to scan the document rapidly.

"640 k ought to be enough for anyone."

Bill Gates

Chapter 2

Graphics Hardware

Fig. 2.1 : *G80 Has Around 1 GB of Memory (NVidia)*

This chapter will give a short overview of the functioning of a graphics card, and the associated so-called *fixed function*, or *graphics pipeline*.

We will not present the details of the classical graphics pipeline, which can be found in [OSW*05]. Further, our presentation here is slightly simplified with respect to the reality, but it will make it easier to understand the functioning and properties, which are of interest in the following chapters of this dissertation. Anyone familiar with graphics hardware is invited to skip this chapter.

For a much more complete and thorough overview of the development of DirectX and the corresponding hardware, we suggest taking a look at NVidia's Programming Guides or [Krü07] that provide much information to different shader models.

1 Rendering

At the basis of image synthesis of virtual scenes are two algorithms: Ray-tracing and Rasterization.

Ray-tracing shoots one ray from the eye through each pixel and finds the first intersection with the scene. This impact point then defines the color of the pixel. This process is usually done using a local illumination model, or a recursive method is applied to yield approximations of more complex models.

Shader Model/DX: *Shader Models describe the minimum specifications of a graphics card. DirectX is a Microsoft library (or API) that gives access to hardware functionality, just like OpenGL. Often (probably this is related to the advertisement power of Microsoft), Shader Model and DX are used interchangeably. Basically, DX10, only gives access to Shader4.0 extensions. We will follow this trend and talk, for instance, about DX10 when we mean the extensions that are described in Shader Model 4.0. Nevertheless, these are also accessible through OpenGL, which we usually relied on to code our applications.*

Rasterization is at the basis of the graphics hardware pipeline. The algorithm loops over all primitives (triangles, lines, points, ...). Using projective geometry, the vertices are first transformed into screen space by a matrix multiplication. The second step builds the projected primitives from the projected vertices. A raster unit then works on all those pixels whose center falls in the projected primitive. It starts by computing a depth value, which is then compared to the

already present depth at the current pixel's location. If it is nearer, a color value is produced, written to the screen, and the depth value is updated.

Today, one can interact with this graphics pipeline through so-called *shaders*. These are basically small programs that modify the standard behavior of the pipeline (see Figure 2.2).

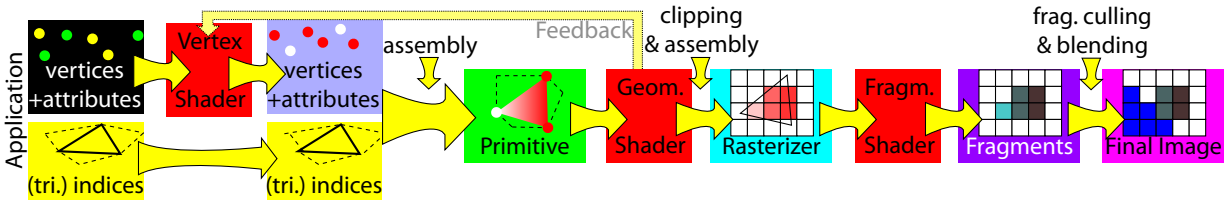


Fig. 2.2 : Graphics Pipeline

The application sends vertices and their attributes. The vertex shader modifies each single vertex. The modified vertices are combined to yield a primitive that is then altered in the geometry shader. Finally, the primitive is rasterized and the fragment shader works on each resulting fragment.

Texel, Pixel, Fragment: *Even though, a texel refers to a texture, a pixel to the screen, and a fragment to the entity that is a candidate for being blended in the framebuffer (e.g., color and depth), we will not always make an exact distinction in order to simplify explanations.*

- **Vertex Shader** - The input of this step are vertices specified by the application. The latter usually needs to also define how these vertices define primitives, but this assembly information will only be used in the next stage of the pipeline. Here, isolated vertices are treated independently of the triangle it belongs to. Usually, in this step, the vertex's coordinates would be multiplied with a matrix to transform it according to the current point of view. Nevertheless, on newer graphics cards, it is possible to transform the vertex based on a program that can make use of the vertex's data or texture memory on the GPU, as well as an array of values sent from the CPU (uniform variables). The output is a single vertex.
- **Geometry Shader** - This rather new element of the pipeline allows us to apply modifications to each primitive. The input data in this step are the primitives assembled from the vertices of the previous stage. During the transformation, access to uniform variables and texture memory is allowed. Interestingly special data arrangements can further give limited access to a a neighborhood of the primitive. For example, for a triangle, its three neighboring triangles can be accessible. It is currently not possible to extend this further. The output of this phase are none, or even several primitives that are clipped to the camera frustum once this step is performed. Theoretically, 1024 vertices are currently possible, in practice, more than 6-18 results in a strong performance penalty.
- **Fragment Shader** - The assembled primitive is rasterized (meaning scan converted to *fragments*, see next sidebox). The fragment shader allows to modify the fragment's color value and its depth before it is blended into the framebuffer. Besides the information available from the current primitive that was rasterized, again uniform variables and texture memory can be involved. The way data is passed from the primitive to the fragment can be roughly specified: continuously varying data (interpolated from the vertices), flat data (one value per triangle) and some more advanced sampling strategies exist. The output can be discarded but is usually a fragment (color and depth).

At the end of this pipeline, the resulting depth value is tested against the depth buffer, if this test fails, the color value computation can be skipped because the element will never show on the screen[1]. If it passes the test, its color is produced and blended with the information already present in the framebuffer.

There is currently no way of interacting with the blending process in a programmable manner. The methods can only be chosen from a set of operations. This includes logical (bitwise) operations and the more general alpha blending. Alpha blending is used to combine the current color value with the current framebuffer content in a weighted way based on a fourth color channel: the so-called alpha channel. This allows, for example, to sum all values.

2 Per-fragment Processing - Culling

Before outputting a fragment, a variety of tests are applied. We already mentioned depth, but further scissor tests (against a user specified rectangle, useful for light sources with attenuation), depth extents, alpha tests (testing the alpha channel against a user defined value), and the stencil test, to name the important ones, are all applied before accepting the fragment. The stencil test is the most complex one and based on an 8-bit stencil buffer. It passes, depending on a comparison of the buffer's current value to a reference. Based on this outcome and the result of the depth test, one can specify how the stencil buffer's value is altered. Currently, one has to choose from a set of predefined behaviors and the stencil buffer outcome is not accessible, nor specifiable in a shader. Only, if all the tests pass, the fragment arrives in the framebuffer.

Depth Peeling *Correct rendering of transparent objects is not straightforward with alpha blending operations. Elements would need to be sorted appropriately from back to front. This is contrary to the parallelized design. Depth peeling [Eve01] forces a correct ordering via several render passes. In the n^{th} pass the n^{th} depth layer can be extracted by smartly discarding all fragments that are closer than the last extracted depth. In consequence, this results in a depth map of all the scene parts that are one layer farther away. Newer, more efficient approaches use conservative read/write operations [LWX06]*

3 The Framebuffer

The framebuffer, that will contain the final image can hold a variety of data types: IEEE floats, integers, fixed point (8, 16 bit), ... and doubles are to come.

It allows us to connect several attachments to which one can write simultaneously (this is referred to as ***Multiple Render Targets (MRT)***). Up to eight such buffers can currently be attached on an NVidia G80. The introduction of MRTs can be considered a small revolution. With a single geometry transformation, several color values can be produced per pixel. This possibility gave rise to *deferred shading*, which is a very powerful technique that is of interest if the bottleneck is the fragment shader. The principle is to render the scene once and instead of producing the final fragment values, only scene data is recovered (normals, world position, materials, ...). The shading is then performed in a successive pass, where the extracted data is used as input. In practice, this is done by drawing a single quad. The shading computations are thus avoided

[1] Graphics hardware usually supports early Z-culling that hierarchically performs depth/stencil tests and delivers a strong speed-up. Unfortunately, this only works if the depth is not modified in the fragment shader.

for any hidden geometry, they become much more coherent and, therefore, more efficient. This method relates to our initial idea of capturing information from the scene and storing it in a GPU accessible way.

The actual speed-up of MRTs come from the fact that fragment processors are working in groups (e.g., to provide derivative information). This introduces various penalties, for example, a sub-pixel triangle will keep all except one processor of a whole group idle. If deferred shading is used, all can work at the same time and benefit from coherent computations that can be exchanged between neighboring pixels.

4 Geometry Representation

Geometry Representation: *Basically, there are two possibilities: flat (n successive vertices define a primitive), or indexed (two buffers, one containing indices, the other the actual vertex data). In our tests, the second solution is not only more memory efficient, but also delivers increased performance. The reason is that the GPU can reuse computations of the vertex shader for several triangles.*

OpenGL offers the so-called ***immediate-mode*** where vertices and primitives can be send directly to the GPU. This does not make sense if the transferred data is the same in each frame[2]. In this case, it is much more useful to store the information directly in the graphics card's memory and thus only transfer it once. The mechanism for this are ***Vertex Buffer Objects (VBO)***. These are zones in card's memory that contain the data to be rendered. The only CPU interaction is thus a call that tells the graphics card to process the data at a certain memory location.

The idea to free the CPU's workload is also reflected by the introduction of ***instances***. Here, one single CPU call triggers multiple processing of a VBO. To avoid that, each time the same processing occurs, a primitive ID is accessible in the shaders. This technique is particularly useful when crowds of similar objects need to be rendered.

There is another advantage of keeping the geometry on the GPU. On newer cards, feedback mechanisms exist that allow to reconnect the output of a geometry shader to the vertex shader and iterate on the data several times.

5 Hardware

In previous generations, there was a clear distinction between the type of processors on the card. Some were fragment-, others vertex shader units. Recently (as of the G80 and before on the XBOX 360), this no longer applies. Today's stream processors can be scheduled to work on all three shader interactions depending on the workload of each. Further, instead of working on vector data (usually 4-tuples in previous generations), each processor is responsible for one floating point value. This much more general description also gave rise to CUDA, which is a programming library that allows to interpret the graphics card as a parallelized multi-core stream processor. It seems obvious that future development will follow this trend of generalization. In fact, NVidia just released a card for scientific computations named TESLA. Basically, it is nothing else but a G80 without the graphics components. Nevertheless, I personally doubt that the graphics

[2]In fact, it never makes sense and this is why this mode is soon to disappear in OpenGL 3.0. In DX this never existed.

pipeline will vanish soon because some of its performance relies on very specialized solutions. For example, scan conversion is efficient because of particular units optimized for this task. Other units are responsible for texture fetches and caches, it will be hard to replace them by generalized mechanisms. All this does not exclude the possibility of adding features and only time will tell what graphics hardware is going to be.

Part II

Simplification

"You gotta quit waiting for something to happen and start doing something about it."

Proverb

Chapter 3

Preprocessing

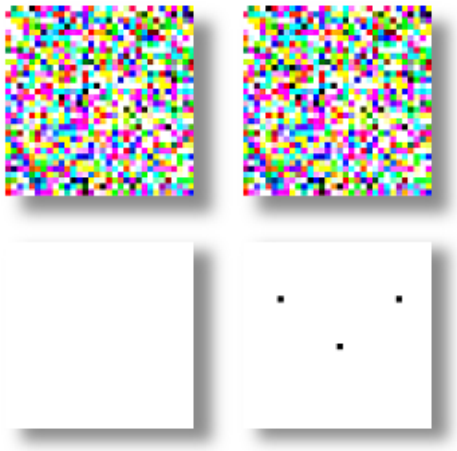


Fig. 3.1 : *Upper row and lower row have the same difference*

Ground truth is an expressions that should be used with care because after all, our scene is only a representation that might have little *truth* to it. All is represented by numbers, that are not even stored at full precision[1]. In this sense, there might not even exist something like a ground truth at least not one we can compute. But it goes further than objects, in fact, this concerns algorithms too. Not because the computations are not precise, but also because many of our realistic models (for example for illumination) are based on physics. Physics model the world, they do not define the world. If everything is an approximation, it is definitely interesting to investigate what an appropriate representation is and how we could benefit from this fact. Most of the following chapters in this dissertation will exploit this principle.

One problem with approximations is that they always come at some cost. After all we cannot say: "Let's make it simpler!" and then expect to obtain the exact same outcome. Or can we...? We have just said that there is an inherent precision limit on a computer. For example, basically each pixel on the screen is containing a color. So all that is smaller than a pixel usually disappears. If we thus perform changes that lie below this visible level, our approximations will remain mostly invisible[2]. This is the motivation to establish error bounds and to measure the committed error.

How much will an approximation deviate from the original? In general, it is difficult to answer this question. An earlier question should be: What was approximated? For example, imagine a table, where the table desk consists of many coplanar triangles. When all these triangles are agglomerated, the visual appearance is exactly equivalent, although the geometry differs. The situation might be different, if values are interpolated from the vertices. All of a sudden, the tessellation might play a role. Therefore, it is important to have a measure of equivalence. This measure is the real assumption we need to make. If it is not chosen reasonably, we might end up with something that perfectly respects an error bound, but might not be what we expected.

[1] Of course, there are libraries to do this, but then still only for rationale numbers or with symbolic calculus that is rarely efficient enough for time consuming tasks.

[2] Mostly - as we said before graphics hardware performs center-accurate rasterization, thus the error remains around one pixel. Also, the discussion here does not address aliasing.

In the case of our work in Chapter 5, we would like to have a measure that tells us how similar two objects are visually. Unfortunately, we do not yet understand all the processes of our visual system and thus such metrics are only in a rough beginning stage. On the other hand, it is still possible to achieve meaningful error measures. We can bound the visual appearance by ensuring that the appearance on the screen is close to the original one. In the end, the understanding of the visual system would only allow us to compress even more. The Figure 3.1 shows such an example. The error in the max-norm of this two images is exactly the same, but the difference on the lower row is more visible than on the upper. Nevertheless, if the maximum difference had been chosen smaller than the perceptual contrast, or smaller than the contrast that can be displayed on the screen, a perceivable difference like in the lower row cannot occur.

In our work, we will limit the maximum angular deviation which will thus bound the projection distance of the points. Unfortunately, this still has its limitations, for a checkerboard, this will not prevent artifacts due to aliasing, but this would also be true for the original model. Another interesting question is whether such accurate bounds are even necessary.

Of course heuristics often do a good job and in practice this is what would often be applied. More accurate computations can also be more costly, although, it has to be pointed out that closed form solutions like the one we present in the next chapter are often efficient. Nevertheless, this is exactly our point: cost should not play too much of a role in a preprocess and if we have the supplementary time to spend, so why not use it[3]? It is only during preprocessing that this augmented computation time is available, so we should use it wisely.

Working on exact error bounds is also important for other reasons, it gives confidence concerning the quality of the output and further it increases knowledge. It is better to degrade a solution than to generalize a hack. Also, knowledge might at long term lead to even more stable, efficient and reliable results or open the road for new approaches or areas of application that really make a difference.

[3]This reminds me of a story David Baraff once told. He said that he found a bug in the Pixar system, which caused a reboot every night and restarted all rendering jobs again. In consequence, any rendering that was started in the evening would basically take twice as long. They fixed this bug and nobody realized that the system got any faster. The program only needed to finish a rendering of a shot per night.

"There's an old saying about those who forget history. I don't remember it, but it's good."

Stephen Colbert

Chapter 4

Previous Work

As mentioned in the introduction, this thesis touches necessarily many areas. We are interested in how complex scenes can be adapted and interpreted in general contexts ranging from display, physical phenomena, visibility, to even collision detection.

In this chapter we will concentrate on simplification. It represents probably the most common way to address complexity.

Simplification is more than just a reduction of geometry. Simplifying a model can have many applications. It filters the input and makes computations more stable. For several level of details, it might also allow to perform hierarchical derivations. Also detail persistence can indicate importance of certain regions.

In this chapter, we will mainly focus on geometric and visual equivalence. This means that we will explain what measures are used to ensure visual fidelity. One typical way is to bound the geometric deviation. This road is taken by many algorithms, but one observation we made is that many rely on heuristics. Several approximate criteria are mixed together to define a measure of quality that might not necessarily provide an accurate error bound. To remedy this inexactitude many approaches add special silhouette terms or saliency measures to improve the simplification's appearance. But these only blur the general shortcomings. To some extent this also shows one deep problem, which is a missing metric that allows to classify the committed error. Of course, an ultimate metric would be based on the human visual system and assure that the perceived object looks similar, but an accurate version of such a metric might never exist and even depend on each individual. Further, the understanding of the involved procedures are currently still on a relatively low level. The only general way of achieving true fidelity is to limit the maximum deviation.

Fig. 4.1 : *(Courtesy of Luebke et al. [LRC*03])*

Our error bound aims at this goal. Interestingly, even in this purely geometric setup the behavior of the simplification is conform to our expectations. Far away objects are more aggressively simplified and even silhouettes are naturally handled. Furthermore, the fact that geometric considerations are at the basis might allow applications in other areas that are related to display-like queries, such as shadows.

In the following chapter we will review many classic, but also some newer simplification algorithms with respect to the question of how errors are measured. We decided to give a more detailed description of many algorithms, instead of a more exhaustive overview. The description results in a better insight into where the approximations intervene and gives a glimpse at the history in this exciting research field.

The difficulty we find ourselves in is that the error bound can often be a consequence of small choices and we cannot provide a complete explanation of all algorithms. Books have been written on this topic. Therefore, the text might be a little steep for beginners, but it should be seen as an overview and a recall for the slightly advanced reader. No previous knowledge related to simplification is needed, but a common familiarity with mathematics and mesh manipulations is assumed. Further, these more in-depth descriptions reflect the complexity of this topic and underlines the problems of bounding an error. Many approaches result in bounds that are valid only in very precise situations, less work investigates more general directions. The second purpose of this overview is to show the large variety of factors that can and could be integrated in error bounds and the ingenuity of many methods.

This overview is not crucial for the understanding of our contribution on error bounds in Chapter 5, which will introduce all necessary details. This part could thus be skipped, but one would miss some of the problems, and small mistakes that have been made, which motivated our research.

1 Simplification

Clark [Cla76] made a very simple observation in 1976: a slow decrease of geometric complexity with distance can have little impact on the final rendering and improve performance.

Further Reading: *Simplification is an important topic in computer graphics and has received much attention. To give an exhaustive description and a complete survey of all existing methods is difficult and* ***not*** *our goal. For more information, we refer the interested reader to the book by Luebke et al. [LRC*03], as well as Heckbert and Garland's course on polygonal simplification [HG97]. The tech report by Erikson [Eri96] is a nice complement, as well as his thesis that gives a more complete presentation of many algorithms [Eri00]*

Back in those days, this was an important insight as it gave the possibility to display more content on the screen than previously possible. At that time, flight simulators were in the center of attention. For these, especially the closely related topic of terrain simplification is of large interest to improve the display of the surrounding environment. The usual assumption is that the terrain is a meshed heightfield. In this chapter, we will focus less on such specific solutions and describe mostly algorithms aiming at the simplification of general meshed models.

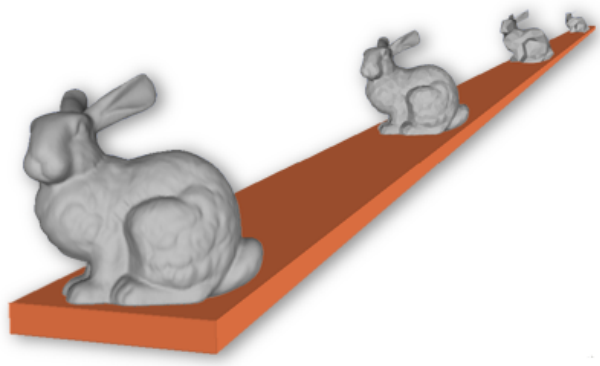


Fig. 4.2 : *Simpler Meshes With Increasing Distance (Courtesy of Luebke et al. [LRC*03])*

Coming back to Clark's discovery; even though of huge importance, there was a catch to the observation: how do we choose and how do we make such models? No automatic approaches existed nor an exact idea of what it meant to have a *slow decrease of geometric complexity with distance*. Clark, himself, suggested to rely on the size of the screen projection of a bounding box. This does not ensure any quality criteria, especially if models are handmade. A new research field was born.

Many different algorithms have been proposed since then. All share the possibility to perform simplification with variable aggressiveness. This allows a trade-off between fidelity and speed with respect to the original input, for some definition of *fidelity*. Each approximation is referred to as *level of detail* (LOD) and a set of such representations is referred to as an *LOD hierarchy* [ZG02].

It is not always possible to establish a continuous relationship between the different LODs. But in practice, continuous geometric transitions are often less adapted to graphics hardware (compare Section 7), which makes it more interesting to switch directly between the representations. Consequently, a natural question that arises is how we can assure that such a switch remains imperceivable. A common way to address this problem is to conservatively bound the deviation of the simplified from the original model for a given viewing distance. If this deviation is small enough (usually of the order of a few pixels as seen from the observer), the model is considered valid for this viewpoint. The involved heuristics can be rather coarse and use a variety of factors (deviation close to the silhouette, overall distance, size of the collapsed edges, *etc.*...).

Other than for visual reasons, simplification also plays a role in other fields. For example, collision detection uses very particular algorithms that combine geometry and motion [CS08]. Biological

computations often require simplification in order to derive approximate shapes of molecules that are then matched one against the other. In this context, the exact maintenance of tunnels might be necessary, which cannot be achieved with arbitrary techniques.

This chapter mainly focuses on the context of visual equivalence, but the reader should keep in mind that other applications exist. Our work in Chapter 5 examines the *exact* amount to which a model can be simplified under a given viewing constraint. The solution is general in the sense that simplification is described by a mapping function of the original model's points onto the simplified representation.

Here, we will present some of the important algorithms in the area of simplification and how they address the approximation error.

2 Classification of Simplification Algorithms

There are many possibilities to group algorithms into different classes based on: their input/output, whether they are topology preserving, error-bound or approximate, memory efficient, time efficient, attribute respecting, viewpoint-dependent, and many others. As we are interested in the error control during simplification, we discuss previous work in this context. We introduce the distinction between ***basic simplification***[1] that does not take a viewpoint, but only a tolerance threshold as a parameter; ***viewpoint-dependent simplification*** that is providing special structures and criteria to choose an LOD at run-time; and finally ***viewcell-dependent simplification*** which is establishing an adapted simplification for a given region in which the observer is allowed to move.

Fig. 4.3 : *Basic Simplification* *(Courtesy of Cohen et al. [COM98])*

Basic simplification allows to create an LOD hierarchy but it is not always easy to obtain an error bound. Typical examples are remeshing applications, where geometry is reconstructed according to a higher-level interpretation of shape. This is usually based on an estimation of the underlying manifold, which is considered smooth and thus differs from the original mesh. This is a major issue with complex implications that we also encountered in the context of our work on vector art [EWHS08]. In the case of a proven geometrical deviation, conservative LOD switches are possible at run-time by projecting this error onto the screen. If no such bound exists, a post-evaluation is needed to estimate the actual error, which is a non trivial task that is either performed by hand or based on a sampling process [CRS98, LT99].

The second class of methods are view-dependent simplification algorithms, that we further subdivide in **viewpoint** and **viewcell** evaluation. The first is often related to hierarchical structures

[1] *basic* is not a judgement

that are chosen or transformed on the fly [Hop96, Hop97, XV96, ESV99, ESSS01a]. The latter usually involves alternative representations which are created according to a set of viewpoints referred to as *viewcell* [SDB97, DSSD99, JWS02, JW02b, WM03].

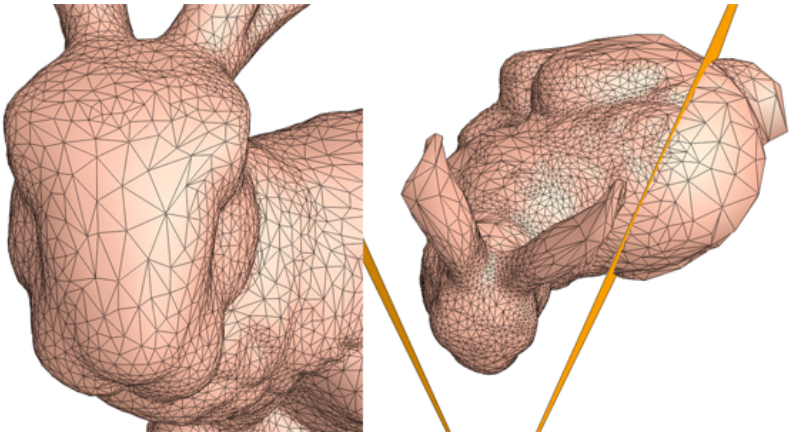


Fig. 4.4 : *View-Dependent Simplification* *(Courtesy of Hoppe [Hop97])*

The viewpoint evaluations have the advantage that only a single (the current) view needs to be taken into account (Figure 4.4 shows an example). On the other hand, to ensure interactivity, the evaluation needs to be performed at high speed, leading necessarily to rather coarse and approximate criteria. The viewcell evaluation is simple to execute at runtime, but a set of viewpoints needs to be considered during the simplification process. Finding a valid representation that respects an error norm for this precise context is challenging and little work exists on this topic. Our contribution in Chapter 5 concentrates on this question. In particular, it allows to establish a quality test similar to [CRS98], but for a view-dependent context.

Finally, there are algorithms that perform simplification by exploiting the particular nature of the input object (city models, trees, water surfaces, etc). Often, this affects the way a model is represented (what coined the term *alternative representations*). We will not investigate them closely as they are usually very specific solutions that do not generalize easily, but this restriction also makes them particularly efficient with respect to their scope.

3 Basic Simplification

3.1 Voxel Clustering

One very early approach to simplification was based on voxels. The basic idea is to place a model in such a grid and to reconstruct a simplified representation by regrouping vertices that fall in the same cluster. A resulting advantage of these methods is that a smooth interpolation between two successive levels of details is straightforward by moving all vertices to the respective position in the next representation. The resolution of these voxels is directly related to surface deviation and can thus be used to compute LODs of different quality. Unfortunately, the uniform structure makes these approaches less appropriate for small error bounds. Further, the algorithms cannot exploit coplanarity for tesselated flat objects.

Rossignac and Borrel [RB93] used this uniform grid to simplify. The basic approach is to replace all vertices that fall into the same given voxel by a single vertex. Collapsed triangles (those fully contained in a voxel) degenerate to points, triangles with one collapsed edges are transformed

into lines and redundancies are removed. This relatively simple method was particularly fast compared to its competitors and dealt with arbitrary topology, even triangle soups.

To find a good representative per voxel, for each vertex, an importance is computed based on an estimation of its contribution to the shape. It is a combination of two elements: first, whether the vertex is adjacent to larger faces (the authors rely on edge lengths to estimate this value), second, the probability of the vertex to be part of the silhouette. The latter, was estimated with an ad-hoc formula ($1/\theta$, where θ is the maximum angle between all pairs of adjacent edges). The maximum or a weighted average based on this importance is then chosen as a representative point for the entire voxel.

Despite all strengths, the method also has weaknesses. Collapses can lead to visual artifacts and a slight perturbation of the grid might lead to a very different outcome due to aliasing, meaning that an element that lies close to the border of two voxels might *jump* quite far.

Low and Tan [LT97] realized this aliasing problem and suggested to perform a clustering with *floating cells*. For a given vertex all neighbors are regrouped if their distance is small enough, instead of enforcing a static structure. Degenerated primitives receive an adapted width/size as well as a normal oriented towards the observer. During the grouping process, the vertices are treated in order according to their importance until all vertices are part of a cluster. This leads to improved detail around the important elements of the model. They also show that a better estimate of the probability of a vertex V to lie on the silhouette is $cos(\theta/2)$. The formula is derived by assuming that the local surface at V is a cone of angle θ. To avoid jumps further, a vertex that falls into several clusters is not attributed to the one with the highest weight, but to the closest cluster. This effectively reduces stretching of triangles and better preserves details.

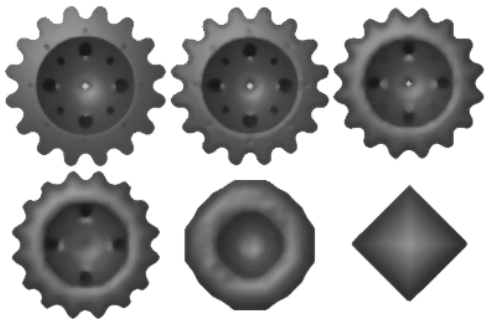


Fig. 4.5 : *LODs From [HHK*95]*

This last observation was also made by Schaufler and Stürzlinger [SS95] who point out that averaging vertices leads to a loss in size and quickly collapses the object. A better choice is a vertex distant from the object's center. Their approach proceeds in two steps, first a tree is constructed by iteratively clustering vertices based on the distance. Every vertex pair is replaced with the average position. The levels of detail are then constructed via a tree traversals, where each node is examined and if the distance of its children is larger than a user provided threshold, the descend continues, otherwise the vertex in the cluster that is the farthest from the object's center becomes the representative.

A different way to deal with the effect of aliasing, when transferring a model into the voxel grid, was presented in [HHVW96] which is a follow-up of their earlier article [HHK*95]. The idea is to compute density values inside the voxels. This is based on a sampling strategy that determines for each point whether it lies in the interior of the model. In consequence, it imposes the model to be watertight/solid[2]. Once the density is derived, a filtering process smoothes the values leading to a multi-resolution representation. The final triangulated model is obtained by applying a modified marching cubes [LC87] algorithm that creates an adaptively refined mesh. Even though

[2]Watertight will be discussed in detail in Chapter 7. It means that each point allows a consistent classification as being interior/exterior based on the Jordan Theorem.

the method is mathematically sound, once a density is obtained, the first step involves standard point sampling and might thus lead to false results.

3.2 Primitive Decimation

Placing an object in a voxel grid has one very significant disadvantage. Basically, the entire information about the actual mesh is lost. Instead, it can be seen as a sampling process at a very coarse scale. So instead of ignoring the actual shape and connectivity, many approaches work directly on the geometry. To simplify the representation two basic mechanisms exist:

- **Removal** remove a vertex/triangle and their adjacent structure, then close the resulting hole
- **Collapse** two vertices to a single one and move the resulting new vertex to an appropriate position (one of the extremities, the center, a general error minimizing position).

The major challenge is to track the committed error from the original mesh to the simplification. In the following sections on primitive decimation we will investigate the various possibilities.

We will also see, that one particular tracking will actually motivate further voxel solutions. These so-called error quadrics will allow to capture geometric shape information that allow simple summations making it very appropriate to store in voxels to describe an average shape appearance in the voxel cell. This possibility is key to eliminate memory requirements and we will discuss this aspect in Section 3.2.4

3.2.1 Vertex Deletion

Schroeder et al. [SZL92] introduced the idea of vertex deletion. They test the distance of each vertex to an average plane defined by its one-ring (that is all its adjacent vertices). If this distance is small, the vertex is deleted and the resulting hole is retriangulated. For this, the one ring is projected on the average plane. Obviously, this can lead to self intersections of the involved edges. In this case, if local topology changes arise, the deletion of the vertex is simply forbidden. Successively, vertices are removed until the mesh has reached the target number of primitives. It has to be pointed out that the error criterion in form of a plane distance is only based on the current mesh, not the original. This means that the result does not allow conclusions on the quality of the final approximation.

This was one of the motivation for Ciampalini et al. [CCMS97]. They maintain a global error bound. Their solution is to sample the new triangles and evaluate the distance to the original surface for each sample point. This would be very costly if performed against the entire model, therefore, they propose to sum up the deviations of previous simplification steps. These deviations are derived via a sampling process. Because the pure sampling itself seems not to lead to sufficient results, they add a second heuristic that tries to match each deleted vertex with a point on the new surface. For these points, a supplementary deviation value is computed and the maximum of all these deviations is combined with the previous error bound. Once a hole is filled, the algorithm optimizes the triangulation locally via edge flips. A whole series of evaluations tests the quality. Besides distance sampling, the aspect ratio and a volume/area optimization are used. The volume change is analyzed by evaluating the tetrahedra defined by the triangles and

the deleted vertex, the smaller the volume change, the better. An observation that will be of high importance in the algorithm suggested in [LT98]. Of course, due to its sampling nature the approach remains approximative.

3.2.2 Plane Distance Error

One of the first real upper bounds for geometric deviation was presented by Cohen et al. [CMO97]. Their technique maintains a correspondence between different levels of detail and is able to establish an upper error bound on the deviation. Furthermore, it allows to track information from the original to the simplifications via mapping functions.

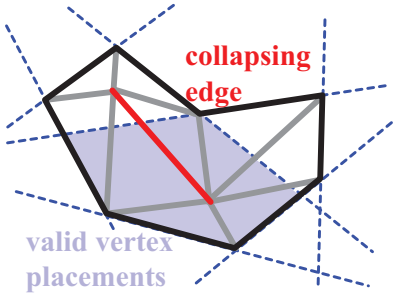


Fig. 4.6 : *After an edge collapse, the new vertex needs to be placed so that fold-overs are avoided.*

The principle are successive edge collapses that are performed in a plane that is chosen such that the one-ring projected on this plane does not exhibit any intersections. This is similar to the condition needed by [SZL92] and [Tur92]. They describe an optimization algorithm to find one such direction. If a projection direction is found, dynamic programming is used to place the new vertex in the plane such that there is no fold-over in the projected geometry - this condition is equivalent to placing the point in *front* of all half-spaces defined by the edges[3] (see Figure 4.6). This also avoids flips of texture coordinates, as long as they are continuous in this region. One degree of liberty remains: the vertex can be moved along the projection direction. A minimization process finds the optimal position of the vertex by comparing to the previous geometry. Working with a projection direction assures that the surface does not locally fold back and allows to recover vertex properties via interpolation such as colors and texture coordinates.

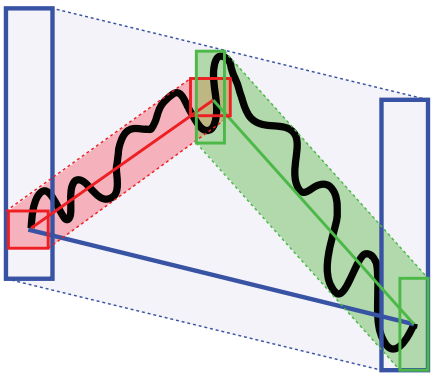


Fig. 4.7 : *Conservative Estimate*

In order to establish an error bound with respect to the original mesh, a conservative upper bound is introduced (see Figure 4.7). Each triangle t stores a box b_t indicating a deviation (one lower level is shown in green and red). b_t is such that the original surface is included in the volume defined by the Minkowski sum (or convolution) of b_t with t. Each newly created vertex is first placed in the plane regardless of the error. Because the mesh is piece-wise linear, it can be shown that the deviation only needs to be measured at the intersections in the projection plane between all previous edges and the new triangles. A new tight englobing box (blue) is created in such a way that, when swept along the new surface, it will include all the boxes of the old one. The final error for a triangle is then given by the corners of this box. This leads to an overestimate because only the maximum deviation for each triangle is stored.

The method introduces another interesting concept: *lazy evaluation*. If an edge is collapsed, all adjacent elements should recompute their cost. *Lazy evaluation* means that the edge is only

[3]The so-described region corresponds to points with an unobstructed line of sight to all other local vertices

flagged as *dirty*. The cost is recomputed solely when the edge is effectively chosen to be simplified. This is particularly interesting for this approach as the evaluation is relatively expensive. To ensure that the geometric bound is respected, the maximum difference of all criteria (e.g., color, texture, geometry) decides on the cost of an edge collapse.

In a follow-up paper Cohen et al. [COM98] rely on the same framework. The major change is to replace the error metric by a maximum deviation in texture space. The idea is to measure the distance of a simplified point to its original correspondent. This is done by transforming it to texture space and then by remapping it from there on the original model. The observation that only the intersections in the plane need to be investigated still hold, as the texture coordinates are also linearly changing values. This allows to create texture maps that encode normals, colors, and of course, the original texture. The deviation in screen space can be limited as before.

Cohen et al's solution replaces the algorithm by Guéziecis [Gué99], which is now mostly interesting from a nostalgic point of view - it was a very early approach ensuring an upper bound on the deviation. It also uses edge collapses, but at each vertex, a sphere indicates the current deviation. These spheres are maintained, such that the volume (resulting from a linear interpolation over the triangle) fully englobes the original surface. Due to their uniform nature, spheres have a tendency to grow faster than bounding boxes. This is obvious when the simplified surface's distance to the original alternates; the radii will increase rapidly, whereas bounding boxes capture this.

3.2.3 Quadric-Based Error Metrics

Previous approaches used coarse bounding volumes to track the error, a different way is to try to maintain information about the surface.

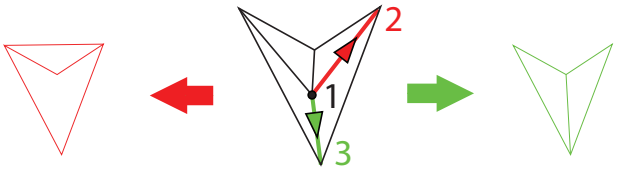


Fig. 4.8 : Triangle Flip - *Merging vertex 1 and 2 gives an overlap, merging 1 and 3 is valid.*

Ronfard and Rossignac [RR96] suggested to iteratively collapse edges based on a plane distance error criterion. They move one of the extremities onto the other and evaluate two local cost functions. The first tends to prevent triangle flips (compare Figure 4.8) and is user weighted. The second limits deviation from the original mesh based on the distance to stored plane equations. At the beginning, each vertex contains the plane equations of its adjacent triangles. Whenever a collapse is performed, the plane sets are merged.

This leads to good results and ensures an upper bound, but there are two drawbacks. First, the algorithm is susceptible to small noise because the plane equations are infinite, whereas triangles are not and thus, the propagated error can quickly become large even when a valid approximation exists (Figure 4.9 shows such an example). Second, the evaluation can become slow when the number of plane equations increases.

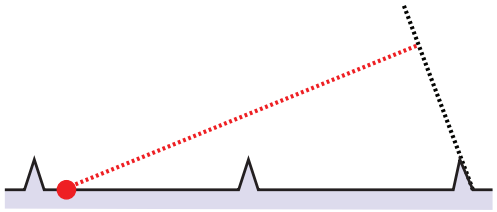


Fig. 4.9 : Error Bound Problem Using Plane Distances - *Large errors (red, black) come from small details, even if a close (here horizontal) solution exists*

This second point is what Garland and Heckbert [GH97] mainly address with their algorithm. It is based on a

similar idea but avoids some of the problems in [RR96]. Instead of storing plane equations, the authors realized that quadrics can be used. Each of these quadrics describes the square distance to a plane (with homogenous coordinates this is a 4×4-matrix). The key insight is that the sum of these quadrics leads to a quadric that allows to measure the sum of the squared distances. The fusion of the "plane equations" of two vertices can thus be reduced to a simple summation, leading to a constant time method. The gained speedup is tremendous. Still today the resulting software QSlim is one of the fastest methods available.

One problem of the quadrics, with respect to the real set of planes, is that the sum can be overconservative. An example are coplanar triangles. When fused, the quadric error increases even though the planes would coincide. This also introduces a dependency with respect to the tessellation of the model.

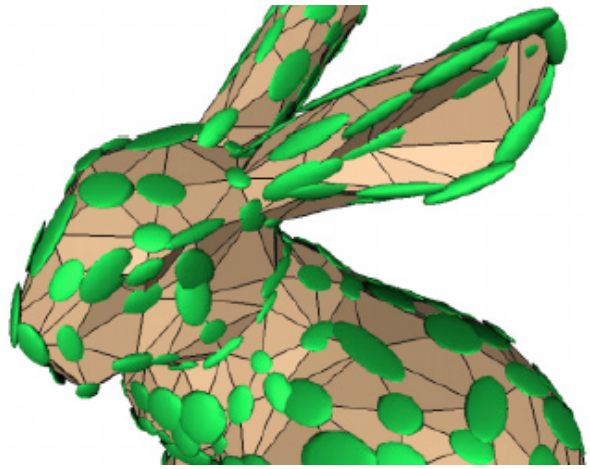


Fig. 4.10 : *Visualization of Error Quadrics (Courtesy of Heckbert and Garland [HG99])*

Another important contribution of Garland and Heckbert's work is the positioning of the new vertex. It is chosen meaningfully: according to the minimum of the quadric. Only when several LODs are derived, it can be of interest to keep the original vertices to reduce storage.

Their follow-up paper [GH98] involved color as well as texture coordinates by extending the quadrics to higher dimensional spaces and they discuss how to treat boundaries by adding supplementary orthogonal quadrics. The problem here is that the extension to a higher dimensional space necessarily creates a much higher cost for the quadric storage and evaluation. Unfortunately, the paper does not provide information on the performance for colored cases and the software is currently not available.

Bounding the error during simplification, especially for the second algorithm, seems difficult. Color and position are typically attributes that cannot be meaningfully combined in an automatic way, but even for the original approach the deviation cannot be easily computed.

Erikson and Manocha [EM99] present a different way to maintain attributes. They develop several error criteria that are weighted together to yield the final error bound (geometry, normals, colors, texture,...). The geometric error bound is still based on quadrics, but this time weighted by the adjacent face area, which penalizes small faces. It should be mentioned, that contrary to intuition, this does not lead to an independence of the tessellation because of the greedy fashion edges are collapsed. The algorithm proceeds like [GH97], but adds penalties for the attributes in the error term via a summation. This allows us to bound the maximum error of each term. Another important contribution in this work is the way virtual edges are introduced. A collapse can be performed between non-adjacent vertices provided that the distance and the change in area - which is computed from the previous and merged configuration - are smaller than some user-defined τ and $\tau^2\pi$ respectively. Once no more simplification steps are possible without breaking the error bound, τ is multiplied by a factor of 2. It is important, that this threshold is only successively increased. This first favors real edges, then the ones in immediate proximity before relaxing the constraint further. This differs from [GH97], where the possibility of virtual edges

was mentioned but only based on a simple distance criterion. Of course, such a test introduces a performance penalty but increases the aggressiveness of a method. Erikson and Manocha also provide a heuristic global error formula: the weighted (by adjacent area) average of all vertex errors multiplied by 10. This value is used as an estimate for switching to the next LOD. The method leads to no accurate bound.

It is disappointing that despite good practical performance no error bounds exist for the quadric methods. This motivated Heckbert and Garland [HG99] to present an extension of the original method. This time they weighted, just like [EM99], each plane quadric according to the triangle's area. With this modification, it is possible to derive a relationship to curvature of an underlying manifold if the triangle size converges towards zero. Of course, this gives no useful error-bound - and only applies to the case where virtual edges are excluded - but, they show that the quadrics give a local surface approximation. Edges will tend to align with the principal directions of curvature, which somewhat explains the good behavior. It is an interesting observation and we will see in Section 3.5 that edge placement is of high importance with respect to approximation quality.

One drawback of the quadric method is that the memory cost is usually elevated and Instead of storing an error-quadric (4*x*4 matrix) to be able to track the deviation from the original model, Lindstrom and Turk [LT98] use a system of constraints to implicitly keep the surface close to the original. They try to perserve local area and local volume. To achieve this goal, the newly added vertex (after an edge collapse) is restricted such that the volume remains locally constant. This condition is similar to restricting the vertex to a plane. This ensures volume consistency only for the *entire* one ring. Each adjacent triangle's slab might still vary. To find the final vertex position, an optimization step is performed based on the *squared* volume changes of each adjacent triangle. In the case that the solution remains under-constrained, a shape-optimizing term (in form of the squared adjacent edge lengths) is added. The cost of edges relate directly to a combination of these error terms. The resulting mesh is usually of higher approximation quality (with respect to a mean error measure) than the method in [GH97], but the computation time increases significantly. Nevertheless, compared to many approaches, the solution remains relatively fast and it is memory friendly because the only additional information is a priority queue that contains the cost of all possible edge collapses.

3.2.4 Memory Constraints

Even though in [LT98] the memory consumption was already decreased slightly, it still relates linearly to the input model. This prevents the application of each of these algorithms to gigantic models, that would benefit most from simplification.

Lindstrom's first algorithm [Lin00] decreasing memory consumption substantially, works as follows: The bounding box is transformed to a voxelgrid. For each triangle, the error quadrics of its vertices contribute to a grid, which is maintained in a memory efficient way; only if data is added to empty cells a new entry is created. All quadrics are summed-up in their vertices' corresponding cells. Once the entire model is processed, the vertex positions are optimized based on the quadrics in each cell.

One catch is that the simplified model is still constructed in memory thus limiting the output's complexity by the available memory. This limitation is removed in Lindstrom and Silva's follow-

up work [LS01]. Here the resulting mesh is directly written on a disc instead of keeping the result in memory. To achieve this efficiently, a disc merge sort algorithms is used to put the data in a convenient order. This allows to read the elements per voxel sequentially and thus construct directly a corresponding quadric. Basically, only one quadric is kept in memory at a time.

Another interesting idea, in this approach is to create a second quadric, orthogonal to the edge and the normal of each triangle. This was previously used in [GH98], but only for boundary edges. As the model is never loaded entirely in the memory this adjacency information is not available. Putting such quadrics at each edge implicitly penalizes the deformation of boundary triangles as they will not have a *antagonistic* quadric from an adjacent triangle. It is an approximation because all adjacent triangles that are not completely co-planar will create unnecessary and polluting penalty terms.

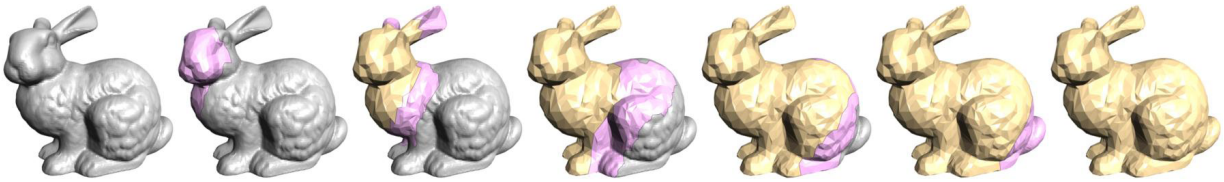


Fig. 4.11 : Processing Sequences

Progressive treatment of large models with adjacency information (Courtesy of Isenburg et al. [ILGS03])

Even though this improves the results, it can still lead to problems if several unconnected surfaces fall in the same voxel. One way to recover the original connectivity is to use streaming techniques that transfers a model by blocks into the main memory [IG03]. The structure allows Isenburg and Lindstrom [ILGS03] to maintain adjacency even over different blocks (see Figure 4.11). Edge quadrics can thus be added only where real boundaries are. They further improve the simplification step by exploiting the adjacency information. All vertices first constitute separate clusters, even those falling in the same voxel. Only during a second step, when triangles are processed, the clusters are collapsed if an edge between two vertices exist. This prevents geodesically distant surface parts from being combined. Once the work on a block is done, all elements that do not interfere with the boundary of the following batches can be written to the disc, keeping the memory cost low.

3.3 Bounding Error by Hulls

Up to here, we have only accurate error bounds that completely ignore the geometry, or accumulations based on plane sets and bounding shapes. The two latter suffer from the fact that iterative approximations quickly overestimate the error and lead to a more restricting penalization of the final output. But there is one surprisingly simple way to ensure accurate error bounds without accumulating approximations at each simplification step. The solution is to define a validity region around the initial object that specifies the exact volume in which the simplification has to remain in order to satisfy the needed geometric proximity.

Such a solution are simplification envelopes by Cohen et al. [CVM*96]. The main idea of the paper is to sandwich the original surface between two offset versions based on a user defined ε value. These offsets can be computed with a numerical scheme that iteratively extends all vertices

simultaneously along the normals and stopping each one, as soon as it enters in contact with the evolving shape. They also present an analytic solution, based on the distance of each triangle to the model and halving it makes sure that intersections are avoided.

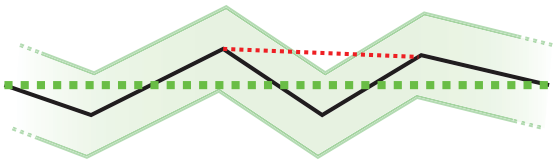


Fig. 4.12 : *Local algorithm gets stuck*

The algorithm is rather simple. All vertices are stored in a queue and processed one after the other. Each vertex removal is followed by a greedy retriangulation of the hole. If an envelope-respecting triangulation is found, the step is accepted and the surrounding vertices are inserted in the queue, otherwise, it is blocked. This local scheme might get stuck (see Figure 4.12, where the green dotted line shows a valid solution for the green envelope that cannot be reached with local collapses). A global scheme on the other hand, where several vertices are removed simultaneously, is expensive. The approximation has necessarily a guaranteed error because the distance of the simplified surface lies between the two offsets. On the other hand, this error is purely geometric, thus disregards color and texture, for which strong deviations can occur. Topology is also maintained, thus limiting the degree of simplification.

As already pointed out by [GH97], these topology changes can lead to a substantially stronger simplification. The idea in [ESV98] is to provide a hull for the topology simplification. Their choice fell on α-hulls[4]. Whenever our two different edges are close enough, a virtual diagonal is introduced between their closest vertices. These edges are then used during a triangulation phase. This effectively removes small holes or spikes from the model. A cleanup step then deletes all triangles that are now in the interior of the object. The algorithm iterates between such a topological simplification and a local simplification based on simplification envelopes [CVM*96]. This leads to visually better results than simple edge collapse operations without topology modification, but the algorithm is costly and the implication of topology modifications on the error not always clear (Imagine, for example, a complex interior of a house. Topologically removing the entrance will lead to a deletion of the entire interior during the cleanup step.)

Fig. 4.13 : *Permission Grids*

Zelinka and Garland [ZG02] used an even simpler method to restrict the deviation of the surface that can also be used with an arbitrary simplification technique. In practice, they rely on QSlim [GH97]. One drawback of Cohen et al.'s approach [CVM*96] is the costly computation of and test against the envelope. Instead, Zelinka and Garland propose a faster solution using voxels (see Figure 4.13).

The idea is to extend all triangle slightly and *raster* them into a voxelgrid, where all touched voxels are marked. The extension assures a conservative overestimate (compare Chapter 7). Only activated voxels will be allowed to contain geometry. A simplification is rejected if it would make the surface leave this discretized representation. The algorithm is extremely fast and due to the voxel distance, the final result is usually close to the original surface.

[4] A nice image to illustrate α-hulls is to think of stracciatella ice-cream. A scoop of size α will eat away the ice-cream while avoiding chunks of chocolate (here the model). Fantasy needs to be stretched a little, when one has to imagine eating ice-cream in the interior of the ball...

From my understanding, I cannot fully agree with the author's claim that the algorithm leads to a guaranteed geometric error bound with respect to the one-sided Hausdorff distance. Then, each voxel should contain a point from the simplified model. Imagine several spheres, one contained in the other, at a distance such that the voxelization leads to an opaque sphere. In this case, any edge collapse is allowed, degenerating the result to nothing. The authors do mention that a two-sided Hausdorff distance would also be possible and propose an extension; all voxels touched by the medial axis are removed[5]. It is not described what happens with elements of the mesh that find themselves in contact with the medial axis (there should be many in a polygonal object). If we assume, that they are not allowed to move during simplification, the sphere example would invalidate any simplification step no matter the chosen error. This is conform with the error bound, but not useful in practice. An accurate skeleton (for the sphere a point) would again result in a disrespected bound. Nevertheless, in practice, it often leads to very good results and a fast execution time.

3.3.1 Energy-Based Simplification

These last approaches assumed that the triangulation is actually somewhat already an important information of the shape. This might often be true, but after all, these triangles were often created with respect to some ground truth that is no longer available. Hoppe et al. produced a whole series of papers on simplification based on an optimization process that starts with a selection of samples from the surface. The most famous in this series is probably [Hop96], which even made it into DirectX as part of the standard distribution. Whereas progressive meshes still derive the simplification from the original model, we will see in the next sections that instead of constructing top-down, bottom-up[6] is an alternative that can have advantages because it is less tight to the original representation.

One earlier related paper is [HDD*93]. Given a set of reference points and an initial triangular mesh M_0, they try to find a mesh with the simplest geometry possible and the same topology as M_0 that best fits to these points. This is done in iterative operations adapting the complexity and shape of the mesh.

The best possible representation is defined by having the lowest energy according to the function:

$$E = E_{distance} + c_{spring}\, E_{spring} + c_{representation}\, E_{representation}$$

Simplification can thus be seen as a gradient descent in *mesh space*. Let's now analyze this complicated error metric. $c_{representation}$ and c_{spring} are user specified constants. $E_{distance}$ is the squared distance of the reference points to the surface. E_{spring} is a spring term, more precisely the square distance of the edges' extremities. This value ensures the existence of a minimum. For example, imagine 3 co-planar reference points, now if E_{spring} is excluded any triangle containing the three points would be minimal. $E_{representation}$ penalizes the number of primitives in the mesh and is simply the number of faces. It is somewhat counterintuitive that the discrete value $E_{representation}$ is part of this (somewhat continuous) equation and we will see later that Hoppe [Hop96] manages to get rid of this term.

[5]The medial axis describes the locus of tangent spheres to the shape. It can be applied also directly in the context of shape simplification. The method in [TH03] mainly aims at removing less important parts of a model by pruning the medial axis, but, in effect, this can lead to simplified models.

[6]top-down: starting from a detailed model and reducing its complexity, bottom-up: construct a simplification by building up a simple model

To find the minimum of this energy function, the method iterates between two modes: First, the number of primitives is assumed constant and only the vertex positions are optimized. Second, the algorithm then modifies the triangulation by performing edge split/collapse or edge swap operations.

For the first step, the algorithm projects the reference points on the nearest surface and obtains a barycentric description of the projected entities. Vertices of the model are then moved such that the "moved" projected points (the movement results from the use of the old barycentric values with the new vertex positions) lead to a smaller distance with respect to the original reference points. This can be formulated as a least-squares problem and efficient solutions exist.

For the second optimization step, the benefit of split/collapse and swap are heuristically evaluated in a local neighborhood around each edge. This includes an optimization based vertex placement in case of a split/collapse. The best move is then selected and the algorithm starts over with the global vertex optimization until convergence. A move can be undone if it leads to folds or local self intersections.

The energy minimization describes a kind of average error, thus exact bounds are impossible. Further, the behavior is not always completely intuitive. Especially the spring energy term needs to be adapted with care. Too high values can lead to tessellation in flat areas, whereas too low values might lead to areas where the triangulation has a large distance to the sample points. To achieve a good behavior, the value c_{spring} is slowly decreased during the course of the algorithm.

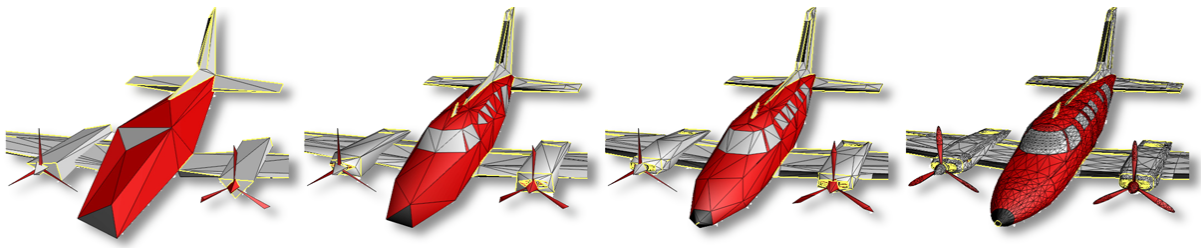


Fig. 4.14 : Progressive Meshes

Progressive meshes is a hierarchy of energy based simplified models. Taking geometric distance, discontinuities (yellow) and scalar values (here colors) into account.

Progressive meshes [Hop96] relies on the previous framework but focuses solely on mesh simplification. The energy function is slightly more complex, but the discrete optimization strategies are much simpler - only edge collapses are performed. An interesting observation is that the movement of the vertices during an edge collapse and its antagonist vertex split allow a smooth transition. It is thus possible to continuously parameterize the transformation from the simplest- to the most detailed representation. In the context of geometry transmission, first, a coarse version can be sent to the observer, before details are added progressively by transferring the corresponding vertex-split operations. This being one of the major applications; the paper describes how to encode information efficiently.

The method is particularly interesting because it allows for usage with scalar values stored at vertices, discrete values stored in faces and even discontinuities where the model's shape exhibits strong angles. The modification of the energy equation is straightforward:

$$E = E_{distance} + E_{spring} + c_{scalar}E_{scalar} + c_{discontinuity}E_{discontinuity}$$

The first two term were retained from [HDD*93] and the needed reference points are sampled from the original geometry (at least each vertex, but also some interior triangle points are used). The newly introduced elements E_{scalar} and $E_{discontinuity}$ penalize scalar attribute differences and mesh discontinuities respectively.

One visible difference is that the primitive number no longer intervenes because each step will simply remove the edge with the lowest energy and thus exactly two triangles. This is efficiently done with a priority queue and by updating only elements whose cost changed through a collapse.

The second change is that the spring constant vanished. Instead E_{spring} adapts automatically during the process. The intuition behind the spring energy was to ensure robustness. As more constraints exist and less primitives are present, the solution is rarely under-constrained, and thus more robust. In consequence, the spring constant is adapted depending on the number of constraints for the one-ring around an edge $\mathcal{R}$. More precisely, its value is selected using a monotonically decreasing function based on the ratio of reference points associated with $\mathcal{R}$ and $\mathcal{R}$'s current number of triangles.

To accelerate computations, the optimization is first performed for $E_{distance}+E_{spring}$ as in [HDD*93]. Only when the minimum is found, the corresponding projected reference points in barycentric coordinates are used to optimize E_{scalar}. This is done in a single linear optimization step based on the square distances of the attributes. $E_{discontinuity}$ can be included in the process in two ways. If a move affects the topology of the discontinuity curves, it is either forbidden, or penalized. The penalty is computed based on sample points from the discontinuity curves and is equivalent to the sum of the squared distances of their projection on the new discontinuity curve. This is the curve-equivalent of $E_{distance}$ and thus embeds a curve-fitting into a surface-fitting problem.

The algorithm does not allow to derive an error bound but shows a good simplification behavior, visible in Figure 4.14. Especially the discontinuity curves help to maintain important features, but the fact that reference points are sampled makes it difficult to ensure any fidelity. This is particularly true for texture coordinates and gave rise to a publication by Sander et al. [SSGH01].

3.4 Face Clustering

Decimation can be considered top-down. Starting from a precise input, consecutively elements are removed until the simplification is achieved. In the following, we will investigate algorithms that work bottom up. Instead of removing details, these algorithms build an increasingly detailed solution.

The main idea of face clustering is to derive groups of faces. The grouping assures that all faces lie close to a plane. Once the clusters are established, their boundaries are simplified and a retriangulation process, rebuilds a mesh. This makes it difficult to accurately control meshing and complexity of the final output.

Kalvin and Taylor [KT96] start with a random seed face. The algorithm proceeds in a dual space where points correspond to planes. Each triangle implies some linear constraints on this dual space which describe the set of all those planes that lie closer than some predefined distance. Each time a new face is added to the cluster, the space is further restricted. If an addition leads to an empty dual set, the grouping is denied. To accelerate the process, the valid volume is approximated. To shape regions more uniformly, a second error component ensures an

equilibrated/disc-like shape of the regions. It is defined by the squared perimeter divided by the total area of the cluster. A union is disallowed if this value exceeds some threshold. On the one hand, the approach effectively bounds the deviation of the surface. On the other hand, many decisions (seed triangles, shape threshold, approximations) lead to various behaviors despite this bound.

Garland et al. [GWH01] use a similar approach. They work on a dual graph where adjacent faces are connected by an edge. Each edge collapse corresponds to the merging of two face clusters in primary space. Each cluster maintains an error quadric which is computed as a mean deviation to a best fitting plane, not unlike the quadric used for vertices. The main difference is that, this time, there is one plane and several vertices. It still allows the same simple fusion by summation. To find the best representative plane of a cluster, a PCA decomposition is performed. More compact clusters can be obtained with the disc shape term from [KT96]. This second error measure is simply added to the overall estimate. The priority for a collapse can thus be higher if the combination with another cluster leads to more uniformly grouped elements. The retriangulation is performed just like in [KT96], but the method cannot bound the maximum error and rather aims at a low average deviation, which in turn enables a faster evaluation.

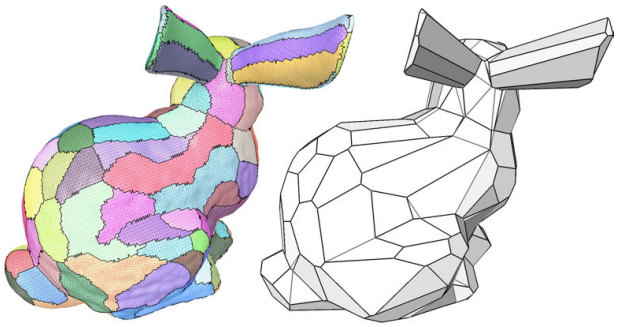


Fig. 4.15 : *The clustering respects curvature*

Cohen-Steiner et al. [CSAD04] present a process that is not directly error-driven and might lead to deviations that, as the authors admit themselves, might even be stronger than for other methods. Nevertheless, the results *look good*, because the algorithm does an excellent job in capturing the essence of an object. The grouping of faces is performed based on a Lloyd relaxation [DFG99]. For some initial seeds, the most similar triangles are added greedily until all faces are attributed to a cluster. For each such region, a *proxy*, basically a plane, is derived, minimizing the deviation. Then, the seed relaxation is repeated from those triangles that best match the resulting proxy inside each region. To measure this similarity, two metrics are presented: $\mathcal{L}_2$, $\mathcal{L}_{2,1}$. The first measures a pure geometric distance error by projecting each triangle's points (not only vertices) on the proxy plane and integrating the square distance. Surprisingly, the latter does not make use of the distance, but solely measures the squared normal deviation. Finding the best fitting proxy for a given region is rather simple - in the second case, it remounts to an area weighted average of the normals of all triangles in the region. This leads to clusters where all faces share a similar orientation, see Figure 4.15.

The algorithm has several advantages: it is insensitive to tessellation and often converges to a solution close to a global minimum of the error norm. It also allows the user to refine the number of seeds or to attribute weights to areas of an input mesh that should receive more details. Unfortunately, an absolute error cannot be provided. In its current version, topology remains mostly unchanged and attributes are not taken into account.

3.5 Resampling

In these last sections we started looking at methods that kind of build a model. Remeshing goes further and is an interesting way to remove complexity and improve the quality of a mesh, even though it is not directly related to simplification. These techniques derive no error bounds on the representation and instead reconstruct a mesh using a sampling process. We will only quickly explain two methods to give an idea of the type of algorithm.

Turk [Tur92] re-tiles a surface by spreading particles over the original mesh using geodesic repulsion forces. Once these points settle, they are integrated into the original triangulation of the original model. Afterwards, successively all vertices coming from the input model are removed iteratively and holes are retriangulated by projecting the mesh locally on a plane[7]. A second test assures that the decimation step does not change topology.

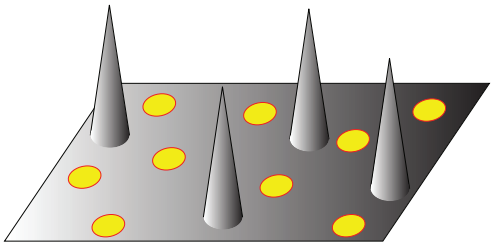


Fig. 4.16 : *Sampling can miss details*

To improve the quality of the approximation, the sampling rate can be increased in regions of high curvature by modifying the repulsion forces of the particles according to the local maximal curvature (a method we also relied on in [SEH08]). Here, the curvature estimate is rather coarse. For a vertex each edge and vertex normal define a cone. This cone is used to fit a sphere of radius according to the edges length. The curvature for the vertex is then defined as the maximum of the values of its edges. Interestingly, the paper also introduces a way to geomorph [Hop96] between higher and lower detail representations by linear interpolation. This assumes that the higher detail model is build by adding supplementary samples to the lower detailed model and fixing the previous ones.

No error bounds can be established because the process does not make use of distance computations with respect to the original geometry. Figure 4.16 shows a pathological case that illustrates this problem. The spikes have a small radius and particles in between might not necessarily capture these details, even when relying on the curvature approximation. But curvature is a very powerful information.

We already mentioned that, in [HG99], edges are collapsed such that the new edges' orientation follows the principal directions. This has been shown to be locally optimal in the sense that it approaches the theoretically optimal behavior presented in [Sim94]. This motivated Alliez et al. [ACSD*03] to directly sample a surface with edges. An overview of the process is shown in Figure 4.17. Their edges result from an estimation of lines of curvature following the principle directions of the curvature tensor on the surface. Basically, these are the directions in which the surface bends most (e.g., for a cylinder, this is the direction along the axis and around the cylinder along the circular cut with a plane orthogonal to the axis). Only in elliptical areas (which have uniform curvature) a point-sampling is used. The method heavily relies on geometric libraries and parameterizes the model using a conformal mapping [LPRM02][8]. The curvature tensor is computed for each vertex and linearly interpolated throughout the triangle. This gives the basis

[7] In contrast to [CMO97], they choose only 13 trial directions to find a projection that will lead to a non self-intersecting boundary

[8] A conformal mapping flattens the model in a way that angles are mostly maintained, whereas area might get distorted. Only areas with zero Gaussian curvature, meaning that at least one principle curvature is zero, can be flattened without distortion.

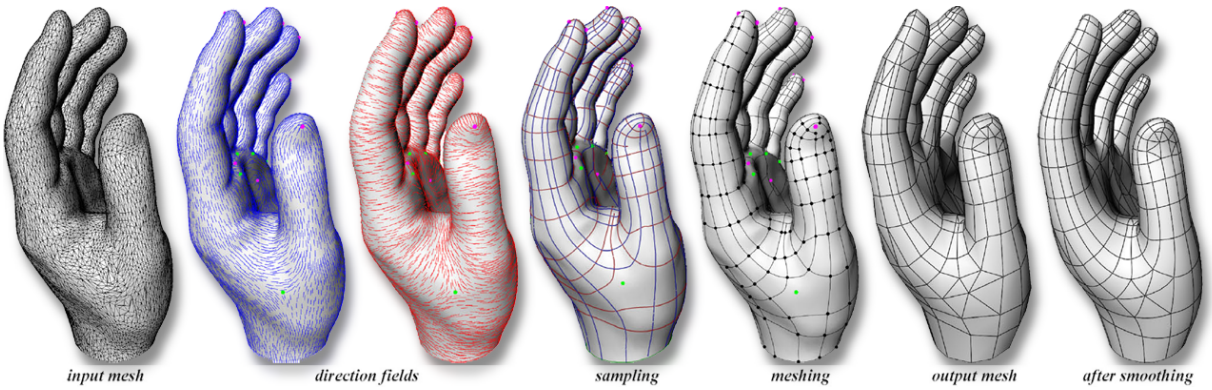


Fig. 4.17 : Anisotropic Remeshing Overview

*The algorithm estimates curvature on the mesh and uses the principal directions to create a network of curves which form the basis of the final mesh. (Courtesy of Alliez et al. [ACSD*03])*

for the curvature line extraction which is obtained numerically by integrating along principle directions and adapting density according to a user threshold and the curvature amount. The resulting structure of intersected curves is a quad dominant mesh because the principal directions are orthogonal to each other. Based on a small set of rules, the final triangle or quad mesh is obtained by simplifying the quad and sampled elliptical areas.

A potential problem is the estimation of the curvature tensor. Although the method is rather robust, the surface should be Delaunay triangulated for a correct estimation and their work uses a supplementary filtering step to obtain more reliable results. In general, it is always difficult to derive higher order information from a piecewise-linear surface representation and especially the derivation of curvature is still an ongoing topic of research. In many cases coarse meshes can result in very mediocre estimates. One surprising example is Schwarz's *Chinese Lantern*. The approximation is achieved by a zig-zagging surface that converges in the Hausdorff distance to a cylinder. Nevertheless, normal field and area are non-converging.

3.6 Image-Based Simplification

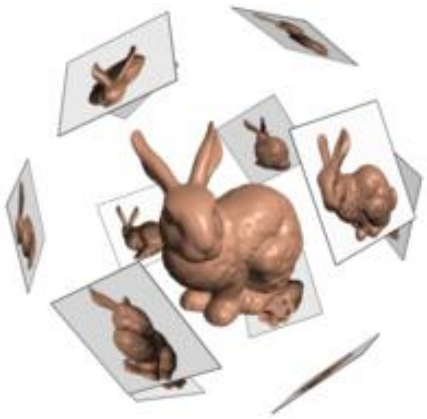


Fig. 4.18 : *Image-Driven Simplification (Lindstrom and Turk [LT00])*

The last section showed algorithms that care less about geometric accurate proximity and try to capture the underlying appearance of the model. Lindstrom and Turk [LT00] had an almost opposing idea. They measure the explicit visual similarity between the simplification and the original model.

They realized that appearance can be captured easily by rendering. After an edge collapse the proximity of the new mesh is simply tested by comparing the images of the new model to those of the original (see Figure 4.18). To avoid a costly rendering of the entire object, they use a local evaluation by restricting the update to the region influenced by the collapse operation. The number of views to evaluate is rather small and, nevertheless, the algorithm becomes very costly, making

it less applicable to complex geometry. For moderately complex meshes, the results seem very good because color, shading, and texture are dealt with inherently. On the other hand, real error bounds cannot be established because of the discrete nature of the sampling process. This sampling can further be problematic when applying the technique to highly detailed geometry.

It is costly to always compare all the images after each collapse, but with a similar technique, some information might be extractable from the original model that is then used afterwards. Zhang and Turk [ZT02] make use of sampled images to create a visibility guided simplification. For each triangle T, a visibility probability V is computed using graphics hardware. V is an integration over several camera positions and initially set to zero. If T is visible for a given camera position, then V is incremented by the cosine of the negative view direction and the normal (this is very similar to ambient occlusion, that we will discuss in Section 7.1). Finally, the resulting value is used to weigh the edge collapse quadrics and the rest of the algorithm is equivalent to [GH97]. Due to the integration of visibility, hidden parts will be more aggressively simplified than exposed ones. It is important to notice that the derivation of visibility is costly, but once done, the simplification is as efficient as Garland's solution. The overall error cannot be bound in this method. Sampling strategies can fail and the limitations from [GH97] are inherited.

In the same context, the idea of using visual proximity is also reflected in the work by Williams et al. [WLC*03]. They presented an algorithm which takes human perception into account. Potentially, perceptional metrics can lead to a much better simplification (e.g., perception of detail is very weak if regions contain much high frequency information like textures. In low frequency regions, e.g., where surfaces are smooth, the introduction of high frequencies results in artifacts that are very evident). This research direction is still very young and much remains to be done. Many aspects of the visual system are not yet well understood. As mentioned before, perceptional simplification is not related to the work in this thesis, and we thus decided to ignore this topic. The interested reader is referred to [LRC*03], which presents a large body of work on this topic and gives more insight into the human visual system.

4 Viewpoint-Dependent Simplification

Viewpoint-dependent simplification takes the current point of view into account and chooses an appropriate representation. Any standard simplification method could be used in this way, hence the large body of basic simplifications we presented. For this, the LOD is chosen according to some distance. If the simplification is error-bound, it is even possible to achieve a conservative error bound for the viewpoint-dependent case by examining the ratio of deviation and distance. Unfortunately, such constructions are very coarse ways to determine an appropriate representation and only work if the observer is not navigating through the object itself. In this section we will investigate algorithms that provide a more continuous and tighter solution with respect to current viewpoint and the error.

Before investigating general algorithms, one particular representation needs to be mentioned: *heightfields* -two-dimensional planar meshes for which a function elevates each vertex along the Z-axis to a certain height. We do not want to discuss the large variety of algorithms related to this topic because we are more interested in general solutions. However, we want to point out one particular solution by Lindstrom et al. [LKR*96]. It is important in our context because it

makes use of an interesting error measure that takes orientation of the viewpoint and perspective projection into account.

The terrain is represented using a quadtree like structure. The authors found a test equation whose sign indicates whether or not a subdivision is needed for the current level with respect to a given threshold. This equation exploits the orientation between the point of view and the vertical direction of the terrain. This allows much more aggressive simplification. Intuitively, a terrain seen from above would not need any elevation and still have the same appearance. Geometrically, the equation results in a bialy (a torus without hole) which we will also encounter in the discussion of our analysis, although the derivation differs. It would be costly to evaluate the equation for all vertices. Therefore, in a precomputation, two distances are obtained and stored that enable a first, fast and conservative error bound for each quad region - if below, a subdivision needs to be applied; if above, no subdivision is necessary. Only if the value falls in between, the bialy evaluation needs to be performed.

Hoppe [Hop97] took this error metric and applied it to progressive meshes. The observation by Lindstrom et al. [LKR*96] underlined that including a direction of deviation results in a better error measure. Hoppe therefore adds a local deviation direction that he assumes to lie along some direction $\vec{n}$. This direction and a deviation distance are acquired from the position of the reference points (used to create the progressive mesh) with respect to the current simplification. To improve quality, a uniform error measure is also attached to each vertex. This also results in a bialy error shape, achieving that the error gets larger whenever the view is orthogonal to $\vec{n}$. $\vec{n}$ is only an approximation and to improve quality another criterion is added in form of a silhouette threshold. The sampling process of progressive meshes does not allow a strict bound. Further, the new metric is only used during evaluation and not during the simplification process, but it results in a good run-time behavior.

For meshes it is a common solution to build a hierarchical structure, where each level corresponds to a different LOD. Hoppe's tree is relatively complex and whenever a part of a mesh is refined, a neighborhood verification is necessary to assure that its current level is compatible with the increase of detail, or needs further refinement.

To avoid these many dependencies Xia and Varshney [XV96] develop a more shallow hierarchical structure the so-called merge tree, which encodes sequences of edge collapsing operations. They define the *region of influence* as all the triangles incident to an edge. At each level of the tree, edge collapses always have non-intersecting regions of influence. This ensures that no folding occurs when applying the vertex splits at runtime. Edges are collapsed until the merge tree level does not admit more. Then the algorithm reiterates on the simplified shape to obtain the next level of detail (see sidebox).

To Block Or Not to Block: *Hoppe pointed out, that even though Xia and Varshney's [XV96] achieve non-interdependence of the simplification steps, this restriction reduces simplification quality. Further, he underlines that enough refinement steps to assure a consistent simplification is not much more expensive. In the presence of currently emerging graphics hardware, it is not sure that this is still the case, because LOD objects should be stored instead of complex hierarchies (compare the short discussion in Section 7).*

Each node of the merge tree stores the maximum distance to the child vertices and the distance to the parent vertex. At runtime, the descent is based on the projected size of these distances with respect to a given threshold. The distances are given level by level, the direction of the edges is not taken into account. The assumption that the projection error of an edge implies a bound on the projection error of a face is not accurate as we will show in Chapter 5, a global

error can thus not be derived. To improve quality, more subdivision criteria were added that are not explicitly described in the text. These include laying more importance on the silhouettes and subdivision based on the lighting situation (e.g., specularities).

A very similar approach was published by El-Sana and Varshney [ESV99]. The main contribution was to overcome the problem that topology remained unchanged. The solution was the introduction of virtual edges, similar to the idea of α-Hulls the same authors applied previously in [ESV98]. LOD's are chosen during run-time based on a creative spline-based error measure (taking the light and the viewpoint into account). Basically the length of a spline with tangents according to the normal and the view-direction are evaluated to decide on the committed error. Again, silhouettes are treated separately with a second normal cone test. The observation that light is an important factor during simplification was also presented in [GLL*03], but the main focus of this work lied on determining visible and shadow sets, not simplification.

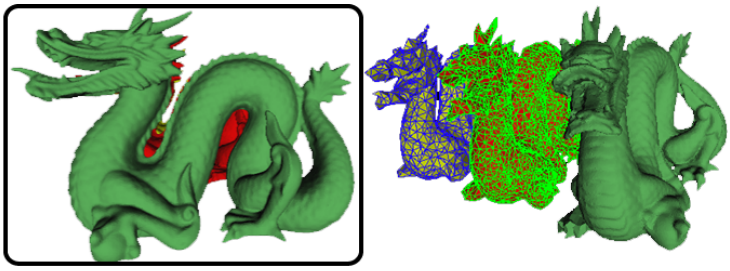


Fig. 4.19 : Occlusion Probability - *in the view (left) leads to simpler dragons (right) (Courtesy of El-Sana et al. [ESSS01b])*

Later, El-Sana et al. [ESSS01b] extended the approach to add visibility information, not unlike [ZT02]. Opacity/Solidity are precalculated and encoded in a grid, based on two heuristics. The first is the projected area of the faces on the six grid cell faces normalized by the grid cell surface, the second estimates a density using a discrete sample set of rays. At run-time rays are tested from the viewpoint and depending on the blocking probability of the traversed cells less geometry is used (see Figure 4.19).

Luebke and Erikson [LE97] present an approach that clusters vertices together and shows similarities with [LT97]. Instead of a hierarchy based on the topology of the mesh, which results in these complicated dependencies, they precompute a hierarchy of clusters of nearby vertices. Each cluster contains the information about the triangles that lie partially in it and are represented by a single vertex in the scene, independent of their original connectivity. During run-time, clusters are unfolded or collapsed, thus *simplifying* all triangles whose corners lie in these clusters. A triangle is removed from the scene when it becomes degenerate, thus reducing geometric complexity.

The error bound by Luebke and Erikson is based on two values stored in each cluster, its extent, and its normal cone. The first is based on a bounding sphere of all vertices in the cluster. During run-time, the screen-size of this sphere is evaluated and, if it is too large, the corresponding node is expanded. Interestingly, this leads to no error bound for the simplification process. A simple example would be a quad that extends far to the left and right of the observer as discussed in Chapter 5. Even if the distance of the endpoints is negligible a collapse can have drastic impact for the current viewpoint. In practice, this might never occur because triangles are usually well-behaved but it is important to be aware. Surprisingly, in [LRC*03] the algorithm's description is modified. Here the sphere's radius is defined by the influence region of the vertex. This effectively bounds the error, but leads to a strong overconservativity. Triangles with bad aspect ratios will introduce large bounding spheres. The second criterion, the normal cone englobing all

the element's normals, can be used to determine whether a cluster is likely to be on the silhouette. In this case it is expanded because sihouettes are often important for the quality of appearance.

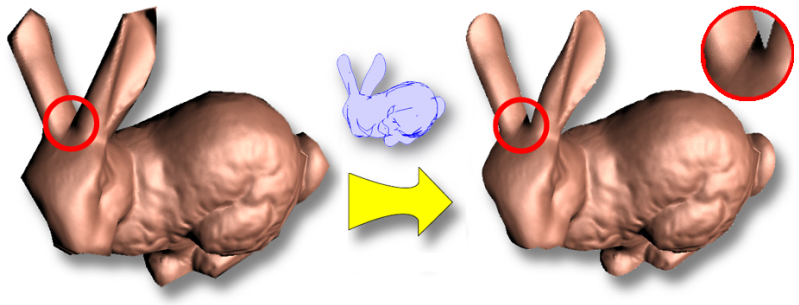


Fig. 4.20 : *Interior silhouettes still show artifacts after clipping*

This observation was one of the motivations for [SGG*00] to introduce silhouette clipping. Here the algorithm uses a very coarse mesh with an applied bump- and texture mapping recovered from the original. The interior of the model looks appropriate due to the image-based improvement, but the silhouette reveals the coarse polygonal nature. Their solution is to derive the silhouette from the model with a rapid algorithm and to use it to fill the stencil buffer to cover exactly the area that the original object would cover (this is done using the winding-rule we also exploited in [SEH08]). Then they draw the coarse model with activated stencil test to precisely clip its boundary. This method fails for any interior silhouette edge of the model which can exhibit obvious artifacts (see Figure 4.20).

To conclude on this part, we would like to mention another innovative approach. In [CK01] spline surfaces are triangulated on the on the fly. The object is placed in an octree, whose block sizes are used during runtime to decide on the detail level. More precisely, a subset of sampling points is chosen from an initially large set, which then defines the surface. Several optimizations are applied to improve speed and also to prevent 'cracks' that could appear between two spline patches that are sampled at different precision. The goal of the algorithm was not to have an exact error bound, which in fact is not provided, but to obtain a detailed representation in form of a triangulated surface. The on-the-fly-triangulation keeps the algorithm from being applicable in situations where high performance is needed. Another problem is that texture information is probably difficult to use in these conditions. It is still an interesting possibility because such a hybrid solution could prove valuable on modern hardware, where the spline surfaces could be traced in the fragment shader.

Image-Based Solutions

Further Reading: *Obviously, much more could be said on image-based methods and our presentation is by far not exhaustive. We refer the interested reader to the excellent state of the art report on image based representations [JWP05]*

We have just seen some approaches that leave the polygonal representations and result in something that looks correct solely from the current point of view. After all, the final output we want to produce is nothing but an image, so one representation that seems particularly tempting are images themselves. These can then be merged to yield the final result.

One of the earliest such solution was provided by Maciel and Shirley [MS95], who coined the term *impostor*. They proposed to replace large parts of the scene with textured quads instead of rasterizing the costly geometry. Their system evaluates a mix of cost/benefit heuristics and visibility estimation to choose hierarchically the textured or geometric representation. This situation does not allow to introduce an error bound on the representation.

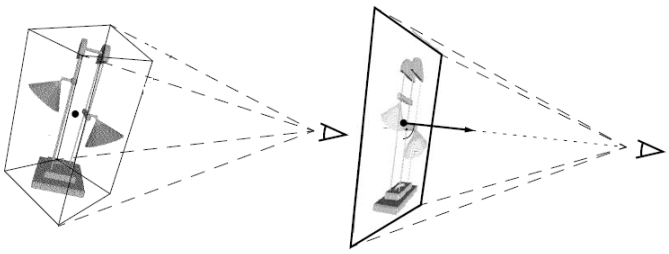


Fig. 4.21 : *Left: Geometry, Right: Impostor (Courtesy of Schaufler and Stürzlinger [SS96])*

Schaufler and Stürzlinger [SS96] presented an impostor method that updates textures on the fly thus giving higher precision. The whole scene is placed in a Kd-Tree and each leaf of the tree contains a certain number of primitives. In a first step, for each bounding box in the hierarchy an impostor texture is created from the current view-point. This texture will be used instead of the original geometry as long as the observer does not invalidate it by moving in the scene. To determine validity, two possible artifacts are considered: insufficient resolution and missing parallax effects. In these cases, the algorithm switches back to the original model or creates a new impostor. Both errors are bound based on the object's bounding box.

To exploit the scene hierarchy, higher nodes in the Kd-Tree might use impostors of lower nodes (if they are still valid) to create their own impostor. It is not specified in the paper how the error is treated exactly in this case. The pseudo-code suggests to use a lower impostor whenever it is currently still valid when the higher element is constructed, but it seems that this is insufficient. The proposed error bound is only valid if the impostor has been shot from the last point of view. This is not the case when rendering an impostor with impostors. Even though, the paper does not mention this case, the error bound can be invalidated for higher nodes. If hierarchy has depth d, the error can be as much as d times larger than predicted. This is not tragic, but worth mentioning if high fidelity is needed.

A very similar approach was presented concurrently by Shade et al. [SLS*96]. Minor differences with respect to [SLS*96] are that the Kd-Tree is replaced by a BSP-Tree and they suggest using only a 2D hierarchy in the case of height-field-like scenes, as is typically the case for landscapes. The main difference is that impostors are only created if a heuristic predicts that it could be a win for the system. They derive an estimation of how long an impostor is going to be valid and compare the time for creation with the predicted accumulated gain over several frames. To estimate how long an impostor will be valid, they derive a so-called safety region around a viewpoint that approximates how far an observer can move before a deviation to the original becomes visible. Texture resolution is ignored, but the parallax deviation is estimated.

The error principle shows similarities to our solution. Given a point P and its simplification on the impostor S, then S is invalid if the angle between P, the viewpoint, and S is larger than some angular error threshold. They point out that, in 2D, the viewpoints for which this fails all lie inside of the zone described by two special circles containing P and S. To apply this observation in practice, they define the safety radius as the distance to this forbidden zone derived from the corners of the impostor's bounding box. The safety radius can then be used to estimate the time an impostor will be valid, given a certain displacement speed. This is not a conservative estimate as the authors point out themselves, and as we will show in Chapter 5. Further, this minimum radius is applied in 3D by associating it to a safety sphere. This is not accurate and the situation is more complex. However, the metric we aim at is similar.

Impostors are always maintained for a certain period until the observer moved so far. So to some extent it is a natural idea to store image representations in a preprocess for certain parts of the scene. This leads us to the the viewcell evaluations.

5 Viewcell-Dependent Simplification

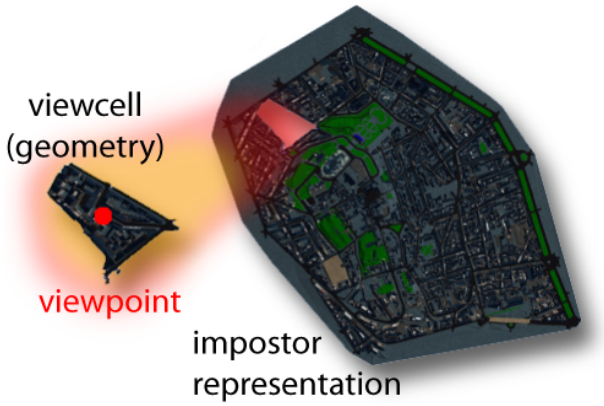


Fig. 4.22 : *Typical Ratio: Geometry / Impostor*

We recall that a *viewcell* is a region in space that describes possible viewpoints while exploring an object. For distance based level-of-details, these correspond to the volume outside a concentric sphere around the object. In the situation of a city walk-through, streets are often used to define those regions. In general, it is a challenging problem to derive appropriate viewcells in a scene. Somewhat related is the question of how to cut a scene into regions for which it makes sense to store a list of visible objects. In a building, these are usually separate rooms. In a city, these can be quarters or streets.

An early approach to decrease the complexity of a scene are the so-called mesh impostors that have been described by Sillion et al. in [SDB97]. The idea is to represent distant geometry by one single mesh grid. The algorithm was proposed in the context of city simplification. Therefore, the observer is supposed to walk on a street, which restricts the movement more or less to a forward/backward translation. For each street, a meshed impostor is derived from the z-buffer. An image is shot at the entry point of the view cell (here a street, thus a segment) storing color and depth information. Strong discontinuities in the Z-image are detected and used to define edges of discontinuity. Using these together with a constrained Delaunay triangulation leads to a final mesh grid. The texture is then applied to this surface and results in a very simple model.

An improvement was then suggested by Décoret et al. [DSSD99] who propose to subdivide the mesh impostor into several layers. Objects are grouped together in a layer based on a parallax measure in 2D. This measure corresponds to the difference between the biggest and the smallest angle under which two vertices, one from each object, can be seen from a segment view cell. A graph is created between all the objects that overlap for a viewpoint in the view cell, weighted by their angular distance. This allows to cluster elements greedily until the distance of each group remains underneath some threshold value. Each of the layers is then created independently. Another important contribution is that only coarse textures are saved and a finer version is calculated during run time while the observer is moving inside of the viewcell. Although the approach is very elegant and works well in practice, one problem remains: the fact that the reasoning is object-based does not assure any general bounds, but it improves appearance significantly. The error created by the transformation to an impostor is not taken into account.

Jeschke et al. propose a different way to represent the scene in [JWS02]. The idea of their technique is not to replace geometry by a coarser geometric representation, but by a set of texture- and alpha-mapped cubes, that surround the cubic viewcell. The original geometry is only used for the innermost cube. For a given reference viewpoint, the textures for each region between the cubes are rendered and stored. At run-time, instead of the original geometry, only the alpha-matted textures need to be displayed, leading to a substantial speed-up and excellent quality. Texture storage is definitely an issue because the number of layers increase substantially

with the view resolution. The distance placement of the cubes as well as the reference point placement is very well analyzed for this particular situation [JW02a].

The error metric takes parallax effects, as well as visibility of pixels between separate layers into account. The number of necessary cubes and their placement are derived to ensure that no gap bigger than a single pixel appears between two succeeding layers as seen from the viewcell. The maximum error is thus measured as the shift between two pixels of the same line of sight. The reference point position is optimized to assure an evenly distributed error for all viewpoints and the question is answered how to split geometry elements between different layers. The derivation is very interesting, but limited to the particular representation and it does not take the scene's geometry into account. This makes it valid for arbitrary scenes and results in an interesting consequence: The maximum error always occurs on the extremities of the viewcell. This is not generally true for billboard representations. This lead to a mistake in [DÓ2] (chapter 3.10 ff) invalidating all derivations of the following pages. If the innermost cube (containing real geometry) is shrunk, the number of layers approaches infinity. This implies that a minimum impostor distance is always necessary. Also, as the scene is not involved in the process, radii might not be optimally chosen with respect to the actual geometry. For example a single flat polygon, perfectly aligned with a cube face might still be projected on the nearest cube instead of adapting the impostor such that a cube face passes through the polygon.

A scene based optimization is the texture-compression scheme, that extracts the opaque elements of a cube face and packs them more densely into micro-textures. This is especially interesting for outer cubes, where the textures can contain large empty areas hidden by the geometry on the inner cubes. Unfortunately, this has implications on the rendering, but texture sprites [LHN05a] could be a solution on modern hardware.

Memory remains the major cost factor and if errors should remain underneath a small error threshold, the amount of layers further increases. In an amelioration of this technique [JW02b] the layers are replaced by a mesh augmented with a projected texture. Basically the same error measure as in [JW02a] can be adapted for this case. The solution efficiently combines fidelity and efficiency.

One major drawback of all mentioned methods is that the impostor's texture information comes from a single viewpoint which can lead to skinning artifacts or holes. This is why Wilson et al. [WM03] optimize visibility during the preprocess and present a fast mix between image- and geometry-based representations. This paper also gives an idea of the actual storage necessities to apply such methods. For the powerplant model (15 Million triangles) Wilson and Manocha report the need of 1.4 GB data to cover approximately 15 % of the terrain, and 9.4 GB for [JW02b]. Similar results hold for [AL99], who sample the scene from many viewpoints and derive Layered Depth Images [GwHC97].

Storage costs can become even more extreme if any geometric information is completely left aside. Light fields [LH96] and Lumigraph [GGSC96] are one of the early pure image-based representations. Given two patches, all rays between these patches describe a 4D space a *light field* (see Figure 4.23). If an object is added to the scene, it results in a 4D function that gives the transported color along each ray. For a given viewpoint, an illusion of the object can be created by querying the function for each needed view ray. The interesting features of these algorithms are that any kind of illumination or complex material can be "baked" into the light field and the display cost is independent of the content. Storage on the other hand is extremely high due to the increased dimensionality. Theoretically, this representation would be exact, if resolution

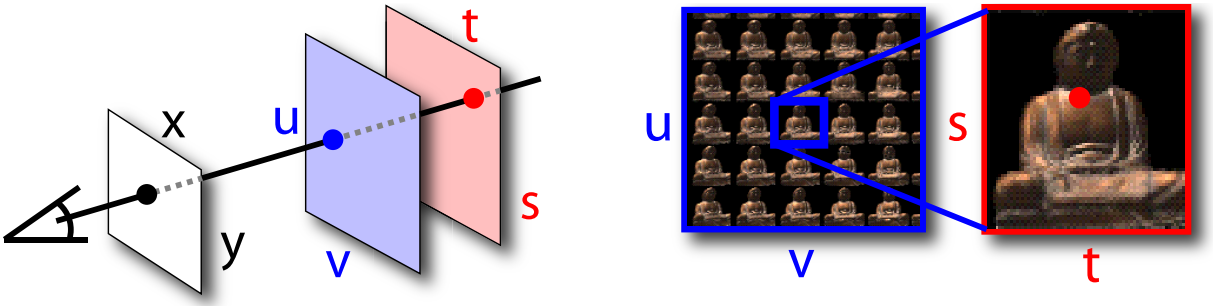


Fig. 4.23 : Light Field

A light field [LH96] is a 4D set of rays defined by two planes.

were sufficient. In practice, one often sees the sampling and ghosting artifacts. Recently a camera has been proposed that captures a limited light field directly [NLB*05] and thus a lot of post-processing can be applied to the photo, like refocusing or view changes.

Much more work has been published on this topic, but as it is very particular we will not further discuss it. As entry points for further reading, we suggest [CTCS00], which presents an error analysis, [IMG00] for manual error corrections and [LS04] for an error analysis including the scene's geometry.

6 Emerging Representations

Using standard images is somewhat only a first step, newer cards offer a much larger variety of possibilities. Much could be said on GPU adapted alternative representations, but as it is less related to error bounds we will only mention some of the directions that are currently of particular interest.

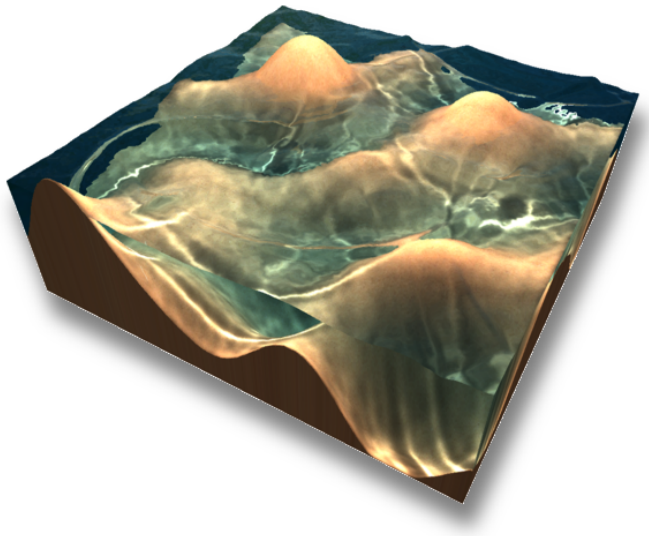


Fig. 4.24 : *Water Rendering [BD06a] with [BD06b]*

Height-field rendering on the GPU has received much attention since Policarpo et al. [POJ05] showed that very high frame-rates are possible with these techniques, but their binary search had many artifacts and could miss intersections. Baboud and Décoret [BD06b] provided a more accurate and efficient method by precomputing a *safety radius*, a distance that can be stepped over, without penetrating the surface twice, it is thus possible to start a binary search whenever a new position is found to be below the height-field. The same idea has been reused in Policarpo and Oliveira's cone stepping [PO07]. These approaches are fast and the cost is view-dependent (only

based on the projected size on the screen). Baboud and Décoret [BD06b] focused on rather simple cases, but even complex objects can be replaced.

In [MJW07] billboard clouds are combined with these representations and in [ABB*07] sampled views around the object are used. Billboard Clouds [DDSD03] are a solution that approximates a given model with a set of textured quads. We will give a more evolved presentation of Billboards and Billboard Clouds in Section 5.1 of Chapter 5 because we we built upon this algorithm to illustrate the usage of our error bound.

Another recently created trend are refinement methods for smooth surfaces. Patches of triangles are sent (or constructed) on the GPU and the vertex positions are then transformed to approximate the smooth surface. This has been successfully applied to terrain rendering [NPC07], but also to general models [BS05]. The integration of view-dependent simplification is possible by interacting on a CPU level. Depending on the needed detail level an accordingly tessellated patch is selected for the GPU. Newer ATI graphics cards also offer a subdivision scheme in hardware, which opens up the road for many new applications, but is currently not accessible from a standard API.

Mixing different primitives (polygons, lines, points, ...) has been used successfully for plant rendering [DHL*98]. Point-based rendering can also be an efficient solution for distant objects [DVS03], or in a special form like far voxels [GM05].

Many of these approaches share the common denominator of representing the objects at a certain visual scale. This relates to stochastic simplification [CHPR07]. It is a method that tries to capture the appearance of a model through statistics and adapts the shape of the simplification to behave like the original. This kind of looking at simplification seems to be a promising avenue of future work.

7 Beyond Error-Bounds: Transitions

Having described this large variety of possibilities, one question remains, which is how to transition smoothly between two LODs. The process has the goal of hiding artifacts that might appear when switching from one representation to the other. It is not really related to a geometric error and rather perception oriented. Also, in many cases, the workload decides on the LODs and not only the visual quality. In consequence, we keep this section very short.

It is quite impressive that, when blending two images that only differ in a small region, we are usually not able to perceive the change. A brute force switch between the two images, on the other hand, will draw our attention directly to the area of change. For LOD rendering transitions should therefore be as continuous as possible[9]

We have already mentioned the possibility of geomorphs [Hop96] for continuous adaptation. Whereas Hoppe [Hop97] applies sufficient subdivisions to avoid any incoherences, El-Sana et al. [ESV99] rely on a smart indexing solution to quickly find dependencies and finally, Xia and Varshney [XV96] ensured during the hierarchy creation that simplification/refinement can be applied coherently. But besides the difficulty of preventing fold-overs and creating a consistent cut

[9]Many surprising illusions and entertaining examples that illustrate the limits of our perception can be found in [Vis].

in the tree, many of these CPU updates are no longer a good choice. The streaming capabilities of graphics hardware are much faster and should be exploited.

Fig. 4.25 : Progressive Meshes

Blocks of geometry are transferred to the GPU at different LOD's (color coded). The geometry is geomorphed. (Courtesy of Sander and Mitchell [SM06])

Progressive buffers [SM06] by Sander and Mitchell are a GPU efficient implementation of geomorphs. Usually, vertex updates on the CPU brake the fast processing on the GPU side. To limit CPU interaction, their solution relies on blocks of geometry. These blocks are computed in a preprocess. For each level of detail, compatible adjacency is assured. Each vertex in a block stores not only information about their current state, but also the destination/color/texture coordinate in the next lower representation. This allows to carry out the transformation directly on the card by passing this information via texture coordinates and adaptation in the vertex shader. Morphing is performed using per vertex weights based on the distance to the observer. This ensures continuity between adjacent blocks even during geomorphs. On the CPU side, a block is exchanged when it reaches the distance where all geomorphs are finished.

Heuristics predict which blocks are most likely to be replaced soon to prepare data transfer. This data exchange is always costly and thus a compression scheme is applied (only 28 bytes are used per vertex, by compressing all attributes. Each object has vertex coordinates that lie in [0,1] and that are stored in a locally scaled frame which is passed as a uniform variable).

A very similar idea was used in QuickVDR [YSG05] for very large models. A different solution to prevent cracks between blocks is to forbid - during the hierarchy creation - the simplification of boundary elements. Only when blocks are merged for the next level these previous border elements will have their geometry reduced. This achieves an out of core rendering where blocks are tested for visibility, loaded, and adapted at run-time, leading to smooth transitions for large models.

Instances: *One problem for [SM06] and [YSG05] is that each block requires its own draw call because texture and VBO cannot be switched on the fly. This is particularly problematic in cases where several similar objects share the same vertex buffer and would thus benefit from being drawn consecutively. Using the now available instances, it is possible to regroup those objects on the CPU and perform a single draw call that will create all objects of the scene. In our tests, we found that this does have little influence on the rendering speed, but the CPU performs less work and is thus available for different tasks.*

The transition we explained so far, are inherently entangled with the simplification process and there are even methods relying on a particular input, like geometry images [LHSW03]. Giegl and Wimmer [GW07d] presented an entirely different and more general approach to blend between two given representations. Alpha blending has long been used to perform an object exchange in image space where the pixels of one object are blended with those of a second. The standard solution assigns an alpha value of t to the first and $1-t$ to the second (more advanced techniques work with alpha masks [KCS08]). To achieve blending in a complete scene is not as simple as it might sound. A depth needs to be produced in order to block the scene from overdrawing parts of the object. On the other hand, if a depth value is written, the two superposed LODs will compete with each other, leading to substantial artifacts. One solution would be to use offscreen buffers in which the blending is performed and simply add the solution back into the scene. This could be done by drawing the object again in the view, but filling the region with the texture from the offscreen buffer. This process is very costly as the operation needs to be applied to all objects in transition from one LOD to another. Giegl and Wimmer propose to always draw one object opaque and blend the second on top. Only the opaque one writes a depth, but both perform a depth test. To switch the two models, the transparent object fades in and once both objects are opaque, the roles change and the newly transparent one starts fading out. Of course, artifacts might occur. The method relies on the assumption that the depth values created by the two representations are similar. The moment the roles are switched, popping can occur if the geometry does not well align, or in areas where several surface elements of one object lie along the same line of sight without being hidden by the depth of the first object. The authors argue that this can be hand corrected, but automatically derived simplifications might not be well-adapted. Particularly, structures like billboard clouds [DDSD03] will be difficult to handle. Nevertheless, it is a fast solution that works well in practice.

Recently, it was pointed out in [SW08] that drawing both LODs in each frame is costly too. Thus an alternating rendering scheme is used and pixels are projected from the previous into the current frame. The blending is similar to [KCS08] and applied in object space.

The attitude to decrease transition quality in order to improve performance is also at the basis of many cost heuristics. Sometimes, a low LOD would deteriorate image quality, but an acceptable framerate can only be maintained if a switch is performed. Funkhauser [FS93] developed a metric to predict the cost of a representation and can therefore predict the impact of changing the representation. Nowadays, their heuristic is probably no longer valid, because vertex- and fragment processors are considered independent, parallel processes. With streaming architectures, like the GeForce 8, this is no longer the case.

More contemporary, Sander and Mitchell [SM06] adjust the transformation distances of the geomorphs to maintain a fast execution, based on limits of the current frame-rate, triangle count system - and texture memory.

Instead of assuring quality in the entire image, Watson et al. [WWHR97] explored a system that involves the image center and special semantics. We perceive the surrounding with less precision than the center and tend to observe some objects more than others. A similar principle was used by [TO96] who rely on a special equipment to track an observers view and adapt the representation accordingly.

Perception is generally an interesting topic that we mostly excluded from the descriptions. In these cases an adjustment is performed based on what is perceivable as a difference in a mesh. Whereas most work focused on single isolated objects (e.g. [WLC*03]), recent work [DBD*07]

in this direction also considers the environment. The idea is to render the scene and decide on the representation needed for the following frame. To do this, the authors extract in each frame several scene layers that are analyzed with a perceptual metric. An interesting side effect is that environmental influence is taken into account.

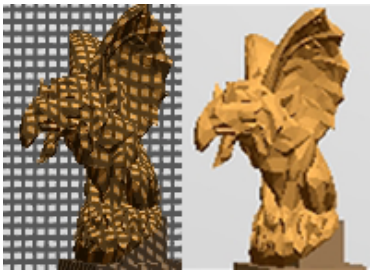


Fig. 4.26 : *Shadow masks inaccuracies [DBD*07]*

A model on which a high frequency shadow is cast can be simplified substantially without introducing a perceivable difference (see Figure 4.26). Transitions between LODs are done exactly when they are the least visible. The framework is very simple and the examples very particular. In general, the approach might often fail, but it is an important step in the direction of scene-dependent perceptual simplification.

Summary:

Many simplification algorithms rely on coarse error bounds. These can fail and are not tight. Most work on simplification is first performed independently of any view-dependence and result in a hierarchical representation. Only this hierarchy is traversed in accordance to the view at run-time. This involves a set of heuristics to chose an acceptable refinement degree. The process can be problematic if the traversal and model adaptation cost is high.

The most efficient transitions are GPU geomorphs, but none can compete with a fixed geometry that is switched appropriately. There are even architectures, like the Playstation 2 which allow only these switches in an efficient way. The according LODs are often created by standard simplification algorithms. These usually accumulate errors, which can quickly become overconservative, or rely on hulls that admit only error-respecting simplification steps. The latter can deliver correct results, but often fail to capture other metrics than the purely geometric Hausdorff distance that does not respect all points of the model, which is important in the presence of attributes, e.g. texture.

For viewcell-dependent simplification, little general work exists on error bounds. Further, there is no approach to evaluate the quality of a viewcell-dependent approximation. Such a solution does exist for standard simplification [CRS98], where only the model is sampled. No error bounds seem to exist that would enable a similar application for viewcell simplification. Some good error bounds exist for image-based approaches but are mostly targeting a very particular algorithm and often ignore the actual scene geometry.

This motivated our work on the geometric error in the context of viewcell-dependent simplification. Similar to other error-bounding solutions we will derive a hull, in which simplification can take place, but that respects the deviation of each point on the surface. We believe that it is of interest to gain an insight into view-dependent simplification and make another step in the direction of fidelity.

"I think that I shall never see
a billboard lovely as a tree.
Perhaps, unless the billboards fall,
I'll never see a tree at all."

Ogden Nash

Chapter 5

On Exact Error Bounds for View-Dependent Simplification

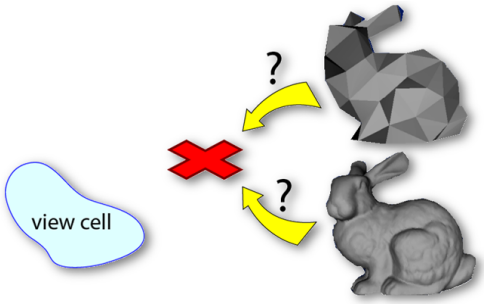


Fig. 5.1 : *Which representation is a good choice when observed from the viewcell?*

In this chapter, we investigate how geometry can be simplified, to allow a faster rendering than when compared to the display of the original geometry.

If the application runs fast enough a level of detail can be determined at run-time. In other scenarios this might be impossible, because the geometric complexity or number of objects is too high to work *on-the-fly* and a hierarchy of LODs has to be derived in a preprocess. This implies that the visual quality needs to be assured not only for the current viewpoint but a group of viewpoints. Previous work focused on the development of upper bounds on the deviation in a viewpoint-independent context. In this chapter, we will present our journey towards exact error bounds in the context of a viewcell. The question is how much an object can be simplified when the observing region is known (see Figure5.1).

This chapter contains proofs that result in an analytical solution which ensures exact error bounds for view-dependent simplification for the general 2D case and particular 3D cases. The discussion is rather general, whereas most previous work either addressed very particular cases or relied on coarse heuristics, that might fail and/or restrict movements or object representations.

We introduce the notion of validity regions as the complete set of possible simplifications with respect to a given error bound. We discuss arbitrary polygonal viewcells which allow a free movement in their interior. We show how to compute these regions for mesh points and faces. Since the validity region of a face accounts for all its points, properties like silhouette preservation and textures are gracefully handled. This is not the case if the error is controlled only at the face's vertices or edges.

Publication notice: *The content of this chapter represents a work with Xavier Décoret and lead to a publication in Computer Graphics Forum [ED07a].*

1 Introduction

Today, models are made out of thousands of triangles, and complex scenes like forests can even contain billions of polygons. Graphics hardware is constantly improving, but not capable of displaying such complex scenes in real-time. One option is simplification and a key point is to control the error caused by using a coarser replacement. This is a difficult problem for a single viewpoint, but even harder if one considers to keep the same representation for a certain viewing region (*viewcell*). In this chapter, we are interested in measuring the geometric error associated with a given simplification and viewcell. Furthermore, it is possible to detect for each mesh part the viewpoints that reveal the maximal error. Reviewing the simplification from this very viewpoint gives the user an idea of the subjective/qualitative error.

We reviewed a large body of literature in Chapter 4, but no work on a general error analysis seems to have been published so far. Interestingly, several previously proposed error bounds, even for the simpler case of a single viewpoint, are based on heuristics that are invalid in the general case. Viewcell approaches (where a simplification has been precalculated and is used while an observer stays inside a certain region, the viewcell) received interest because run-time simplification can be inefficient for very complex models. The query-cost simply exceeds the gain from rendering the simplified version. In this chapter we define and solve the problem of *exact* (not only upper) error bounds for points analytically in two-dimensional scenes and with arbitrary precision in 3D.

In practice, models are often well-behaved, making heuristics work well, which mostly produce large errors only in pathological cases. This is probably why the actual error has not yet been examined very closely. Nevertheless, our analysis is of interest in situations where faithfulness is a must and warns of failure cases. It serves as an invitation to further explore a field that existed for more than 30 years. Simplification still contains lots of basic open questions to which our work gives some answers and provides a deep understanding.

First, the definition of the problem will be presented (Section 2). The calculation of exact validity regions is not trivial and we will derive it in several steps. Then, starting with mesh points, we show how we can indirectly obtain the exact solution using particular viewpoints and describe how to find them for different types of viewcells (Section 3). This is interesting, as the shape of a viewcell often depends on the input. For a city model street-viewcells might be of interest, for a moving object, a shaft-like decomposition might be better adapted. We further extend the approach from mesh points to faces (Section 4). We then discuss our work, conclude and give an outlook on future research (sections 5,6).

2 Basic Definitions

The input of the problem is a mesh $\mathcal{M}$ and a viewcell $\mathcal{V}$. The *mesh* is defined by vertices V_i and faces $\mathcal{F}_j$, and consists of all the *points* inside those faces (usually denoted M). The *viewcell* $\mathcal{V}$ is a set of *viewpoints* (usually denoted V).

The goal is to measure the distance between $\mathcal{M}$ and a simplified mesh $\mathcal{S}$. A *simplification* is a mapping from the points of $\mathcal{M}$ to the points of $\mathcal{S}$, formally: $s : \mathcal{M} \mapsto \mathcal{S},\ M \rightarrow s(M)$

Note that s can be many-to-one, i.e. several points can be ”simplified” to the same place.

A classic approach to measure the error is to compare $\mathcal{M}$ and $\mathcal{S} = s(\mathcal{M})$ using the Hausdorff distance. This metric measures the geometric change between the mesh and its simplification. If every point on the mesh has a color (e.g., through texturing), two meshes can have the same shape (i.e. a null Hausdorff distance) but look very different. For that reason, distance measures between $\mathcal{M}$ and $\mathcal{S}$ should be considered for each point (not only vertex) on the mesh. Thus, in a first step, we will focus on single points on the mesh.

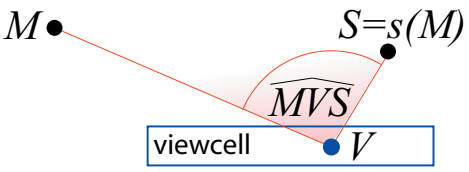


Fig. 5.2 : Angular Distance - *It is evaluated for each viewpoint V in viewcell $\mathcal{V}$*

For a single viewpoint V, the *distance* between M and $S := s(M)$ is defined as the angle $\widehat{MVS}$ under which the segment $[M,S]$ is seen from V (fig. 5.2). It is important to notice, that for a fixed view frustum, the angle implies a bound on the projected screen distance and vice versa.

For a viewcell $\mathcal{V}$, we define the error as the maximum angle under which a point M and its simplification S can be seen from within the viewcell.

$$e_{\mathcal{V}}(M,S) := \max_{V \in \mathcal{V}} \widehat{MVS} \tag{5.1}$$

We say a simplification is *valid* for a given error $\Theta \in [0, \pi/2)$, if for all mesh points $M \in \mathcal{M}$ the error between M and its simplification $s(M)$ is smaller than Θ.

3 Validity Regions of Points

The *validity region* for an error bound Θ of a mesh point M and a viewcell $\mathcal{V}$ is defined as the set:

$$VR^{\Theta}_{\mathcal{V}}(M) := \{S \mid e_{\mathcal{V}}(M,S) \leq \Theta\}$$

This definition is equivalent to:

$$VR^{\Theta}_{\mathcal{V}}(M) = \bigcap_{V \in \mathcal{V}} \{S \mid \widehat{MVS} \leq \Theta\} \tag{5.2}$$

From this we can obtain:

$$VR^{\Theta}_{\mathcal{V} \cup \mathcal{W}}(M) = VR^{\Theta}_{\mathcal{V}}(M) \cap VR^{\Theta}_{\mathcal{W}}(M) \tag{5.3}$$

Intuitively eq. (5.3) states that a bigger viewcell leads to smaller validity regions. In eq. (5.2), the set $\{S | \widehat{MVS} \leq \Theta\}$ is a cone of apex V in direction $\overrightarrow{VM}$ and aperture Θ. Therefore the validity region corresponds to the intersection of cones.

The remainder of this section describes the determination of the exact shape of the validity region. Fig. 5.3 shows some examples and one sees that the shape is not polyhedral, thus depending on an infinity of viewpoints.

From eq. (5.2) we see that validity regions are convex, because the set is defined as the intersection of convex cones. Convexity implies star-shape which means that the validity region of M can be represented radially by its *extents* with respect to M and a given direction $\vec{d}$. This gives a convenient parametrization of the set we look for.

$$VR^{\Theta}_{\mathcal{V}}(\vec{d})(M) := \delta VR^{\Theta}_{\mathcal{V}}(M) \cap r(M,\vec{d})$$

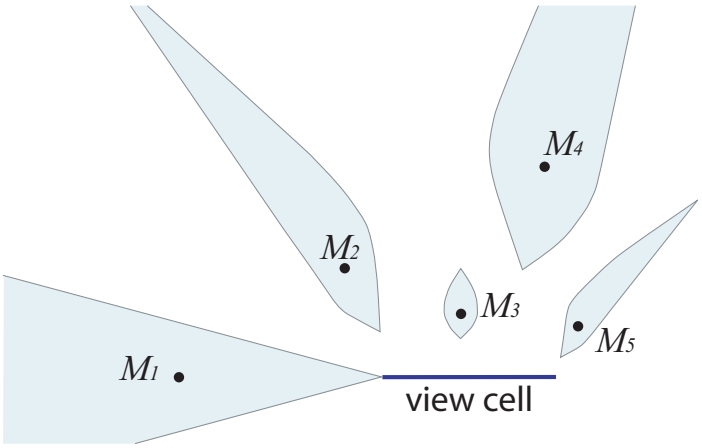


Fig. 5.3 : Point Validity Examples - *Validity regions of mesh points based on a given segment viewcell and a fixed error bound. Notice that the shapes are not symmetric and might be unbounded. Further, they are not always polyhedral which was an indirect implication of the claim in [DÓ2]. The image was obtained via sampled cone intersections.*

where $r(M,\vec{d})$ denotes the ray from M in direction $\vec{d}$ and $\delta VR^{\Theta}_{\mathcal{V}}(M)$ refers to the boundary of the validity region.

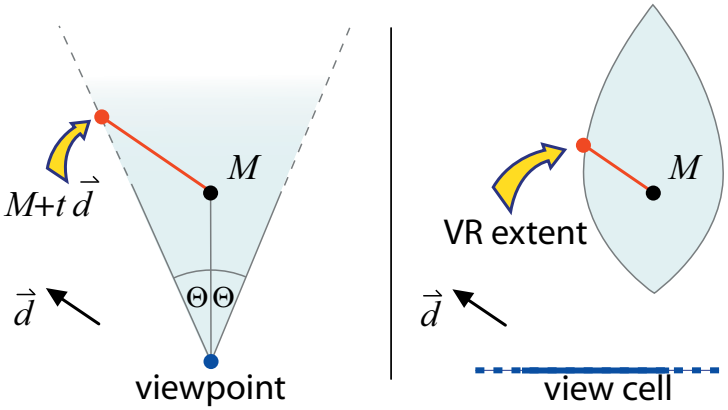


Fig. 5.4 : Validity Region Extent - *For a single viewpoint it is the intersection with a cone (left), for a hyperplane viewcell it becomes more complex (right)*

For a single viewpoint V it is simple to calculate the extents. It corresponds to the intersection between the view cone borders and a ray through M in a direction $\vec{d}$ (fig. 5.4).

For an error bound Θ the validity region extent in direction $\vec{d}$ of M for a viewpoint V is given by $M+t\vec{d}$, where t is specified by the (smaller) positive result of eq. (5.4). If there is none, the validity region is unbounded in this direction. If we denote $<|>$ the standard scalar product and $||\ ||$ the L_2 norm, we obtain the following formula for the validity extent for a given viewpoint V, mesh point M and direction $\vec{d}$:

$$\frac{- < \overrightarrow{VM} | \vec{d} > \pm \cot\Theta \sqrt{1 - < \overrightarrow{VM} | \vec{d} >^2}}{< \frac{\overrightarrow{VM}}{||\overrightarrow{VM}||} | \vec{d} >^2 - tan^2\Theta} \tag{5.4}$$

For general viewcells, it is difficult to calculate the validity region. On the other hand, for a finite set of viewpoints the solution can be found by evaluating expression (5.4) several times. The key

idea is thus to find a finite set of viewpoints in the viewcell that can be used to bound the whole validity region extent.

Summary:

So far, we have seen the basic definitions of validity regions. This is the region defined around a point in which it can be moved, such that the angular deviation from any point in the viewcell remains below some error angle Θ. For a single viewpoint this region is a cone, now we will examine general viewcells.

Max Viewpoints: It can be shown that for a closed viewcell there has to be a special viewpoint, the so-called *max viewpoint*, V_{max}, such that $VR^{\Theta}_{\mathcal{V}}(\vec{d})(M) = VR^{\Theta}_{\{V_{max}\}}(\vec{d})(M)$. V_{max} depends on the mesh point, the viewcell, the error bound and the direction.

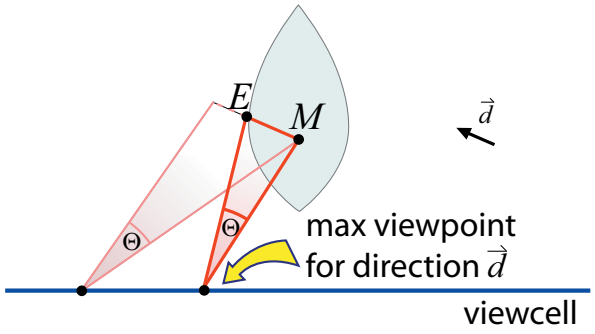


Fig. 5.5 : Max Viewpoints - *For a closed viewcell and a direction $\vec{d}$, there is one specific viewpoint, the max viewpoint, which implies the validity region extent E of mesh point M.*

In other words, the cone associated with the viewpoint V_{max} is actually responsible for the validity region extent in the direction $\vec{d}$. Denoting the validity region extent E, it follows $\widehat{EV_{max}M} = \Theta$ (fig. 5.5). The max viewpoint thus gives the validity region extent (expression (5.4)). This gives us an an indirect way to find the exact answer to our problem. The existence of these points will follow from the discussion in the special case of polygonal viewcells but it is generally true for closed viewcells.

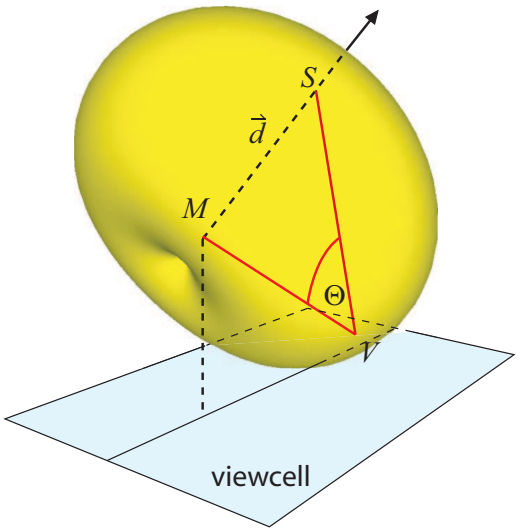


Fig. 5.6 : In 3D Iso-Viewpoints Form a Bialy - *Its points see $[M,S]$ under the same angle. The max viewpoint has to be on the line through the projection along the viewcell's normal of the line $(M,\vec{d})$*

The inscribed angle theorem states that in 2D all points that see a given segment under a fixed angle Θ lie on a bicircle, the outer contour of two intersecting circles. In 3D, the set is invariant under rotation around axis $\vec{d}$, thus leading to a so-called bialy, a torus without hole. This shape also played an important role in [LKR*96], where it was derived in a different manner for height field simplifications during run-time. All points inside this bialy see the segment under an angle greater than Θ.

This shows that the bialy defined by the mesh point, the validity region extent and the max viewpoint is tangent to the viewcell and leads to the observation:

$$VR^{\Theta}_{\mathcal{V}}(M) = VR^{\Theta}_{\delta\mathcal{V}}(M)$$

Hence, it is sufficient to calculate the validity region extent for the borders of a volumetric viewcell.

Due to eq. (5.3), each part of a polygonal viewcell can be treated separately by intersecting the corresponding results. In 3D we can restrict ourselves to faces, in 2D to simple line segments. It thus suffices to suppose one codimensional viewcells. We start by considering hyperplanes.

Hyperplane Viewcells: The bialy's tangency is key to finding the max viewpoint. Due to the bialy's rotationally invariant shape around direction $\vec{d}$ the tangency point has to lie on the orthogonal projection of the line $S(t)$ on the plane. This implies that for hyperplane viewcells the 3D case can be solved in 2D. Fig. 5.6 shows a bialy and the possible restriction to 2D.

The parametrization of a simplification point $S(t) = M + t\vec{d}$ leads to a parameterized bialy of points that "see" the segment $[MS(t)]$ under the angle Θ. The idea is to find the parameter t to obtain tangency (fig. 5.7). Instead of working on the bialy/bicircle itself, we treat the two circular parts separately.

Without loss of generality, the viewcell corresponds to the x-axis and the mesh point $M = (m_1, m_2)^\top$, with $m_2 > 0$. Then the circle is given by the equation for the center $C(t)$ and the radius $r(t)$.

$$\begin{aligned} C(t) &= M + \frac{t}{2}(\vec{d} \pm \cot\Theta \vec{d}^{\perp}) \\ r(t) &= \frac{t}{2\sin\Theta} \end{aligned}$$

where $\pm$ indicates the considered part of the bicircle.

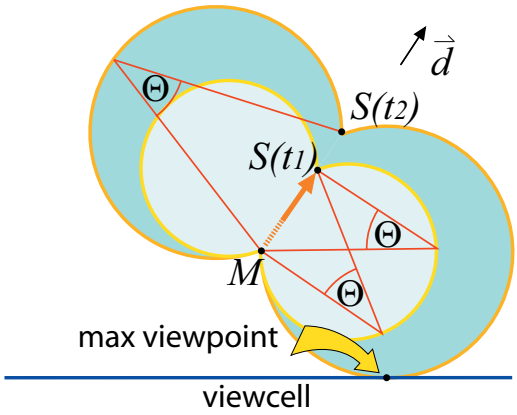


Fig. 5.7 : Bicircle Parametrization - *Parameterizing a simplification point leads to a parameterized bicircle (consisting of two circles). The max viewpoint is the tangent point on the viewcell.*

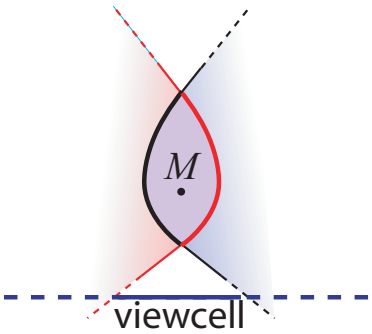


Fig. 5.8 : Hyperbola Validity Region - *For a hyperplane viewcell point validity regions are bounded by hyperbolas*

As the viewcell corresponds to the x-axis and M is situated in the upper half-space, the tangent/max viewpoint must have coordinates $(x_{max}, 0) = C(t_{max}) - (0, r(t_{max}))$. Solving the implied linear system leads to the coordinates of the max viewpoint:

$$V_{max} = (m_1 + m_2 * \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}, 0) \quad (5.5)$$

$$t_{max} = \frac{2\sin\Theta m_2}{\pm\cos\Theta d_1 - sin\Theta d_2 + 1} \quad (5.6)$$

again, $\pm$ depends on the considered part. More details can be found in the sidebox on page 88.

At this point, we get a first interesting result. If we parameterize t, which corresponds to the distance between the validity region extent and the mesh point M, via the direction $\vec{d} = (\cos\gamma, \sin\gamma)^\top$, given by an angle γ, we get:

$$t_{max}(\gamma) = \frac{\sin\Theta m_2}{\pm\cos(\Theta \pm \gamma) - 1}$$

which is an equation describing a hyperbola. The validity region of a mesh point for a line viewcell is thus formed by two intersecting hyperbolas (fig. 5.8). In 3D this region is rotated around the normal of the hyperplane viewcell. This is an exact description of the validity region for a hyperplane viewcell.

Summary:

A direct evaluation via minimization of the error leads to higher degree polynomials, thus we opted for an indirect solution to find the exact validity region. To simplify, we first considered only the extent of the validity region into a single direction $\vec{d}$. We found that in this case it is possible to restrict oneself to a smaller set of viewpoints: the max viewpoints. These are responsible for the boundary in direction $\vec{d}$. In a first step we examined hyperplane viewcells and we found that the validity region is accurately described with hyperbolas in 2D and 3D.

One Codimensional Viewcells: In the next step we want to restrict the hyperplane viewcells to polygonal areas. We will describe how to solve the problem in the 2D case and explain how to transfer the idea to the 3D case.

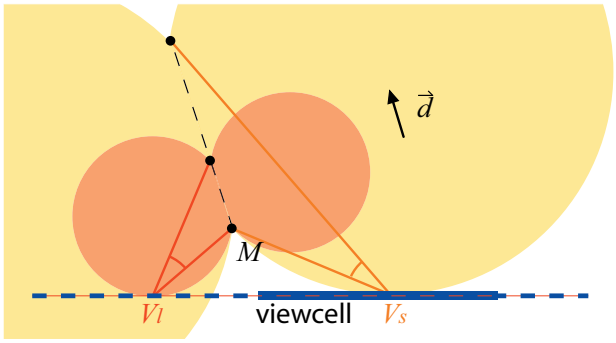


Fig. 5.9 : Max Viewpoint Location - *The line viewcell's max viewpoint in direction $\vec{d}$ is on the left, the segment viewcell has its on the right.*

When working with two separated circles instead of the bicircle, we actually find two "max viewpoints", one for each circular part. Although one of these two might not be a max viewpoint in the strict sense, to avoid confusion and with a slight abuse of notation, we will refer to both of them as max viewpoints. It is not problematic to test several points, as long as their number is finite. In particular it is important to consider both sides of the bicircle, because the position of a max viewpoint for a segment viewcell is in no direct relation to the one for the corresponding line viewcell (fig. 5.9).

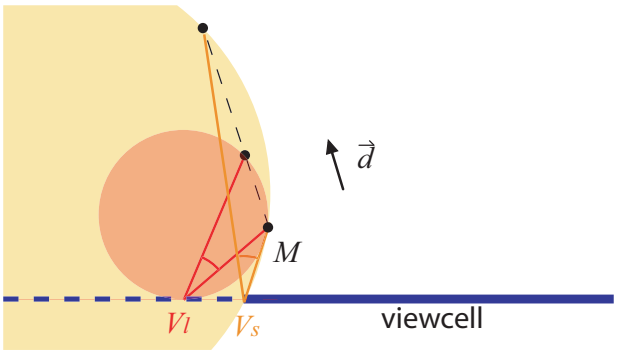


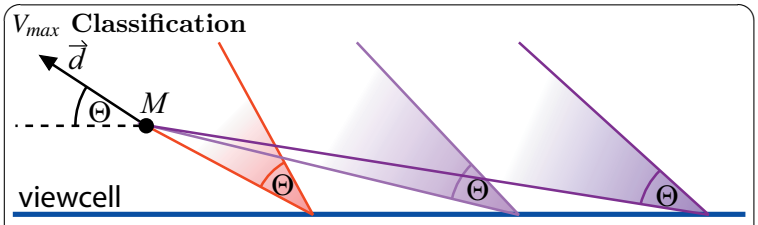
Fig. 5.10 : *Max Viewpoint on Extremity*

Let $\mathcal{V}_S$ be a segment viewcell and $\mathcal{V}_L$ its extension to a line viewcell. Also, let V_s (resp. V_l) be the max viewpoint for $\mathcal{V}_S$ (resp. $\mathcal{V}_L$). If $V_l \in \mathcal{V}_S$ then $V_s = V_l$. We now want to show that if $V_l \notin \mathcal{V}_S$ then V_s is an extremity of $\mathcal{V}_S$ (fig. 5.10). V_l has to be in the interior of the tangent bicircle at $\mathcal{V}_S$ (inscribed circle theorem). Therefore the segment $[V_s, V_l]$ lies also in the interior of the bicircle of V_s. $\mathcal{V}_S \cap (V_s, V_l) \neq \emptyset$ contradicts the tangency property, thus V_S is an extremity.

In the 3D case, we can establish the same distinction. The proof is essentially the same, the bialy for the hyperplane is contained in the bialy

for the viewcell, therefore the space between the two has to be empty. Unfortunately in 3D, this border is not a point, but a segment. To solve for the validity extent it can still be treated as a line and then the segment's extremities are taken into account. Still, finding max viewpoints for the line involves higher order polynomials and we did not yet succeed to obtain an analytic expression. Nevertheless, it is possible to approach the result arbitrarily close using numerical methods.

It is *not* a trivial result that numerically stable solutions can be provided. The proof comes from the geometrical interpretation that we established. The idea is to embed the line into a plane and look at iso-values for the sizes of the bialys (which are in direct relation to the lengths of the validity region extents). The result are half-moon shaped areas around the actual max viewpoint of the plane (fig 5.11). Thus, on a line embedded in this plane at most three extrema can occur. All viewpoints for which the validity region is unbounded lie inside a cone. Therefore one can quickly determine which part of the viewcell is not needed for the determination of the validity extent. To restrict the search space one can pick an arbitrary viewpoint V of the remaining viewcell and define a minimum distance such that all viewpoints exceeding this distance are less restrictive than V. These properties make it possible to compute the validity extent in a stable way with arbitrary precision.



Working with two circles instead of the bicircle might lead to an intersection with the part that lies in the interior of the bicircle. It can be shown that no meaningful max viewpoint occurs if the angle between $\vec{d}$ and the line viewcell lies in $[\pi-\Theta,\pi+\Theta]$. The case $\pi-\Theta$ leads to no intersection at all. For this "side" of the bicircle, the validity region extent is unbounded. The figure illustrates such a case. Even without this test, false detections will be unproblematic (in the worst case, one unnecessary evaluation is performed).

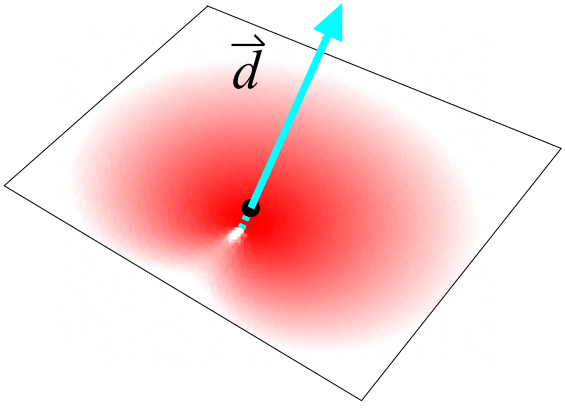


Fig. 5.11 : Halfmoon Iso-Values - *The bialy radii are shown on a plane (red low, white high).*

The result is general enough to be used with any simplification algorithm that provides a mapping between the original and the simplification (for example simplification of point clouds or sampled geometry (or texels of a textured mesh)). If the creation of the impostor was not performed using this error bound, the verification of validity still remains possible opening up the road for applications like [CRS98]. Our computation also applies to level of detail changes. Here, our investigation allows to derive the distance at which a representation change is needed. According to the desired error, all that is needed is to choose the distance such that the viewer remains outside of every error bialy. Nevertheless, an in depth investigation in this direction lies out of the scope of this work, but represent interesting avenues for the future.

Summary:

So far we have shown that it is possible to calculate the validity region of points given a finite set of (not necessarily bounded) polygonal viewcells. For a given point M and a direction $\vec{d}$, possibly given via an existing simplification point S, the max viewpoints are classified for each face of the viewcell. This is done by considering the hyperplane using eq. (5.5) and discarding viewpoints outside the viewcell face. Next the borders of the viewcell are examined. This gives a finite set for which eq. (5.2) is evaluated. This leads to the minimal extent (the boundary of the validity region). The next part of the paper discusses how to extent this notion of validity from mesh points to mesh faces.

4 Validity Regions of Faces

In this section we will extend the notion of validity regions of points to mesh faces, therefore taking color information (like texture) into account. Mathematically a face contains an infinity of points. Thus to avoid sampling, we want to establish a way to assure the validity of the modification of a whole face. This part is quite technical and currently our result is two-dimensional.

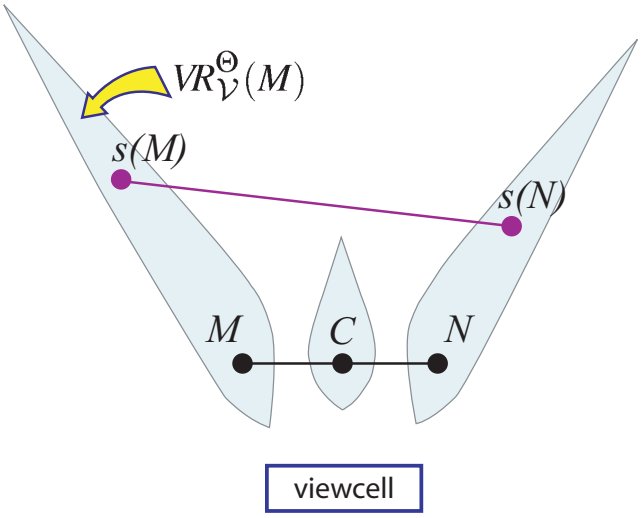


Fig. 5.12 : Face Validity - *When M and N are moved within their validity region (to s(M) and s(N) respectively) not all points on [MN] can be simplified on [s(M),s(N)]. E.g., C's validity region does not intersect [s(M)s(N)]*

The solution is not straightforward because validity regions for points inside the face cannot be simply interpolated from the validity regions of the vertices. In other words, if the vertices of a face are moved within their validity region, mesh points on the face do not necessarily remain in their respective validity region, as illustrated in fig. 5.12. This directly implies that algorithms basing their error estimate only on edges or vertex distances cannot succeed in establishing an upper bound on the error of the actual image (neither in 2D nor in 3D). A comparison of any algorithm like this with a method involving our analysis could thus be arbitrarily biased in our favor. In other words heuristic can and will lead to arbitrarily large errors (at least in special cases).

To define a validity region for faces, we must first fix the way a face is "moved" during the simplification process, thus specifying the "movement" of the corresponding mesh points. We chose a directional projection onto a simplification plane. This is actually less restricting than it might sound and has several benefits. The order of the mesh points and a certain injectivity and connectivity is kept during simplification. Compared to a perspective projection, no specific viewpoint has to be selected and faces could still be cut into parts to allow different directions

per patch. Finally it should be mentioned that the analysis is independent of the way the final representation is stored. If the obtained simplification is texture based, it can still be transformed under perspective projection to gain memory. Only the resolution changes, not the position of the samples.

In this part, we assume projections along the plane's normal to ease explanations, although the approach described generalizes to arbitrary directions.

A simplification plane is said to be *valid for a face* if each point on the face, when projected onto the simplification plane, remains inside its corresponding validity region (fig. 5.13). The approach of this section can also be used to find optimal texture resolutions for all viewpoints inside the viewcell. All that needs to be done is to measure the texel size by associating a simplification direction that lies in the simplification plane. The idea is to choose as the simplification direction the surface orientation, thus representing texels on the surface.

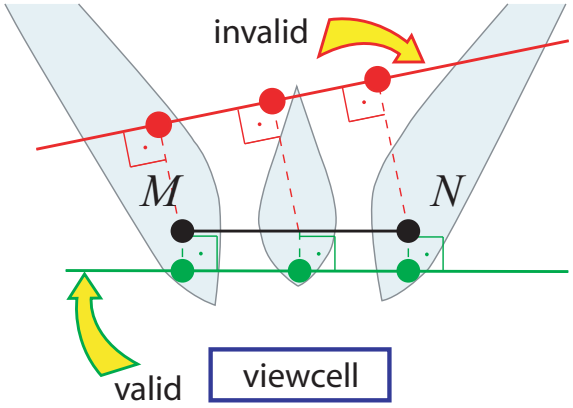


Fig. 5.13 : Valid Simplification Plane - *A simplification plane is valid for the face* $[M,N]$ *if* ***all*** *face points remain inside their corresponding validity region after being transferred.*

A plane can be represented by its Hough transform: a normal and an offset from the origin [Hou62]. Fixing a normal (and thus projection direction) leaves us with the question of finding the possible offsets for a valid simplification plane. For a single mesh point the offset is given via the validity region extents. For a mesh face, this is more difficult.

From eq. (5.6), we observe that for a line viewcell the extents of a validity region vary linearly with the distance of the mesh point to the viewcell. In this case it is sufficient to ensure validity at the face's extremities to obtain validity for all points on the face. This also holds in the 3D case for hyperplane viewcells. If all the max viewpoints fall inside the segment viewcell, we encounter a linear behavior and it is actually sufficient to test the vertices of the face.

This observation leads to the idea of decomposing the mesh face into several parts, for which we will determine the offsets separately. Having detected the linear part, we will see that the remaining parts are non-linear, but involve only one single viewpoint. Once offsets have been found for each part, an intersection leads to the correct solution for the face. An empty set means that there is no valid simplification plane for this projection direction, hence at least one point cannot be simplified. Realize that for each face there are always valid simplification planes, in particular the one containing the face itself.

Detecting the Linear Part: The determination of the linear part is actually equivalent to the detection of those points on the face for which the max viewpoint of the corresponding line viewcell falls into the segment viewcell. Due to eq. (5.5), we can determine for each point on the face the corresponding coordinates of the max viewpoint for a line viewcell. Actually there is a small subtlety; eq. (5.5) corresponds to a mesh point in the upper half-space, therefore faces should be clipped if they are intersected by the extension of a segment viewcell to a line. Of course, it would be possible to deduce a similar equation for the lower half-space, but splitting and

culling for polygonal viewcells implies that less geometry needs to be considered. The correctness is proven similar to the proof that max viewpoints lie on the boundary.

There is a linear correspondence between the position of the mesh point and the max viewpoint for hyperplane viewcells. Now, if the face is extended to a line, we have two points on this line for which the max viewpoint corresponds to an extremity of the segment viewcell. These borders of the linear part can be inferred by solving for $(m_1,m_2)^\top$ in eq. (5.5) where the max viewpoint corresponds to the segment viewcell's extremities (fig. 5.14).

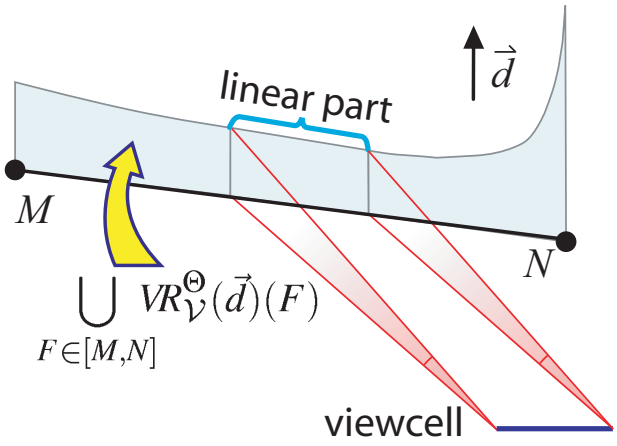


Fig. 5.14 : Linear and Non-linear Decomposition - *Validity extents in direction $\vec{d}$ of the face's points form the shape above, which contains a linear part. Accordingly the face will be decomposed into a linear region, where it is sufficient to test the extremities and the non-linear regions, which have to be dealt with separately.*

Supposing the face is given by the segment $[M,N]$ we will refer to the line through the face, as the *face line*, here given by $l(\alpha) := M + \alpha(\vec{MN}) = M + \alpha(w_1,w_2)^\top$. Without loss of generality, one viewcell extremity is $(e,0)^\top$. Thus, the equation to solve is: $V_{max}(l(\alpha)) = (e,0)^\top$. The solution is given by:

$$\alpha_e = \frac{e-(m_1+cm_2)}{w_1+cw_2},\ c := \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}$$

$\pm$ corresponds to the two sides of the bicircle.
If $(w_1+cw_2)=0$ all mesh points share the same max viewpoint. If this viewpoint lies inside (respectively outside) the segment, the whole face is linear (respectively non-linear).

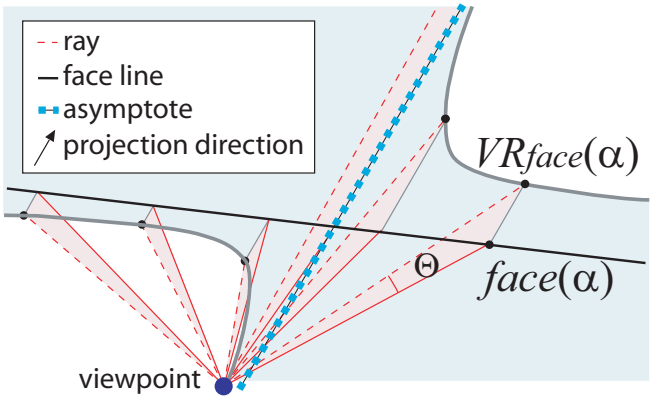


Fig. 5.15 : Face Validity For a Single Viewpoint - *The validity region extents of the mesh points on the face are given by the intersection between a ray from viewpoint with angle Θ and a line from the mesh point in projection direction.*

Dealing with the non-linear part: The way we detected the linear part actually implies one property of the non-linear part; it only depends on a single viewpoint: one extremity of the viewcell. Thus we only need to examine how the validity region of a face behaves for a single view*point*.

We will consider the validity region extent of points on a face line. For each such point P, the validity region extent is given by the intersection of a line passing through P in the projection direction and the viewpoint's view cone. If each side of the cone is treated separately as a line and we plot those intersections for every point on the face line, we get a graph as shown in fig. 5.15. It may contain "false" intersections which represent intersections with the line instead of the ray. This is unproblematic, as a false intersection implies a more restrictive intersection for the other view cone ray. The right part of the figure in the sidebox on page 93 shows an example.

The graph visualizes the validity region extents of all points on the face line. A valid simplification plane has to remain inside the hull described by the portion of the graph corresponding to the mesh face (blue region in fig. 5.16). Thus to find the offsets of our plane, we need to find tangents with normal $\vec{d}$ at the contained curve parts.

The following reasonings will have to be performed for Θ and $-\Theta$ (both branches of the view cone). The resulting graphs delimit the simplification plane's offset. Both cases are similar and we will only focus on Θ.

Let's develop a mathematical description. Given a point M, the viewpoint V and the projection direction $\vec{d}$, we are looking for the intersection between the two lines $l_1(\alpha) := M + \alpha\vec{d}$ and $l_2(\alpha) := V + \alpha R_\Theta(V - M)$, where R_Θ describes a rotation of angle Θ.

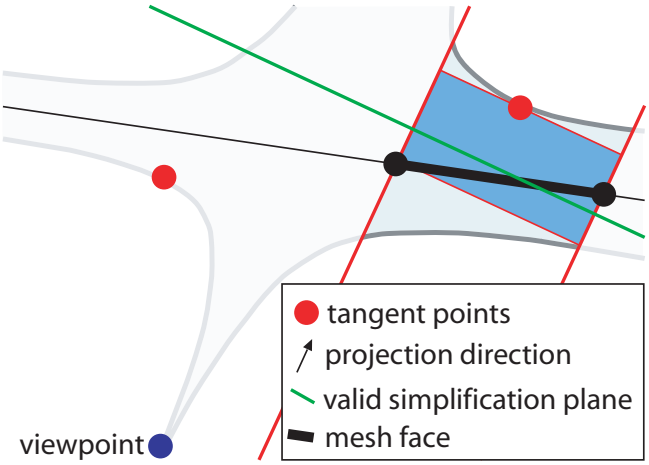


Fig. 5.16 : Example: Single Viewpoint Face Validity - *For Θ and $-\Theta$ we obtain functions representing the validity extents of all points on the face line. We are actually only interested in those points on the face line corresponding to mesh points. Therefore both functions are cropped. There are two cases: an extremum corresponds to a mesh point (upper curve) or only the extremities have to be tested because of monotonicity (lower curve).*

To ease the calculus, but without loss of generality, we can assume that $V = (0,0)^\top$, $\vec{d} = (0,1)^\top$. M is on the face line given by $face(\alpha) = (0,m)^\top + \alpha(\cos\gamma, \sin\gamma)^\top$. This last assumption is actually a restriction, as it is now impossible that the face describes a vertical segment, not lying on the y-axis. On the other hand, this is not crucial as the projection direction was assumed to be vertical. We simply exclude this case where a face would be simplified to a point. This makes sense and the remaining case could be treated separately.

The final curve corresponds to:

$$VR_{face}(\alpha) := \alpha\frac{\alpha\sin(\Theta+\gamma) + m\cos\Theta\cos\gamma}{\alpha\cos(\Theta+\gamma) - m\sin\Theta\cos\gamma}$$

We are now interested in the tangent points with a normal equal to the projection direction. As the projection direction has been chosen to be $(0,1)^\top$, we are actually interested in local minima and maxima of this quadratic rational. The curvature can only have two different signs, one for each side of the definition gap. Thus any point for which the derivative vanishes is automatically an extremum, there cannot be any saddle points and the function remains monotonic for the branches separated by the extremum. Due to this monotonicity, it is sufficient to test the extremities of the face if the extrema do not correspond to mesh points.

Finding extrema is equivalent to finding roots of the derivative. The resulting expression is at most quadratic and therefore possible to solve. The discussion is quite technical, because the function varies from a line, to a parabola, hyperbola and a "real" rational function depending on the angle between the face and the projection direction.

Summary:

For face validity it is not possible to simply assure validity of its extremities, which is a common error. We show that in 2D these faces allow to find a decomposition, such that for one part of the face we encounter a linear error behavior and the others solely depend on a single viewpoint. This allows to treat them separately and facilitates the task.

Concluding Face Validity: Let's wrap up on how to detect the validity of a face. (To accelerate the approach a more profound theoretical discussion of the functions would be necessary. These contributions are of rather technical nature and were thus excluded from this document).

For both parts of the bicircle, that is to say for both circles we find the linear region on the face. Then we calculate the maximum offset in the projection direction for the linear part by evaluating eq. (5.4) for its extremities. For the remaining non-linear parts there is only one viewcell extremity to be considered. The tangents at the curve are found for Θ and $-\Theta$. If the extrema correspond to mesh points these are evaluated, otherwise only the face's extremities need to be checked. The process is repeated for $-\vec{d}$, to get the lower offset bound for the simplification plane.

Tangents for Face Validity *If one supposes that the function can be simplified to* $c\alpha$ *(with* $c \neq 0$*) by division, we obtain*

$$\alpha(\sin(\Theta+\gamma) - c\cos(\Theta+\gamma)) + m\cos\gamma(\cos\Theta + c\sin\Theta) = 0$$

as $\cos\gamma \neq 0$*. Assuming* $m \neq 0$*, the constant part implies* $c = -\cot\Theta$*. Leading to*

$$\sin(\Theta+\gamma) + \tan\Theta\cos(\Theta+\gamma) = 0,$$

and thus $\cos\gamma = 0$*. This case had been excluded, as the face would be vertical. If* $m = 0$*,* VR_{face} *simplifies to* $tan(\Theta+\gamma)\alpha$*. A linear function, except if* $cos(\Theta+\gamma) = 0$*, then the ray becomes parallel to the projection direction* $(0,1)^\top$ *(validity extents are unbounded).*

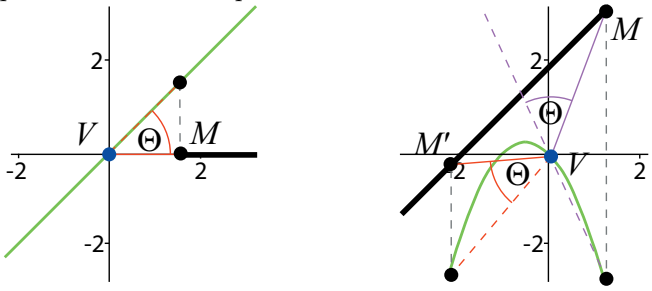
The case $\sin(\Theta+\gamma) = 0$ *leads to a real hyperbola and both branches are monotonic. There cannot be a tangent with normal* $(0,1)\top$ *at a hyperbola. One exception occurs if the numerator is completely zero (therefore* $m = 0$*), here we have a linear function, the x-axis. The case* $\cos(\Theta+\gamma) = 0$ *leads to a parabola except for* $m = 0$*, when the function is always undefined (see first case). In the parabola case the extremum is at*

$$\alpha_{parabola} = -\frac{m\sin\Theta\cos\gamma}{2\sin(\Theta+\gamma)} cot\Theta$$

For a quadratic numerator, a linear denominator that cannot be simplified by division the extrema are:

$$\alpha_{1,2} = \frac{m\cos\gamma}{\cos(\Theta+\gamma)}(\sin\Theta \pm \sqrt{\frac{\cos\gamma\sin\Theta}{\sin(\Theta+\gamma)}})$$

The figure illustrates the case of a line and parabola (right). The right part is also an example for a false intersection.



5 Discussion

In this chapter we examined the exact error in the case of viewcell-dependent simplification and describe how they can be visualized and represented geometrically. We treat several 3D cases and provide closed-form solutions for the 2D situation.

The innovative contribution is the notion of validity regions, which correspond *exactly* to the region of points which are close enough to be used as a simplification. We explain how their exact calculation can be achieved. Thus, we were able to establish an *exact* error bound. In other words, we give an answer to the question whether a simplification is valid, how much we can simplify and for which viewpoint the error will be most evident.

The evaluation of the validity regions is fast due to its closed-form representation. It takes ≈ 1 sec. for 290.000 point validity extents (≈ 0.0036 ms per evaluation) on a Pentium 1.5 GHz (face validity is usually about 3-4 times slower). Nevertheless we completely acknowledge that good heuristics can be simpler and often lead to acceptable results (especially for finely tesselated models). This is not surprising, as for distant geometry a small viewcell appears almost like a single viewpoint just like and small triangles behave like mesh points. Nevertheless, it is *not* equivalent. This work allows to evaluate the error resulting from simple heuristics and to compare to an exact reference. LODs can change drastically which is especially true if little attention has been paid to the induced error by the replacement. With accurate bounds the proximity is guaranteed throughout the viewcell and everywhere on the geometry. Overlapping viewcells or blending between LODs thus becomes meaningful.

Theoretically this metric could be useful in several contexts, like simplification envelopes [CVM*96]. Geometry modifications are accepted if the shape is close enough to the initial one. This approach would be applicable for any edge-collapse or vertex deletion algorithm if a simplification function can be defined. This is also possible for point clouds or by sampling in 3D (for example at texture resolution). This is somewhat related to [COM98], but involving viewcells. Our distance metric could be used directly during the simplification process [Fre00]. Other approaches might benefit from our work too. One example, is [AHL*06] a paper on soft shadows. Here the occluder was represented using a depth map and the error was estimated based on the gap between depth samples. Our analysis allows a classification of this error and can predict the region on the ground for which this error is biggest.

5.1 View-Dependent Billboard Clouds

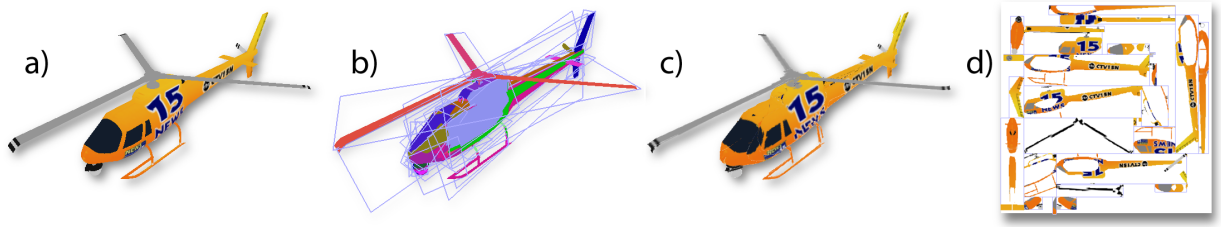


Fig. 5.17 : Billboard Clouds [DDSD03]

An input model (a) is transformed into a billboard cloud (BBC) (c). A BBC consists of several alpha-mapped, textured intersecting quads. The textures are compactly stored (d).

We also implemented a view-dependent billboard clouds (VD-BBC) approach whose run-time scales linearly with the geometric complexity. BBC [DDSD03] are a simplification via textured and alpha mapped planes. A triangle can be simplified (meaning projected) on a plane if all its points are in a proximity of some user-defined ε to the plane. It is similar to our plane validity,

except that the validity region for a point is a simple ε-sized sphere. For VD-BBC, we will use the validity regions to determine on which planes a face can be projected in two dimensions. The algorithm then chooses planes iteratively until all triangles have a simplification plane. To find the according planes, a so-called density value is used. This density value corresponds basically to the accumulated projected area of the faces for which this plane is valid[1]. Finding the minimal set of planes is an NP-hard problem, so the authors resorts to a greedy solution.

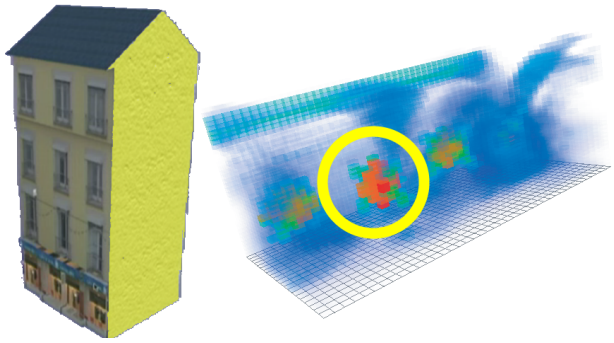


Fig. 5.18 : *Discretized Density and Best Plane with According Triangles (yellow) [DDSD03]*

The algorithm makes use of the Hough space, which is a dual space in which points correspond to planes in primary space (we will use the Hough space also in Chapter 12). Planes in Hough space are encoded with two angles (resulting in a normal) and a distance to the origin, there are poles but this only means that some areas will be oversampled. To select planes in this space, a density value is computed for a discretized representation of this dual. The discretization results in bins (each representing a set of planes). A triangle T contributes to the density value of a bin if there is a plane in its set which is valid for T. After looping once over all triangles we have an initial density value in each bin (Figure 5.18). A plane is chosen from the bin with the highest density. This plane is found by subdividing the bin and repeating the density computation a fixed number of times. Finally, an optimally approximating plane with respect to the remaining triangles is derived by distance minimization. All triangles that can be simplified on this plane remove their density contribution and the algorithm continues finding the next best plane until all triangles are simplified.

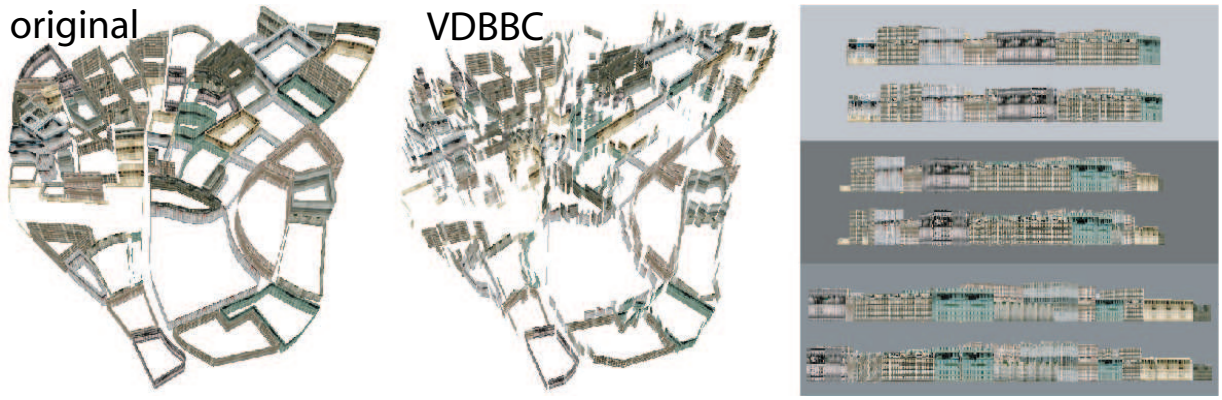


Fig. 5.19 : A City Model Is Simplified On Only 115 Quads.

The viewcell is a halfspace at the bottom of the image. Even though from above both representations seem to differ a lot (left, middle), seen from the inside of the viewcell both look quite similar (right). In each of the three sample views the original is on the top and the simplification underneath. (Model: Courtesy of J. Dorsey)

[1]The paper also describes a penality value, but this one is of no use for our purposes is only useful for rather convex scenes.

A first example is shown in Figure 5.19. The method was applied to a 4194 triangle city model. Due to its simplicity the model does not seem to leave much possibilities to simplify. Nevertheless, the resulting representation contains only 115 textured quads while maintaining a proximity of 0.1° (≈ 5 pixels) neglecting the vertical error. The viewcell in this example is a halfspace situated at the lower bottom of the image. In particular it is interesting to see how the structure of the city is maintained for nearby parts, whereas far away areas are aggressively simplified. Especially the structure becomes unrecognizable from above, but appears close to the original as seen from the viewcell. As in the original BBC approach small cracks might be visible where neighboring faces project to different planes, but these openings are bound.

The second example is shown in Fig. 5.20. The scene is a billboard forest around a centering segment viewcell. The left part depicts the 2D representation to which we applied the VD-BBC approach. The upper left quadrant shows a quarter of the scene and its geometric complexity. The lower left quadrant shows that our method successfully simplifies distant geometry. The red segments show the orientation of the created billboards and all shown green segments have been simplified on these. The figure shows only those billboards representing more than 100 trees. Finally the right half shows an example of this simplification. All green trees simplify to the same billboard whose orientation is shown in red. In particular you might notice that the simplified geometry forms a star. This results from the segment viewcell, which is aligned horizontally. The parallax effects are thus less pronounced along this direction. All in all we realize that even exact bounds still allow for aggressive simplification in higher polygon count scenes. Only $< 2.3\%$ of the original complexity remained. On the one hand this does not have a real signification because we could simply add more geometry to further improve these statistics.

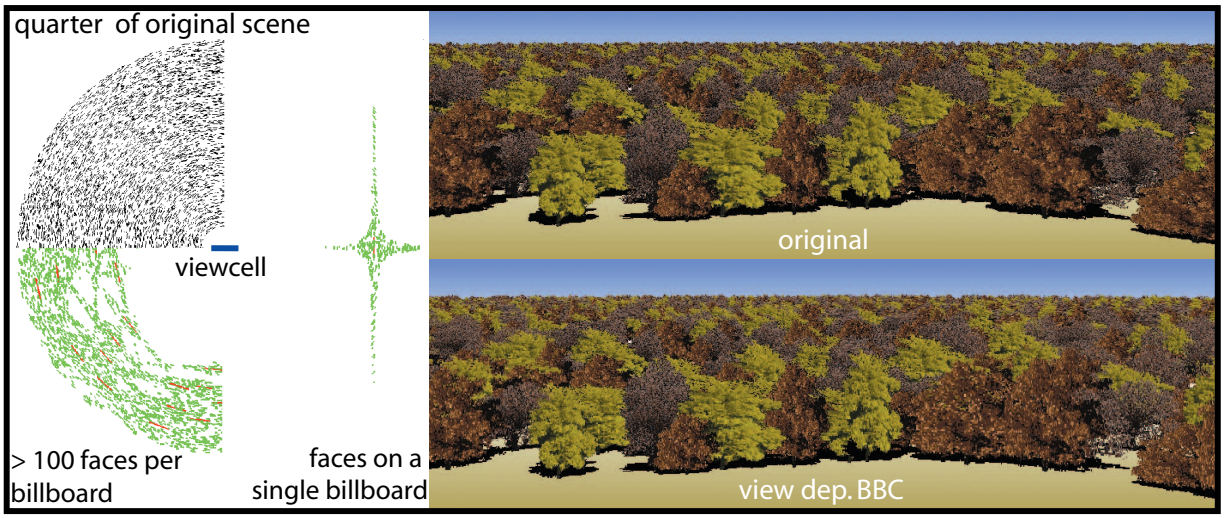


Fig. 5.20 : Simplification of a Complex Forest Model

The left two quadrants of the left part of the figure show the original scene complexity (top) and a selection of billboards containing more than 100 *trees (the red segments indicate the billboards, the green elements the simplified trees). In this scene the viewcell is a centered segment. Its shape influences the validity regions and leads to a star like set of trees that are simplified on the same plane. Seen from the viewcell the original and the billboard cloud representation look very similar, even though less than* 2.3% *(compared to the original) of the geometry is involved.*

In addition, we point out the importance of considering all points in a face to respect texture. The error measure naturally leads to silhouette and parallax effect preservation. This approach combines local point with mesh information and avoids sampling. Movement of the observer is not restricted and arbitrary polygonal viewcells are possible. As mentioned before heuristics that only test observed vertex distances or edge lengths can result in arbitrarily large errors.

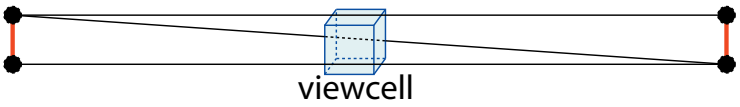


Fig. 5.21 : Pathological Example - *The vertical edges are small as seen from the viewcell, but their collapse makes the triangles disappear.*

Figure 5.21 depicts such a situation where the error converges to infinity. It is impossible to perform an unbiased comparison and we do not claim that well chosen heuristics are of no use. Very often heuristics perform very well and are easier to apply, but it is important to be aware of the shortcomings.

6 Future Work

Lots of interesting avenues arise from this work. We solved validity in 3D for the special case of hyperplanes and point validity in a numerically stable way. An analytical closed form expression, as the one we presented for the 2D case, would be an interesting result. Treating arbitrary (non-polytope) viewcells also remains a challenge.

At the moment visibility is not involved in the calculation of the validity region. For two-dimensional scenes, several algorithms exist to calculate the visible part of the viewcell (e.g., [Hal02]). This information could be used, but algorithms are generally computationally expensive and might have numerical issues.

Perception starts playing an important role in simplification [WLC*03, LT00] and we would like to incorporate this kind of information in our approach. In particular, we want to investigate what visual errors arise from a representation. Due to the way rasterization works on current cards, billboards can have a very different appearance for grazing angles.

We would like to further explore other applications of our work, in the spirit of Cornish et al. [CRL01]. One idea consists in considering a light source as an observer to create simpler but more or less equivalent shadow casters.

Summary:

In practice this work might have less impact because heuristics often work well and in the industrial world, exactness or beauty of a formula rarely matter. When trying to sell algorithms, there is a strong waging of cost vs. benefit. In many situations a cheap hack is sufficient, although it does not give you the confidence that it will never break. In science however, hacks are insufficient and unsatisfying because it does not bring you knowledge. It is better to have the true formula and degrade to a fast approximation than trying to generalize a hack.

In this spirit, our work is interesting for several reasons. First, to point out possible artifacts and problems that can arise during simplification. Second, it defines the problem properly and gives a solution. We show how to apply point validity in 3D with arbitrary precision and how to transfer face validity for particular settings and thus deliver a workable and efficient solution.

Part III

GPU Voxelization

"Bah weep graaagnah wheep ni ni bong!"

Universal greeting of a Transformer

Chapter 6

Transformation

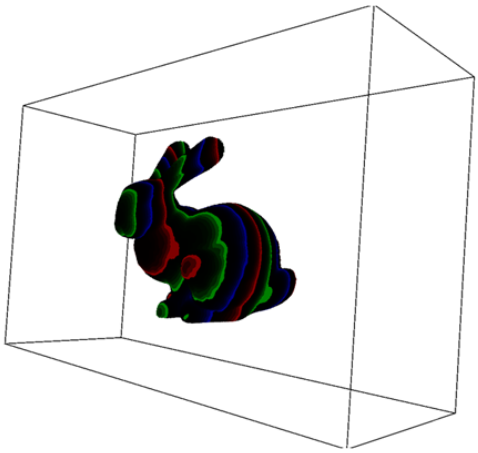


Fig. 6.1 : On the Fly Transformation of a Triangular Model - *into a voxel encoding of* $256 \times 256 \times 96$ *resolution.*

In the previous part, we analyzed preprocessing. In a preprocess the computational cost is less of an issue. It is thus interesting to increase accuracy and we focused on exact error bounds. On the other hand, preprocessing often implies that not all configurations can be considered. In the previous section, it meant that objects have to be considered mostly static and might only be deformed with affine transformations.

In real-time applications, scenes are often dynamic and follow no predefined behavior. Due to interaction, physical simulations or character movement based on artificial intelligence, configurations cannot always be predicted.

The major problem that blocks efficient computations is that the information misses structure. The basic data type usually involved when drawing on the screen is only a list of triangles and their indices. This is often inappropriate for other purposes because it contains, but does not directly give information about the actual spatial configuration.

To allow a structured access, which is necessary for many calculations, hierarchies would need to be maintained or build, what is usually a task executed on the CPU and thus quickly costly.

Another problem is that, the more complex a scene becomes, the more complex the data structure is going to be, and the more difficult it becomes to maintain these. In many situations, we would like to extract information about a scene at some fixed scale compatible with our computations and independently of the actual representation of the scene. Triangles are often difficult to work with because of strongly varying aspect ratios, sizes and orientation.

Further, it is important to respect the barrier between CPU and GPU. Some alternative representations might not even exist as such on the CPU side. The same problem occurs if models are deformed on the GPU, which is currently the standard way of animation because then the information will not be available on the CPU neither and only costly transformations mimicking the GPU would lead to a consistent data set on both ends. It is thus important on which side of the architecture a representation is created. Many computations are now performed by the graphics hardware.

For these reasons, we need new possibilities to capture a scene and transform it into a uniform and simple representation in each frame that can then be exploited for various tasks. Because of the importance of graphics hardware as a processing unit for an increasing amount of tasks, it should be GPU adapted and live in this realm. Our proposition is to create a voxelization in each frame. This representation is advantageous for many optimized computations (some we will illustrate in Chapter 8), is very simple, uniform, and, relative to the resolution, very memory efficient. The information we extract is mostly binary (even though we show some limited extensions), therefore, we currently miss much of the information of the scene, but the data we extract is very rich.

Visibiliy is at the basis of many applications and one of the factors that quickly make computations complex. By approximating the shape, even if it is solely binary, we also approximate visibility. How strongly shape and visibility are related can be seen on a large body of literature where models are recognized based on visibility events. Interestingly, in many mesh repositories, binary voxelization is used to classify and group models. Having a fast way to produce and compare voxelizations is very valuable in this context and of importance for larger data bases. Both tasks are supported by our techniques.

The second information we provide are normals. This opens up the road for many other applications and is a step further in the direction of capturing a scene in a uniform manner. We believe that light transport could benefit from this representation and more general collision detection would be possible. Nevertheless, this was out of the scope of our work.

In summary, transformation is a very powerful way of addressing complexity. With the increasing computational power of graphics hardware it becomes possible to use supplementary renderings that only reorganize scene data. The interesting point of this work that it delivers a unified representation, independent of the input, at a variable scale, that is rasterized. This makes it directly accessible by and compatible to several rendering paradigms of the GPU. Its uniformity, often eliminates special cases or distinct treatment of particular situations, as is often necessary for more complex representations. It thus simplifies data and facilitates many algorithms.

"Quadratisch. Praktisch. Gut."

Ritter Sport Slogan

Chapter 7

GPU Voxelization

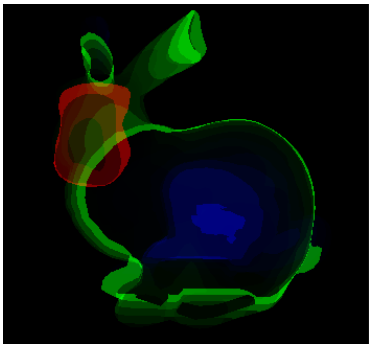


Fig. 7.1 : *Voxelized Bunny*

In this chapter, we will explain how to use graphics hardware to dynamically calculate a voxel-based representation of a scene. The voxelization is obtained during run-time in the order of milliseconds, even for complex and dynamic scenes.

The result is created and stored on the GPU, avoiding unnecessary data transfer between CPU and graphics hardware making the applications that we will present in the next chapter very efficient.

The approach can handle both uniform, and perspective grids. We also introduce a means to modify the grid structure along the z-Axis, which is useful for some applications. With our solution, we support regular and introduce locally optimized grids that better fit the scene geometry.

Further, we explain how to achieve solid voxelization for watertight models, as well as accurate, conservative voxelization.

The obtained information is binary, thus indicating the presence of matter in the scene at a very high resolution ($> 10^9$ voxels are possible at $> 90Hz$ for scenes with 300,000 polygons on a GeForce 8). We will also describe the derivation of a local density field from this representation which by nature defines normals via its gradient. Its storage cost is very low and the computational performance high, due to an adapted data layout and parallelized computations.

Publication notice: *The content of this chapter represents a first part of the articles published with Xavier Décoret. The first work [ED06a] was republished as a sketch [ED06b]. The second paper was accepted to Graphics Interface 2008 [ED08b].*

1 Introduction

Many tasks go beyond pure display. Interaction becomes more problematic when the number of primitives increases. This makes alternative representations important, one of which is voxels.

The popularity of voxels comes from their simplicity, regularity, and general advantages of volumetric representations [Gib95, He96].

Voxels have a long history in volume graphics and are still of importance in medical visualizations, representing the acquired data from CT scans. A variety of fields exploit voxel representations including effects such as shadows [KN01], CSG operations [FL00], visibility queries [SDDS00], collision detection [LFWK04,HK06], or even to resolve occlusion issues during the 3D reconstruction of a scene [OTT98]. Voxels are three-dimensional entities that encode volumetric information, as opposed to boundary representations such as meshes which only describe the surface of objects.

Converting between voxels and boundary representations is a well-studied problem. For example, marching cubes can extract a surface from a potential function defined over a voxel grid. For a long time, however, the inverse has been a costly task often performed in a preprocess: a model is placed in a volumetric grid and approximated by storing information representing the geometry in each grid entry. The particular binary voxelization only uses a Boolean (indicating the presence of matter). Here, we will present an efficient approach to perform the task of binary voxelization from an input mesh.

2 Previous Work

Early approaches voxelized based on point queries against the model [LR82]; even today this is not practical for larger models. Haumont and Warzee [HW02] presented a robust approach that deals with complicated geometry, reporting computation times on the order of seconds for typical models. The same holds for [NT03], where a layered depth representation is derived to count the parity of intersections from each voxel center in the projection direction. We avoid extracting and storing the layered depth images and achieve real-time performance with accurate point sampling.

In our approach, we sample a binary response at the resolution of the voxelgrid. This leads to aliasing any binary sampling's artifacts. It relates to aliasing of standard rasterization where mostly super-sampling is used to hide this problem. In the same spirit, the high resolution of our voxel grid allows us to capture most details, and we derive a smoother density estimate. Nevertheless, supersampling only hides, and does not solve the problem. An alias-free voxelization is presented in [WK93], involving an expensive distance calculation for all primitives. In practice (even for medical applications), a binary representation of $< 256^3$ often seems sufficient [RBB03]. This situation cannot be generalized, because other sets might need a precision of several billion voxels.

Recently, approaches have been suggested that benefit from the tremendous performance increases of graphics hardware. Chen and Fang [CF99] store binary voxels in a bit representation using clipping planes and a transfer into bits of an accumulation buffer. Normal estimation is done by replacing the accumulation texture by slices of a 3D texture (bits become floats). Our approach

gives normal information without this costly (in both time and in memory) process. Even on the latest cards with 1 GB of memory, no Giga Voxel volumes would be possible.

Previous work could not extract large numbers of slices with normal information in real time [Dro07]. Modern hardware can extract a small number of property layers in one pass (currently 8 RGBA buffers in DX10); this decreases the grid resolution or increases the number of passes. Further, their solution uses REPLACE blending to avoid incoherent results, thus keeping only the last value of a written fragment in a voxel. Blending operations are not programmable, and evidence suggests that they will not be for a long time due to optimization issues. Thin elements can thus have front and back faces, fall into the same clipping region, and have only one normal stored. *Thin* refers to a 16th of the scene, because these techniques rely on around 16 layers due to performance issues. This is problematic when particles interact from all sides. Our attribute extraction is limited, but uniform and normals are based on the voxelized shape, leading to coherent values.

Karabassi et al. [KPT99] and Kolb and John [KJ01] use simple depth maps to deduce voxel information and cannot handle concavities correctly. Depth peeling [Eve01] is used in [LFWK04]. Arbitrary surface attributes can be trivially obtained. The number of peeling passes is unknown because they are object *and* viewpoint dependent. This implies the need for a high amount of texture memory especially if extra attributes are retrieved. It also involves a costly occlusion query after each peel. Typically, low depth complexity can be handled. Furthermore, to evaluate the voxels efficiently, all pixels are reprojected from the extracted layers into a uniform representation. ≈ 250.000 vertices are scattered per layer for a 512×512 resolution. Holes may appear for grazing polygons because fragments define one voxel, whereas the depth range they represent might be larger. Depth peeling from several viewpoints tries to address this problem, but it also overwrites concurrent information. Our method handles grazing angles automatically.

The slicemap is an approximation of the different layers of matter visible from a camera. In a sense, it is related to Layered Depth Images (LDI) as described in [SGwHS98]. LDIs are more general and store arbitrary information per layer, but are much more costly to obtain, usually using image warping or ray-tracing in a pre-process. Hardware acceleration can be used by performing *depth peeling* as described in [Eve01]. However, this requires several renderings of the scene. Potentially the number of layers depends on the viewpoint, and should be fixed in advance to avoid varying framerates when using occlusion queries. Detecting matter in a depth layer can also be achieved using occlusion queries [LWGM04], but once again this requires several renderings. Recently octrees have been constructed on the GPU [LHN05b], but instead of filling the octree with scene geometry, it is used as a structure to perform texture mapping.

Fast solid voxelization on the GPU was first presented by Dong et al. [DCB*04], who propose a flood-fill along the third dimension. Their algorithm fails in cases where two fragments fall in the same voxel. Eisemann and Décoret [ED06a] perform solid voxelization in a single geometry rendering pass. Voxelizations of front- and back-facing polygons, and a special texture allow us to derive the enclosed space without explicit flood-fill. Ambiguous situations occur when several front- and back-facing triangles fall in one voxel.

Recently, the importance of solid voxelization was shown in the context of interaction with fluid/gas simulations [CLT07, Lla07]. The stencil buffer alternates between one and zero to find the parity of the number of intersections towards the eye. This was also used in [CF99], where the stencil buffer is not reinitialised when passing from one voxel slab to the next. Both approaches are limited to the extraction of one voxel layer per rendering path. Our solution provides ≈ 1000 binary layers in a single pass (and 128 even on older hardware (GeForce 6 series)).

GPU-conservative voxelization is a problem that was solved recently in [ZCEP07]. The authors derive depth extents for fragments and transform them into bitmasks using a 2D texture lookup. Conservative rasterization is also used in [HLTC05]. The approach is accurate, but slow even for small resolutions, as only one slab of voxels is created per render pass. We focus on conservative *solid* voxelization. In this case, two conservative possibilities exist: voxels lie entirely in the interior, or voxels lie partially in the interior. Both are presented in Section 3.5. By default, our technique performs artifact-free and accurate sampling of the voxel centers.

Summary:

Voxelization is usually costly and, only recently, GPU solutions were proposed that are significantly faster. This opened up the road for many applications.

Unfortunately, most algorithms relied on a layer-wise extraction, which can become prohibitive because, usually, to extract a very small amount of layers many renderings of the entire scene are necessary. This holds especially for solid voxelization, where the existing work relies on the stencil buffer and thus extracts one layer at a time. Furthermore, memory transfers are often necessary before the data is in an adapted format that can then be efficiently accessed. Our solution addresses these issues.

3 Principle of the Slicemap

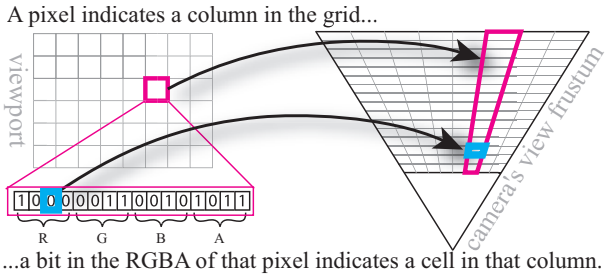


Fig. 7.2 : Grid Encoding in the Camera Viewport - *For clarity: 4 bits per color channel (16 slices)*

To voxelize a scene, a grid of cells is defined around it. The primitives are traversed and, for each of them, the cells they intersect are found. Our approach accomplishes this task efficiently with the graphics hardware, based on two observations. First, a rendered view of a scene implicitly defines a grid. The outline of that grid is given by the view frustum of the camera and its resolution is given by the resolution of the viewport and by the finite precision of the frame buffer. Second, when the graphics card renders a view, it does traverse every primitive. It can thus find the cells intersected in this implicit grid. Indeed, for every fragment produced during the rasterization of a primitive, the (x,y) pixel coordinates and z value indicate a cell. In classical rendering, the z value is used for hidden face removal and only the color of the closest fragment is kept. Although other fragments are discarded, the system has had access to it at some point. Our idea is to keep this information instead of discarding it, and to encode it in the RGBA channels.

3.1 Grid Encoding

We define a grid by placing a camera in the scene and adjusting its view frustum to enclose the area to be voxelized. The camera can be orthographic or perspective and can be placed at any

position. Then, we associate a viewport to the camera. The (w,h) dimensions of that viewport indicate the resolution of the grid in the x and y directions. A pixel (x,y) represents a column in the grid. Each cell within this column is encoded via the RGBA value of the pixel. Instead of considering this value as 4 bytes typically encoded on 8 bits, we consider it as a vector of 32 bits, each one representing a cell in the column. With modern hardware, 128-bit vectors can be obtained by using 32-bit data types per component, further MRTs allow a supplementary increase. To simplify explanations, we will restrict ourselves to the 32 bits first.

The division of a column into 32 cells can be done in different ways. The simplest, most natural one is to evenly divide the range between the near and far planes, but we will see how to improve upon this in some situations. Once a camera, viewport, and division scheme are defined, the corresponding image represents a $w \times h \times 32$ grid with one bit of information per cell. We will use that bit to indicate whether a primitive passes through a cell or not. Figures 7.2 and 7.3 show illustrations of the grid and the terms we will use.

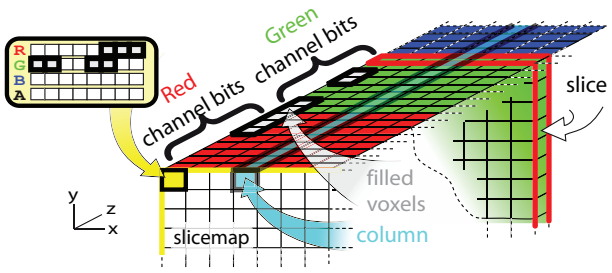


Fig. 7.3 : *Encoding in a texture*

The union for all columns of voxels corresponding to a given bit defines a *slice*. Consequently, the image/texture encoding the grid is called a *slicemap*.

3.2 Rasterization in the Grid

To construct the slicemap from a polygonal scene using graphics hardware, we render it into a texture using a simple fragment shader. The projection, modelview, and viewport matrices are set to match the chosen grid. The texture is initially cleared with black so all bits are set to 0.

For each primitive, we must find the voxels that it intersects and set the corresponding bits to 1. Rasterizing the primitive will produce a single fragment for each of the intersected columns. The depth d of that fragment indicates in which slice it falls. We use a fragment program to transform this depth into a 32-bit mask with zeroes everywhere except for a one in the bit corresponding to the slice. The depth values for fragments are in the range $[0,1]$ but the distribution is not uniform in world coordinates. Using this depth for slices would put too much resolution close to the near plane and not enough close to the far plane. Instead, we use the real distance to the camera's near plane by transforming the 3D position of the vertex by the transformation matrices. This distance is passed to the fragment shader as texture coordinates. Due to on-surface interpolation, a fragment obtains a correct z in $[-z_n, z_f]$ which is then mapped linearly to $[0,1]$ using:

$$z' = \frac{z + z_n}{z_n + z_f} \tag{7.1}$$

This normalized distance is used to perform a texture lookup in a 1D texture that gives the 32-bit mask corresponding to the slice in which z' falls. Currently, the texture lookup is much more efficient than performing the arithmetic in the fragment program. The resulting texture will be referred to as the *cellmask texture*. Its format is RGBA with 8 bits per channel to represent the 32 slices. Note that it is independent of the actual voxel grid's position and is computed only once.

It could even be included on the chip and provided as a function in shaders. Our convention for the cellmask texture implies that the values in the mask are between 2^0 for the nearest one and 2^{31} for the farthest cell.

The bitmask obtained from the texture must then be OR-ed with the color in the frame buffer to get the correct slicemap in the end. OpenGL's logical operation provides that functionality.

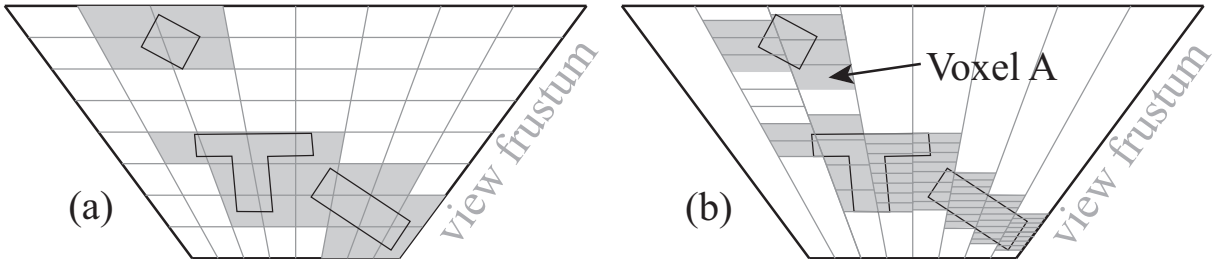


Fig. 7.4 : Uniform vs. local slicing

(a) Columns are sliced uniformly using the camera's depth range, yielding a "regular grid" and coarse voxels (b) Each column is sliced using local depth range, yielding a "distorted grid" and generally finer voxelization.

3.3 Uniform *vs.* Local Slicemap

As we mentioned earlier, there are various ways to divide a column of our grid in 32 cells. We just described a column-indepedent scheme that produces a *uniform* slicing, potentially wasting some resolution.

If the depth of fragments in a given column does not range from 0 to 1, we end up with useless cells in empty areas and cells too coarse to capture the details in other areas, as can be seen on Figure 7.4a. Equation (7.1) reveals that z_n and z_f could be chosen for each column independently in order to enclose the geometry in this column more tightly. To perform this local fitting we recover the scene extent for each pixel seperately by rendering two depth maps. These renderings could also be done using a simple bounding geometry.

These two textures will be called *near* and *far depth textures*. We then generate the slicemap as before by rendering the scene, applying a modified shader so that eq.(7.1) now uses *local* values of z_n and z_f. Figure 7.4b shows that it generally creates a finer voxelization, but locally voxels might be less fitting (compare voxel A in Figure 7.4-b).

Summary:

A 3D grid encoding matter can be conveniently stored in a texture, by interpreting the bits of the color channels as a position in space. Even more interesting is the fact that this representation can be directly rasterized via the graphics pipeline. Familiarity with the bit-encoding is of high importance for the understanding of the following sections on solid voxelization and conservative voxelization.

3.4 Solid Voxelization

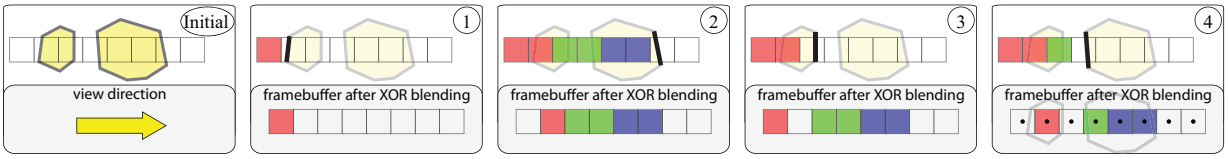


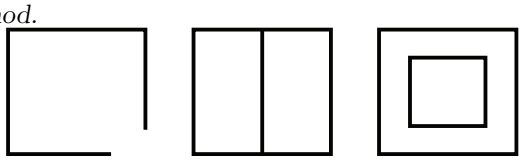
Fig. 7.5 : Solid voxelization overview

Solid Voxelization for a column in the slicemap. To simplify the illustration, only one framebuffer with two bit color channels is shown. Left: The scene, consisting of two watertight objects, is voxelized in the column along the view direction. 1-4: During rendering, fragments can arrive in an arbitrary order. For each fragment, a bitmask (upper row) is computed in the shader which indicates all voxels that lie in front of the current fragment. This mask is accumulated in the framebuffer (bottom, initialized at zero) using an XOR operation. Once the rendering is complete (4), the framebuffer contains a center-sampled solid voxelization in a grid shifted by half a voxel.

Our approach for solid voxelization relates to closed-curve filling in the plane [BP96]. To achieve high performance, we will exploit the definition of watertightness: a point lies inside an object if for any ray leaving the point, the number of intersections with the object's surface is odd. In particular, this condition holds for a view-ray and, for example, is used to test points inside a shadow volume [Cro77]. Therefore, determining whether a voxel lies inside the model amounts to counting the fragments f_i rendered in front of it.

Figure 7.5 illustrates the voxelization process. Let n fragments lie in front of voxel (i, j, k). It lies inside the model if n is odd ($n \bmod 2 = 1$). Consider for a moment that each voxel contains an integer counter and each fragment increments all voxels situated in front of it.

Watertightness *We are interested in a fast method to voxelize watertight models. Our definition of watertight follows the one in [NT03]. A model is watertight if for any connected component in space (separated by the geometry), all its points share the same classification: being in the interior or exterior. A point in space is considered interior/exterior if the number of intersections with the model of any ray originating at this point is odd/even (Jordan theorem). This definition excludes some models from being usable with our method.*



*The above figure shows examples where the definition of an interior is problematic. The left object exhibits a crack in its hull and, therefore, does not define a proper interior. The middle object contains a supplementary wall that separates the inner volume into two parts. Rays shot from one inner part into the other will intersect the model in an even amount of intersections, while shooting vertically leads to a single intersection. This model is thus not watertight in the above sense. The same holds if the wall coincides with the outer hull. Finally, the rightmost example illustrates a box englobing an inner box. Here, the definition implies that the inner box is an empty region. It is coherent, but not all models are conform to this. This is a limitation that our method shares with several previous works [WK93, RBB03, DCB*04, ED06a, CLT07].*

It is possible to use more advanced techniques in a preprocess to derive a coherent model which is adapted to our algorithm. This step could also exploit supplementary knowledge that is ignored by our solution, such as normals, if this information is accessible. Interestingly, our approach can be integrated into previous work to accelerate the derivation of a coherent mesh, e.g., [NT03]. Further, most correction methods derive an implicit representation of the input model. A triangulation based on marching cubes [LC87] is always compatible with our method.

Instead of letting the shader output a value having only a single 1 in the kth position based on the fragment's depth (as for the boundary voxelization), it returns a 1 in all positions smaller than k.

Adding this column to the corresponding column of counters in the voxel grid increments the value in exactly all those voxels where any ray along the view direction would intersect the incoming fragment.

Maintaining a real counter per slice is impractical on current graphics hardware. To make the accumulation work, we need a second observation:

$$n \ mod \ 2 = (\sum_{t=0}^{n} 1) \ mod \ 2 = \bigoplus_{t=0}^{n} 1,$$

where $\oplus$ denotes an XOR operation. In this form, the counters can be stored in a single bit, maintaining an in/out status. An incoming bitmask (based on a rasterized fragment) is accumulated by blending with an XOR operation. In practice, this bitmask can again be created by a lookup in a small 1D *cellmask texture* based on the fragment's depth.

Due to how rasterization is performed on current cards and our choice of the bitmask, the voxelization accurately samples centers of a voxel grid shifted by half a voxel along the z-axis. There is no imprecision or aliasing introduced due to the XOR operator. The shift comes from the fact that we choose the bitmask based on the voxel into which the fragment falls. Thus, the separations are naturally at the boundary between two column voxels. However, the offset can be counteracted by a adding half a voxel to the fragment's distance, thus virtually shifting the column.

Cutting Planes

As for shadow volumes, we must ensure that polygons are not clipped by the near and far planes of the rendering camera. *Depth clamp* (NV_depth_clamp extension) performs this operation by clamping depth values to the frustum, and thus outputting all fragments that otherwise would have been excluded by the near/far plane. As we count intersections of rays shot away from the viewpoint, the voxelization remains correct even when created from inside of a volume.

3.5 Conservative Voxelization

For non-solid voxelization, the found voxels do not create a "continuous" set: there might be holes. They will occur when the slope of the primitive in the z-direction is too high. The situation is illustrated Figure 7.6. This is a well-known problem when rasterizing in a grid [Bre65]. An even more extreme case can be encountered: if a primitive is perfectly aligned with the view direction, as shown on left of Figure 7.6, OpenGL will not produce any fragments and the primitive will be missed (a possible workaround for this situation is described in [AAM05]). Depending on the application, this may or may not be an issue. An important point is that because the slicemap can have a large resolution in x and y directions, the impact of those holes is usually not dramatic.

A similar problem arises for solid voxelization, although it is accurately point-sampling the geometry thin elements might not pass through the voxel centers and thus remains uncaptured.

Technically, the faces on each side fall in the same voxel; they result in the same bitmask, which is annihilated by the XOR operation. This makes sense: the resolution of the grid is lower than the details.

For some applications, like conservative visibility testing [DDS03], it might sometimes be necessary to fill even those voxels that lie partially in the interior. This problem is addressed by a so-called *conservative voxelization*.

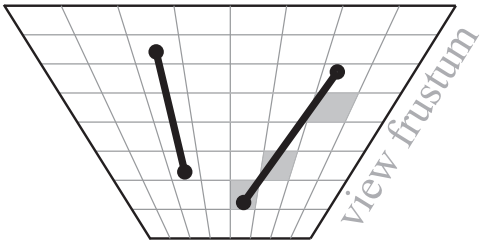


Fig. 7.6 : Slicemap Limitations due to Standard Rasterization - *The left primitive is missed because of alignment with the view direction; the right has a large slope in z and is not voxelized continuously.*

To obtain a conservative boundary voxelization we rely on Zhang et al. [ZCEP07]. The process involves conservative rasterization [AAM05] that creates fragments where polygons touch a pixel and further provides a corresponding depth interval (also compare Chapter 13). We do propose a slight improvement with respect to their solution. Our small 1D cellmask texture can be used instead of allocating large 2D textures (that become especially expensive for DX10 hardware) to transform the two depth extremes into a bitmask that encodes the voxelization. To conservatively activate voxels between these two extents, we shift the farther value by the distance of one voxel and leave the other unmodified. We then perform the lookup of the corresponding bitmasks and achieve a conservative filling in the column by computing a XOR in the shader (or equivalently a substraction for older cards). The offset we applied to the farther depth value ensures that all the voxels lying partially between these two extremes are activated. Finally, the result is blended into the buffer using an OR (not XOR) blending, leading to a conservative boundary.

The scene is rendered once with our standard algorithm. Then, in a second step, a conservative boundary voxelization [ZCEP07] is added into this same texture. The solid interior is simply kept in the framebuffer before adding the hull. This works because the logical operations are compatible. The result is a correct conservative solid voxelization because either a voxel is touched by the surface, and thus detected via the conservative boundary rendering, or all points (especially the center) of a voxel lie inside/outside. The only approximation comes from the currently problematic depth range derivation (compare [ZCEP07]).

For solid voxelization there is a second conservativity definition: one can also only keep the entirely interior voxels. To achieve this one can derive the conservative overestimate and then substract the conservative boundary voxelization by blending with an XOR operation. This delivers the conservative interior without boundary in three passes.

3.6 Grid Resolution

The number of columns in the grid is the resolution of the slicemap. Therefore it is limited by the maximum viewport size supported by the graphics card, currently 8192×8192. This (x,y) resolution is huge compared to the typical size of volumetric datasets, which rarely exceeds 512^3. On the other hand, for standard data types the number of slices is 32 and using 32-bit data types only results in a maximum of 128, which is rather small. Nevertheless, there are different

ways to bypass this limitation and increase the number of slices. The most natural one is to use additional slicemaps.

Every extra map requires a supplementary rendering of the geometry so there cannot be too many of them or the performance would suffer. Fortunately, we have Multiple Render Targets (MRT) to draw to several color buffers at once. We can nowadays obtain 8 MRT $\times 4$ channels $\times$ 32 bits $= 1024$ slices in one single rendering pass. Using two instances - a technique recently exposed in OpenGL - allows us to increase the resolution to 2192^3 with a single render call. This limit is imposed solely by memory. The shader stays almost the same.

On the 32^{nd} Bit *With the current drivers, it seems that the unsigned integer format reserves the 32^{nd} bit (possibly for exceptions). Values are not correctly compared via equality if it is activated. This could be an issue for applications using our technique. In our applications, we thus primarily used resolutions of $1024^2 \times 992$. The modifications to avoid the 32^{nd} bit are minor. For the voxelization algorithm, only the 1D texture needs to be modified to exclude the highest bit.*

In the following, we will downsample this texture and it is important to keep a size that is compatible with the sampling. In practice, we use 28 bits per color channel, which is the biggest multiple of four below 32, leading to 960 instead of 1024 initial slices. (When comparing our performance to previous work, we took the smaller amount of slices into account.)

Summary:

Very efficient binary voxelization with high accuracy is possible. We showed center sampled solid voxelization and showed how to obtain conservative voxelizations with the same principle. Only a single render step is necessary and the solution is very simple to implement. Using MRT's very high resolutions (e.g., 1024^3) are possible while maintaining a very high performance. In the next sections we will investigate how to derive normal information from the solid voxelization.

4 Our Method for Density/Normal Estimation

GPU voxelization is useful in itself, but in some contexts the binary nature hinders its usage. In this section, we will transform the binary grid into a *density representation*.

The contributions in this section are twofold. We provide a solution to *compute* and to *store* the density in a GPU-adapted manner, respecting parallelism and the limited memory. There is a variety of literature focusing on the question of how to choose an appropriate filtering kernel for the density estimate. A good overview is given in [ML94]. In our situation, because speed is a concern, we opted for a box filtering.

4.1 Overview

We filter and downsample the slicemap to construct a *density map*. Each density-map voxel contains a *non-binary* value which indicates how many filled voxels of the original slicemap it represents.

Our solution is general, as any power of two kernel can be obtained dyadically. To simplify, we will concentrate on a kernel of size 2. We perform a box filtering which computes the sum of 2^3 adjacent neighbors. The result is then (just like for mipmapping [Wil83]) stored in a downsampled version of the volume. Formally, we compute:

$$d(i,j,k) := \sum_{l,m,n=0}^{1} v(2i+l, 2j+m, 2k+n),$$

where v is the binary value (0 or 1) of the slicemap in the considered voxel. Consequently, d takes integer values in $[0,8]$. Zero indicates that the voxel lies entirely outside of the model. Eight implies that it lies entirely inside. The boundary between these two regions takes values in the range $[1,7]$. Division by eight leads to an approximate occupancy of the model, hence the name *density map*.

The implementation is a little more complex for several reasons. First, densities are no longer binary; we need several bits to store them. Second, we must organize computations for the GPU (e.g., minimize texture lookups, and optimize parallelism). Consequently, we need to intelligently layout the density map in texture memory.

4.2 Details of the Density Map Construction

Because of the downsampling, we will assume an initial slicemap of size $2w \times 2h$ stored in one integer texture with 32 bits per color channel. Thus the slicemap represents a voxel grid of size $2w \times 2h \times 128$. Next, we derive a density map of size $w \times h \times 64$, where each voxel contains a number in [0..8]. This requires 4 bits of storage instead of 1. Combined with a downsampling of a factor of 2 per axis implies a need of 256 bits per density-map column. Thus, the representation no longer fits into a single texture and needs to be spread over two $w \times h$ textures representing half of the density map's column (in practice, one tiled texture). Each texture only needs half a column of the slicemap to fill it up. As a consequence, the filtered result for the slices of the first 64 bits (two color channels) will be processed for one, the remaining 64 bits for the other. Figure 7.7 summarizes this mapping.

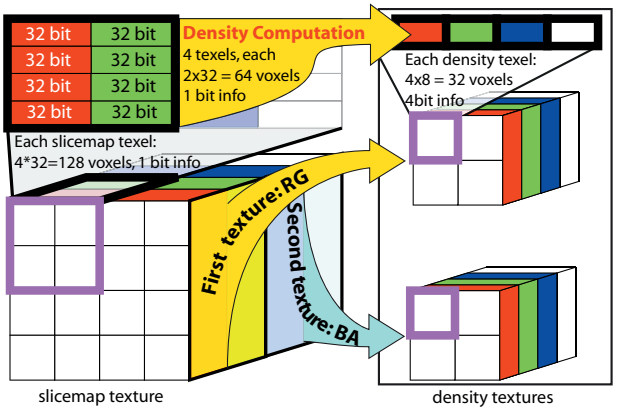


Fig. 7.7 : *Density Encoding*

We can now focus on an input slicemap representing a $2w \times 2h \times 64$ grid whose density will be stored in a texture of size $w \times h$ with 128 bits per pixel. The storage is sufficient, but the major challenge is to fill this texture efficiently because treating voxels separately is extremely slow. We will derive the density in two steps, first along a column, then spatially in the x,y plane.

To achieve parallel execution, we observe that the sum of two integers of n bits can be stored in $n+1$ bits. Bits higher than $n+1$ will not be touched. Adding the integers i_1^n with i_2^n and j_1^n with j_2^n (where n indicates the number of bits needed to store them) can be done by putting them in subparts of integers with $2n+2$ bits. A sum then actually evaluates two sums in parallel. This holds if more number are concatenated.

With this observation, we compute an intermediate *z-density map* via an in-place operation. It stores two bits per voxel (encoding the pairwise sum of neighboring voxels in a column c) and is given by:

$$(c\& \; I_{10101\ldots}) + ((c\& \; I_{01010\ldots}) \gg 1),$$

where & denotes a bitwise AND, and $I_{abc\ldots}$ an integer with bitmask *abc....* The succeeding zero bit (introduced by the mask) after every copied bit, ensures that the summation will not pollute the solution. This operation also performs the downsampling. Figure 7.8 illustrates this.

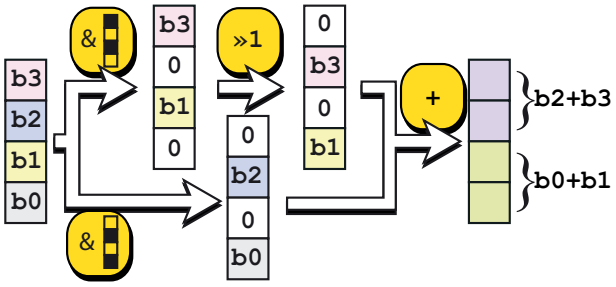


Fig. 7.8 : Sum along z - *Bits pairs store the result.*

The next step is to sum up voxels of the z-density map in the xy-plane. Four neighboring voxels need to be combined, where now each voxel is represented with 2 bits. Their sum will thus need 4 bits of storage and we cannot perform an in-place operation. Instead (to benefit from parallel execution), we calculate the sums of even and odd voxels seperately as:

$$ESum := \sum_{i=0}^{3} (c_i\& \; I_{00110011\ldots}) \gg 2$$

$$OSum := \sum_{i=0}^{3} c_i\& \; I_{11001100\ldots}$$

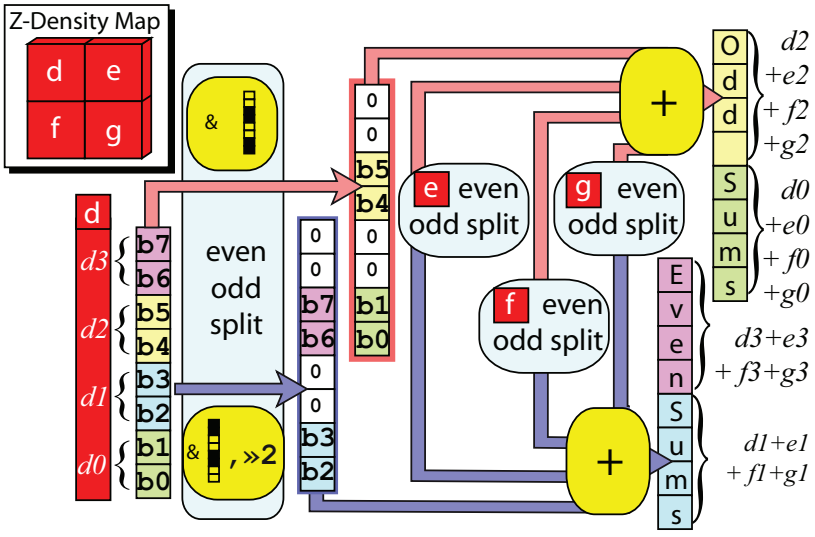


Fig. 7.9 : Summing the z-densities in the XY plane - *The columns of the z-density (d,e,f,g) contain the sum of neighboring voxels along the z-axis (encoded on 2 bits). The final density is obtained by summing four neighboring z-densities in the xy-plane. This is done in parallel by splitting the vectors into even and odd entries that are summed separately. (To simplify the illustration each column is represented with only 8 bits.) Note that the splitting leads to a result that is ordered 0,2/1,3 and not 0,1/2,3.*

i is iterating over the four neighbors in the xy plane. The masks do not only recover the right bit pairs, but they also assure that each is followed by two zero bits which are needed to assure a correct summation. The resulting integers *ESum* and *OSum* can then be stored in the density map. Figure 7.9 illustrates this step.

At this point, all entries of the density map are already computed. The only catch is that the voxels alternate along the z-direction between values in *EvenSum* and *OddSum*. For normal derivation, this is not problematic, for other applications it might be. To simplify usage, one can reorder the result in a parallelized process, detailed in the corresponding sidebox on page 115.

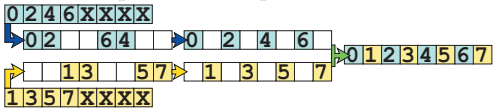
4.2.1 DX9 Implementation

On a DX9 system, the realization is actually simpler. All the above steps are encoded in lookup textures. We use a 256^2 RGBA texture that, based on two 8-bit voxel columns, gives the same configuration we obtain with the above algorithm, namely four 8-bit components that contain neighboring bit sums separated by two zero bits. These lookup values of the neighboring columns can then be safely added to yield the final density.

4.3 Normal Derivation from Density

Following the definition for an implicit surface, we can compute the gradient ∇d along the three axis via finite differences and derive the normal n from it by $\nabla d/||\nabla d||$. Each component has a value in $[-8,8]$, thus about 17^3 directions are possible. Removing those for which normalization leads to the same result gives a total of $4034 \approx 2^{12}$ distinct directions.

Reordering *two integers containing 4-bit groups representing voxels 0,2,4,6 and 1,3,5,7 is done in a parallel manner. The idea is to shift groups of information at the same time whenever possible. The figure below depicts the steps:*



We use a symmetric kernel needing 6 values around the center voxel. Five lookups are sufficient (the two neighbors in the same column are retrieved together). A simpler kernel, with only the center and three neighbors, would require 3 lookups, but the normals are of lower quality for an insignificant speedup. In [MMMY97] alternative normal derivations are discussed.

4.4 Strengths and Limitations

The slicemap has several benefits. First, it fits in one (or a few more, if MRT is used) texture. Thus it can be queried from vertex or fragment shaders and several useful applications are demonstrated in the next chapter. Second, it is generated by the hardware. Not only does this mean high performance, but also that the slicemap lives on the GPU and never needs to be transferred from or to the CPU. Moreover, it can be generated from arbitrary geometry that itself lives on the GPU (e.g., ray-traced volume graphics) and in particular geometry that is moved within a vertex shader (e.g., through skeleton animation). This means reduced bandwith, no copying of the geometry on the GPU, and no CPU work to transform or animate the geometry. In practice, on a GeForce 6800 TD, we are able to voxelize a scene of 1,124,571 triangles in a $512 \times 512 \times 96$ (3 MRT) grid with uniform resolution at 70Hz. If we use local resolution, we need two extra renderings and the frame rate is 60Hz. On a GeForce 8800 GTS, we achieve 90 Hz for 300.000 triangles and a really high grid resolution of 1024^3. As a comparison, the octree-based approach of [HW02] et al. takes ~ 3 min to intersect ~ 100K polygons with an octree of maximum depth 8 i.e. a 256^3 grid. Of course the entries of the octree representation are much richer.

DX10 (and recently also OpenGL) exposed the *render to texture array* extension which allows the storage of the slicemap directly in a texture stack for efficient access. This becomes interesting for the applications we present in the next chapter where we use the information stored in the slicemap to solve several otherwise costly problems. Another possibility is the rendering into a 3D texture. The latter is slower in access, but was available in DX for a longer time. Unfortunately,

DX does not support the bitwise blending operations needed for the correct accumulation of the fragments during the voxelization. It has been removed in a previous DX release, but indications exist that they might be reintroduced. For DX9, we use texture tiling to store the result of each pass.

The efficiency of the slicemap encoding and generation has a price. We can only store one bit of information per voxel, typically the presence of matter, but we cannot store the color or number of objects. It is possible to decrease the resolution and allow a few bits per voxel as shown in the next chapter in Section 1 but this is a very limited trade-off.

Often elements in a scene are heterogenous, meaning consisting of watertight and other objects (for example the cape of a character). The latter are not well suited for solid voxelization because they do not define an interior. It is possible, though, to perform a separate boundary voxelization and combine it efficiently with our solution for the solid parts. Basically, this corresponds to a union between the two representations and works even if these elements intersect each other. We show how general CSG (Constructed Solid Geometry) is possible, constructed using Boolean set operations on solid primitives. In particular, this includes the union of two objects.

Summary:

We combine a downsampling process with a parallel summation by packing numbers in sub-bit-blocks of integers. This results in high performance and much lower memory cost than in previous solutions and the normals reflect the actual voxel representation.

5 Results

5.1 Resolution and Storage

We tested our approach (implemented in OpenGL) for DX9 cards on a simple G6 NVidia 6800 (*non* GT/ULTRA), and on DX10 hardware with NVidia's G80 (8800 GTS). The latter supports 32-bit integer RGBA textures of resolutions up to 8192×8192, and can write into 8 MRTs in a single pass. We can represent a grid whose resolution in x and y is 8192 pixels, and whose theoretical resolution in z is $4 \times 32 \times 8 = 1024$ bits. The memory footprint of these 8×1M pixel textures amounts to 128 MB.

Prior to the G80 series, the integer types were not supported. Therefore, the behavior can only be emulated (in the shader, all values are floating point, no matter the texture type). As a result, only 8 bits per color channel are possible, leading to 32 bits total per texture. From the G6 series on, 4 MRTs are possible, thus allowing 128 usable bits per rendering pass. The texture resolution is limited to 4096^2.

One important implication of storing the information in bits is that in both implementations, the memory consumption is at least 8 times lower than for other approaches, e.g., [Dro07,CLT07].

5.2 Timings

As a test scene on the DX10 hardware, we used a Stanford dragon model with 262,078 triangles that almost filled the whole frustum. In a second test, we added four copies (leading to 1,310,390 triangles). Timings for solid voxelization are shown in Table 7.7. In the case where the interior is less dense and contains empty parts between objects, the framerate increases. For five dragons at 1024^3, the cost drops below 6 ms if only a fifth of the grid is occupied (which is the case when placing them with small separations). The timings for the DX9 card are shown in Table 7.6.

Table 7.6 DX9: Solid voxelization timings on NV6800

resolution	64^3	128^3	256^3	512^3
500 tris	0.19 ms	0.22 ms	0.6 ms	2,5 ms
5,000 tris	0.26 ms	0.33 ms	0.9 ms	3.9 ms
12,500 tris	0.36 ms	0.4 ms	1.1 ms	5 ms
25,000 tris	0.58 ms	0.6 ms	1.6 ms	6.3 ms
262,078 tris	3.5 ms	3.6 ms	7 ms	23.3 ms

Table 7.7 DX10: Solid voxelization timings on NV8800 GTS

resolution	512^3	1024^3
262,078 tris	1.6 ms	10.65 ms
1,310,390 tris	5.29 ms	41.5 ms

For DX10 the timings for the density computation in Table 7.8 depend solely on the resolution of the initial slicemap and include the reordering, which in comparison is not very expensive. In contrast, for DX9 the reordering is free, but the content of the slicemaps play a role because the cache comes into play due to dependent lookups. In practice, we realized that results seem to vary little (around 10%). Table 7.8 summarizes the timings as an average of several models.

Table 7.8 DX9| DX10: Density computation timings

res.	64^3	128^3	256^3	512^3	256^3	512^3	1024^3
ms	0.25	0.6	2.9	20.9	0.28	0.9	6.5

6 Discussion

Our voxelization method is simple to implement, and, as we will show in the next chapter, it allows for many interesting applications. It is a single-pass method and thus faster than its competitors, and contrary to the previous single-pass solution [DCB*04], stable even if an arbitrary number of fragments fall in a single voxel.

Although once exposed the solution to solid voxelization (especially after our solution to boundary voxelization) sounds simple, it must be pointed out that accurate single-pass methods did not previously exist. Between our first work on boundary voxelization and solid voxelization many attempts have been made to achieve this goal, but not a single one achieves the same quality and speed [LH08, FWLG08]. Also, contemporary multi-pass solutions are largely outperformed (as

we will show hereafter) [CLT07]. Our approach for solid voxelization is accurate (point sampled or over/under conservative), and its simplicity makes it appealing.

As for many CPU based methods, we do not need a manifold or a topologically coherent mesh. A watertight input suffices for solid voxelization, and the method derives the interior defined by the Jordan Theorem. If two watertight models intersect, our algorithm assumes the symmetric difference of the two, meaning that two concentric boxes will lead to a hollow representation, not to the union. This is consistent with the definition of watertight (see sidebox on page 109). In Section 4, we will show how to perform general CSG operations.

6.1 Comparison

With respect to previous work, our algorithm performs at higher speed (see Figure 7.10). In all tests, the frustum was fit to a bounding sphere. For a fair comparison, we optimized [CLT07] to work directly on the texture using logical operations (without using the stencil buffer). We also followed their future work suggestions, and tried instancing and texture tiling to accelerate the approach. In practice, it turns out that instancing is less interesting from a performance standpoint because the shaders are slightly more complicated.

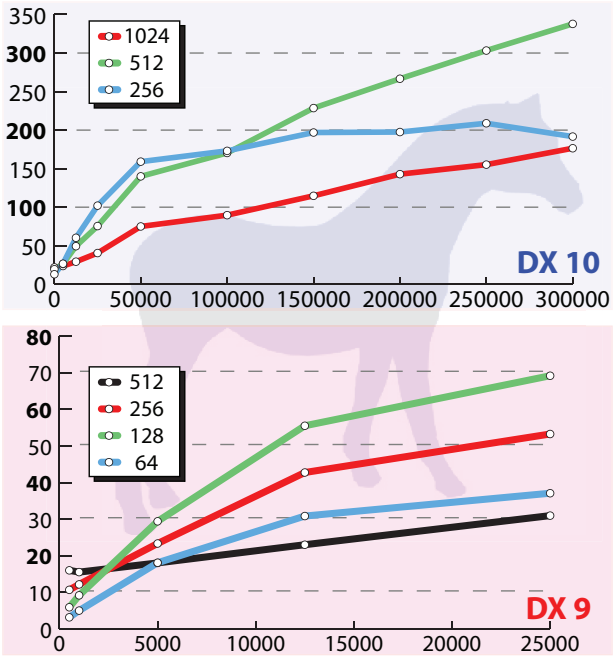


Fig. 7.10 : Speedup with respect to [CLT07] - *Top: DX10 (G80) Bottom: DX9 (G6)*

The difference to [CLT07] is remarkable for complex models (up to a factor of > 300, thus about 2 orders of magnitude). This stems from the fact that several layers (up to 1024) can be extracted in a single geometry pass, and thus the theoretical speedup is 1024. In practice, the fragment output still has some cost and the blending comes at an expense too. Nevertheless, even for a simple cube (12 triangles) we gain a factor of around 20. The cube is a type of worst-case scenario as its geometric complexity is low and fill-rate high. Standard rendering is very cheap in this case and already runs at $> 100Hz$ for a 256^3 volume, while the framerate drops quickly for high polygon models and voxel resolution.

For the DX9 implementation, we also obtain strong speedups even for less complex models. The cards are less powerful in treating geometry than the recent generation, which comes in part from the architecture. Now, shading units are stream processors reassignable to the fragment or vertex shader according to the workload. In the old generation, the number of vertex units was fixed and thus our algorithm compares quickly favorable with respect to [CLT07]. Figure 7.10 illustrates the obtained speedup for several levels of details of a horse model with 300,000 initially triangles.

Finally, our method is also interesting for even older cards without MRT support (e.g., GeForce 3). To show the advantage, we ran the tests deactivating the MRT feature of the G6, still allowing 32 layers to be extracted per rendering. Surprisingly, performance remained about twice the cost compared to four MRTs for models up to 12,500 triangles and converged towards a speedup factor of 23 with respect to [CLT07]. Generally, MRTs come at some cost, but one reason why the difference is not closer to four could be that the G6 had a sweet spot concerning its MRTs. Already, four MRTs impose more than the expected 33% supplementary workload with respect to three MRTs, and could explain the behavior. Exact performance depends on many factors including the chosen model, but the measured tendencies remained the same throughout all our tests.

6.2 Density and Normals

Our density computation is fast and memory efficient because all the information is tightly packed in bits and evaluated in parallel. In contrast to other methods [Dro07, LFWK04], we do not rely on the mesh's normals, but those defined by the voxelized surface. This is key to obtaining high resolution and normals coherent with respect to the voxelized volume. As described in Section 4.1, the kernel we use for our density estimate is of size 2^3. We found, just like [NT03], that even this small kernel gives acceptable quality in practice.

Our DX9 implementation is well adapted to those cards, because all four components of a color will be treated equally during the process, which reflects the vector capacities of its processors. It would *not* scale for DX10 hardware. A lookup texture for a 32-bit integer would need 2^{32} entries and does not fit into the memory of the GPU. Breaking the integers down into smaller parts of 16-bit lengths would still imply two lookups per channel, leading to a total of $8 \times 4 \times 2 = 64$ lookups per 1024-bit voxel column. In this case, our bitwise arithmetic proves more efficient.

The density can be seen as a localized version of distance fields. Much work has been published in this area, and in particular GPU implementations exist [SPG03, SGGM06, SOM04]. The larger support of these distance functions allows a more general usage and can make them preferable to local density. On the other hand, our method is faster to compute and interesting for applications that need only limited distance information, some of which are going to be presented in the next chapter.

Our method does not need to store normals explicitly because the gradient computation is not very costly and in the context of a (one million) particle system, the number of issued queries is much lower than the size of the density map (512^3). In practice, the proposed simple scheme leads to sufficient precision and allows us to evaluate the normals on the fly.

Summary:

Slicemaps allow a fast, hardware-assisted determination of binary voxels for arbitrary and dynamic scenes. It delivers a compact, GPU-friendly encoding into textures.

Our solution outperforms previous approaches in terms of speed and quality. We showed how to obtain a point-sampled solid voxelization, as well as a general conservative solution. The basic method is easy to implement, does not need knowledge about the scene geometry (only a depth value has to be produced), and is compatible with shader-based transformations.

Our solution to compute a density function is a quick way to derive something resembling a local distance field, which is usually costly to obtain. The normal estimate based on it is more efficient, and simpler to handle than depth peeling, they are of good quality, but less accurate than the original normals, since depth is quantized.

We introduced local precision, a feature which is perfectly suited for use on graphics cards, along with MRT to increase resolution. Our approach benefits from the structure of the graphics cards and can integrate with shaders to produce various new algorithms. Some of which we will present in the next chapter.

"There are no such things as applied sciences, only applications of science."

Louis Pasteur

Chapter 8

Real-Time Applications Based on GPU Voxelization

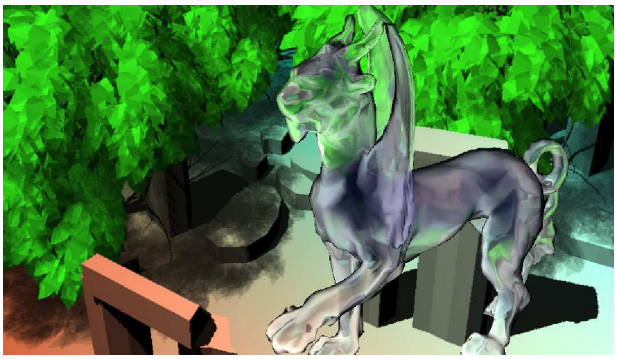


Fig. 8.1 : *Several effects of our technique in one figure.*

In this chapter we examine several applications of our GPU voxelization from the previous chapter. The variety is very large, ranging from transmittance/translucency effects, to shadow-volume clamping, to collision detection, and we sincerely believe that there are more to come. The idea of extracting global information will be of huge importance in future research.

The original motivation for the work on slicemaps was the application to CC Shadow volumes [LWGM04] and visibility in general. We wanted to find a way to quickly query information about the presence of matter in a scene.

For many applications the real volume is of broad interest: for example, to calculate repulsion forces or morphological operations which are important for many tasks (e.g., visibility determination, path finding). The nature of our output was initially binary and did not provide supplementary information about the surface properties of an object. This led us to work on the derivation of normals.

We are convinced that these representations could be of major interest in many ways. One application could be photon mapping [Jen01]. The benefit is that the scene is automatically structured in a regular way, which potentially allows for important optimization possibilities (a simple hierarchical solution is already presented in this chapter).

1 Transmittance Shadow Maps

A standard shadow map [Wil78] stores the depth of the closest occluder along sample rays. Points along that ray are classified as shadowed or lit by comparing their depth against the stored one. Instead, a deep shadow map [LV00] stores for each ray a one-dimensional function describing the light intensity along the ray. This technique achieves realistic self shadowing for very complex volumetric structures like hair. Points are shaded continuously, based on their position, by evaluating these functions.

Deep shadow maps [LV00] account for three phenomena: transmittance of semi-transparent surfaces (e.g., tinted glass), partial occlusion of the light beam by thin occluders (e.g., hair strands) and volumetric extinction (e.g., fog). In this section, we describe how slicemaps can be used to render the first effect. Partial occlusion, the second phenomena, could be handled, as done in general, using a higher resolution and interpolation. Section 1.1 will describe how they can be used to render volumetric effects. The approach in this section also relates to Opacity Maps [KN01], which do not aim at real time-applications. The scene is decomposed into layers using several render passes to obtain a local opacity value, and the values between the maps are interpolated linearly (which is also possible with our approach).

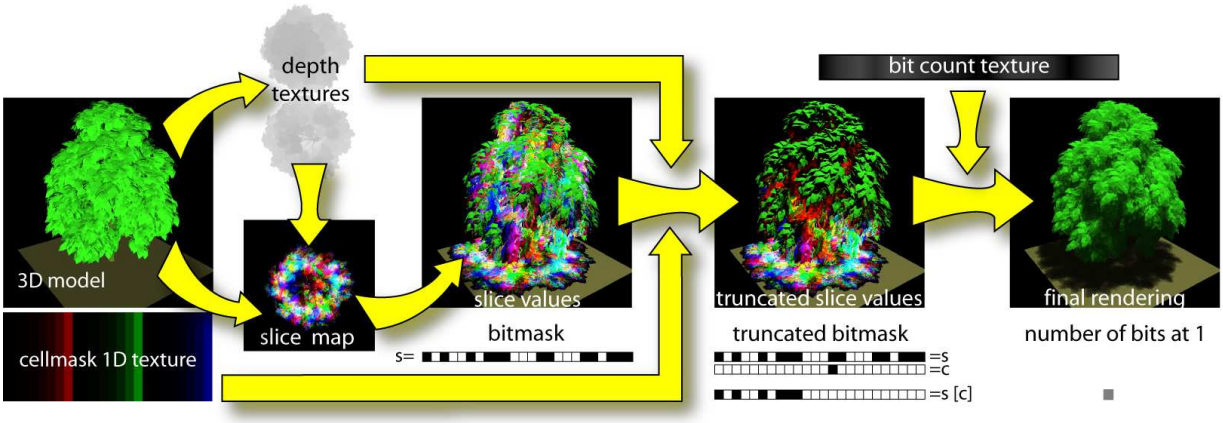


Fig. 8.2 : Overview of the transmittance shadow-map algorithm

here with 24 slices stored in RGB channels of 8 bits

The approach is summarized on Figure 8.2. Let's consider the foliage of a tree lit by a point light source L. If we neglect indirect illumination and refraction, the irradiance at a point P on a leaf is found by tracing the ray to the light source and summing the contributions of all traversed leaves. If we assume all leaves attenuate the light with the same factor, it amounts to counting the number of leaves intersecting the open segment (P,L). For that, we render a local slicemap from the light source. To shade a point P at depth z, we do a projective texture lookup into the depth textures to get z_n and z_f. If $z \leq z_n$ the point is fully lit. If $z_n < z$, we first retrieve the cellmask $c = 2^i$ such that P lies in cell i (cf. Section 3.2). We then do a projective texture lookup to get the bitmask s from the slicemap. In that bitmask, we must set all bits j to 0 with $j > i$. Indeed, such bits correspond to cells further from L than P (cf. Section 3.2). Mathematically, this corresponds to a modulo operation $\bar{s} = s \mod c$ (e.g., `fmod` in a Cg shader). Finally, we need to compute the number of ones in $\bar{s}$.

Counting could be done by looping over bits, which is highly inefficient, or using a precomputed texture, which is difficult for integer textures as it would result in 2^{32} entries. We use a different method, whose complexity is logarithmic in the number of bits. We duplicate the bitmask, zeroing odd bits in the first copy and even bits in the second. We shift the second copy and add it to the first. We repeat the process $\log_2(n)$ times to get the result. Figure 8.3 illustrates the principle. More details can be found in [And].

The final shading is then computed as $(1-\sigma)^n$ where σ is the attenuation factor for a leaf and n the number of filled voxels encountered on the way to the light.

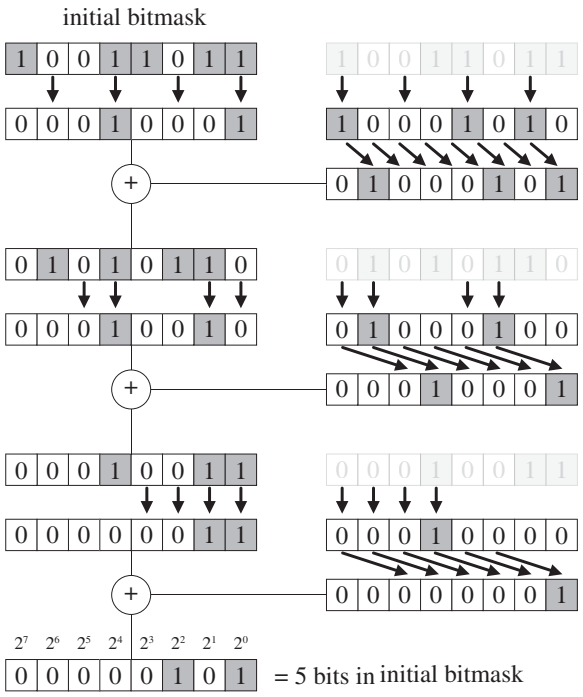


Fig. 8.3 : *Logarithmic-Time Bit Counting*

We implemented this algorithm on a GeForce 6800FX Ultra. We used a resolution of 512×512 and 3 MRT buffers which gives 96 slices. Our system combines semi-transparent objects and opaque objects (e.g., the trunk of the tree); the shader does an extra lookup in a standard shadowmap generated with only opaque objects. If the point is not shadowed, we continue with the transmittance shadow map generated with only the semi-transparent parts. We tested it on a tree model containing 160,653 polygons. Figure 8.4 shows the drastic difference between our transmittance shadow mapping and standard shadow mapping. Note the variation of shadow intensity in the foliage, which makes the shape of the tree a lot more perceptible. Attenuation effects can also be observed on the ground and can be changed dynamically by varying σ. There are fewer leaves close to the silhouette of the tree, thus the shadow becomes less pronounced. The rightmost images show an interesting variation where we trade off slicemap resolution for increased number of bits (therefore information) per voxel. Simply put, using 3 MRTs we can make a slices for green, reddish, and yellowish leaves in a single pass.

The system runs in real time as shown on table 8.1. It scales well with the geometry, as it is mostly pixel-shader bounded. In particular, we evaluate the shader for hidden fragments. For a model with high depth complexity such as our tree, deferred shading [ST90] could even reduce the cost for the fragment shader by performing an early z-culling. In our experiments, we found that if we first render the scene to fill the z-buffer before the final rendering, we get a 20% increase in speed. This is due to the fact that the driver detects that our shader does not modify the depth of fragments and can perform culling before shading, which is thus evaluated only for visible fragments. If real deferred shading is used, even $50\% - 80\%$ is possible (compare Chapter 2). In the measured timings above we still lost performance, because shadow maps are used pbuffers. Depth recovery using framebuffer objects did not work properly on a GeForce6800 Ultra.

Our method is similar to the deep shadow approximation in [ND05]. They can have four slices without and 16/32 with MRT. Our approach has 32 slices without and 128/1024 with MRT

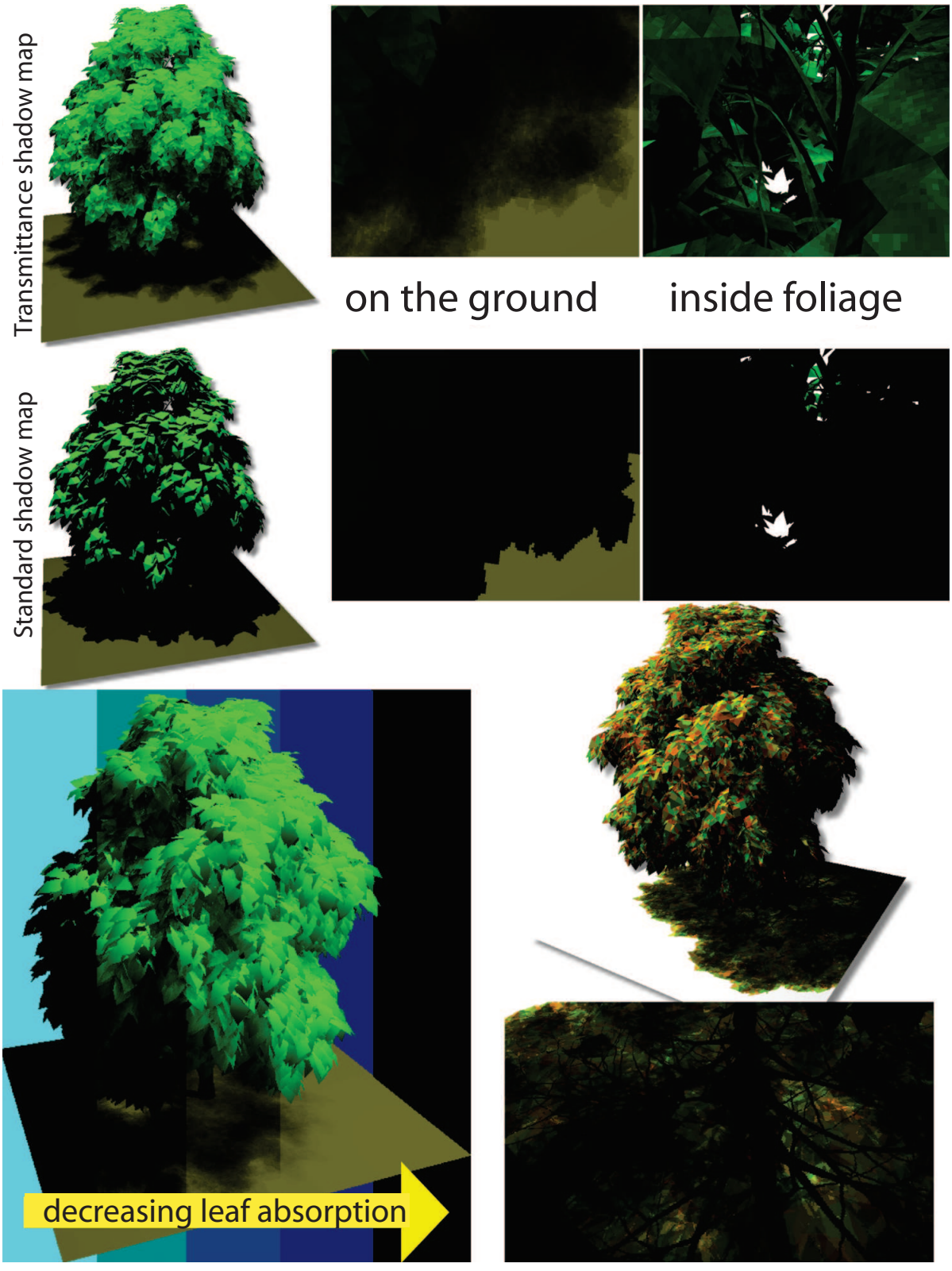


Fig. 8.4 : Transmittance shadow maps

Upper part: difference between standard shadow map and our approach. Close-ups on the ground emphasize the difference. Lower left: combined view of transmittance-based shadows with a different absorption coefficient. Right: using three different leaf colors, transmittance shadow maps can achieve colored shadows. Notice the multi-colored shadows cast on the ground.

Table 8.1 Frame rate (Hz) and timings (ms) for standard shadow mapping and transmittance shadow mapping (TSM) with uniform and local slicing. For TSM, we give the timings for 1, 2, and 3 MRT. Near, far and slicemaps are computed only for the transparent geometry. The tree model contains 160,653 polys of which 1,493 are opaque.

chestnut tree	SSM	TSM	
		uniform	local
frame rate	128	60/50/40	37/29/24
opaque map	3ms	<1/<1/<1 ms	<1/<1/<1 ms
near map	-	3/3/3 ms	3/3/3 ms
far map	-	-	2/2/2 ms
slicemap	-	2/4/7 ms	7/9/14 ms
shading	5	14/16/18 ms	20/28/32 ms

on a GeForce 6/8. On the other hand, if one wants (as they do) to treat partial occlusion via linear interpolation, it adds some extra cost to our solution. Our method approximates the one-dimensional transmittance function. The function is evaluated at equidistant samples instead of the non-uniform sampling of the deep shadow maps. Only the first and the last samples are placed at the exact location.

Our transmittance shadow maps are also closely related to [BMC05]. Here self-shadowing for hair is performed at interactive rates for directional light sources. The authors also create a voxel grid, but theirs is completely uniform. Each hair strand is sampled by points, and each point is transferred into the grid via the CPU. One difference is that their voxel grids can contain arbitrary density values, whereas we either use our binary result or rely on a filtering process. On the other hand, our grid usually contains much more voxels (other more recent GPU implementations of this approach rely on only 16 layers [KHS04]). No point sampling has to be performed: our grid automatically encloses the object tightly. It does not have to be adjusted in each frame and we can naturally treat point light sources.

1.1 Refraction, Frosted Glass, and Translucency

The voxel representation can be used to calculate an approximation of the volume traversed by a ray. Based on this, distance refraction can be increased, scattering can be approximated, or colors can be shifted towards the color of the object to simulate gaseous effects or even caustics [WD06].

In [Sou05] a simple approach to obtain reasonable refraction effects has been presented, and our work is inspired by this approach. Based on the surface normal where the eye ray hits the object, a lookup in an environment texture is perturbed. The idea of taking volume into account has been presented in [Wym05]. A normal and a depth map for the closest and the farthest surfaces are calculated and for each vertex of the object a depth along the normal is precalculated. The algorithm perturbs rays based on an interpolation between the precalculated depth and the difference of the depth maps. This information is then used together with the two normal maps to obtain the final ray. One major problem is that using the closest and farthest surfaces to approximate the volume traversed by an eye ray is correct only for convex objects and will give strong deviations in other cases.

In this section, we present our solution. It requires no precomputations and uses more reliable information about the actual volume traversed by an eye ray. It simulates three effects: refraction, attenuation translucency, and finally scattering related to the traversed volume. The input model in this case should be watertight to define a volume.

Fig. 8.5 : Refraction/Absorption

Images are obtained at $> 200 fps$ *in a resolution of* 512×512 *on a GeForce 6800 Ultra*

Similar to the previous section, the traversed volume can be queried at any point in the scene. This is the major difference when compared to solutions that compute the traversed volume by accumulation. These methods attribute the negative distance to the eye for each front-facing fragment and the positive for each back-facing fragment. Summing the values results in the correct volume that is traversed by a ray to infinity. For the eye-ray, this is possible, but if we want to query information about the light path somewhere on the eye ray, it becomes impossible. In the next section, we will see such an application in the context of CT rendering. Because our refraction model is quite simple, some artifacts are observable, such as the exact borders of those parts of the object that shine through.

We hav, nevertheless found this very acceptable, especially in comparison with a simple volume estimation only based on depth maps which yields an unrealistic appearance in all concave regions. In our implementation, all parameters for refraction indices, color, and absorption can be changed dynamically. If these parameters are intended to be constant, the shaders could be simplified and the frame rate would even increase. Some results are illustrated in Figure 8.5. Using the same principle, we can also add translucency effects; Figure 8.6 shows a result.

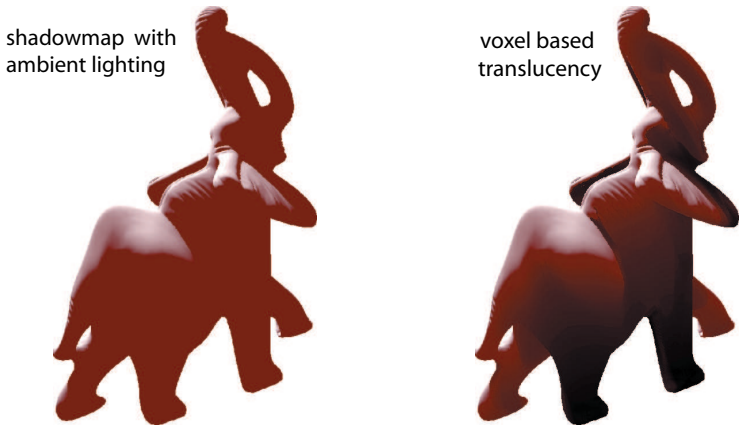


Fig. 8.6 : Translucency Effect - *Even for more complex models with 60,000 triangles, the framerate exceeds* $200 Hz$ *on a GeForce 6*

The presented model is a first step in the direction of exploiting the voxel representation for volumetric effects. It could be interesting to combine our approach with translucent shadow maps [DS03], or to write the volume values in a first step in a texture, where they could be further processed (e.g., filtered), which would then allow the object to have an influence on itself.

2 Visualization

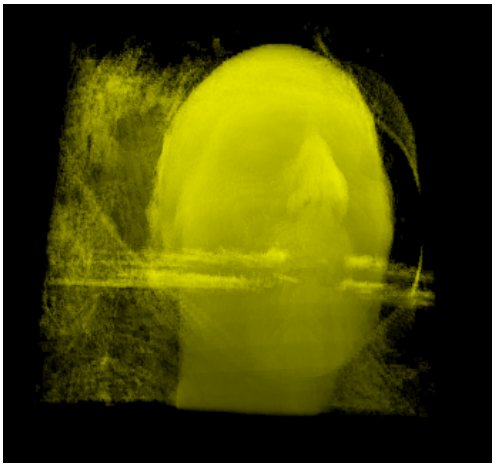


Fig. 8.7 : Volume Rendering - *Translucency increases volumetric appearance*

Compactness is one interest of slicemaps. Estimating normals compensates to some extent for the lack of other than binary information, and provides more surface data in real time. This becomes useful when visualizing level sets of large CT scans (1024^3). Usually, this amount of memory would not fit on the card, making interactive visualization via 3D texture slicing [IKLH04] impossible; the pipeline would stall with memory transfers. Marching cubes [LC87] are also not feasible at this resolution. A slicemap of a level set is small and can be created by one slicing pass on the GPU or transferred directly from the CPU. The display is interactive, and several level sets can be kept at the same time and blended together. Figure 8.7 shows the translucency method applied to a CT scan; rendering is performed using ray-tracing, where in each step the shadow contribution of the volume is evaluated and accumulated with the values encountered on the current view-ray. This technique is also used used in other sections, such as Figure 8.11 and 8.13 (where false colors represent our normal estimate).

3 Shadow Volume Culling and Clamping

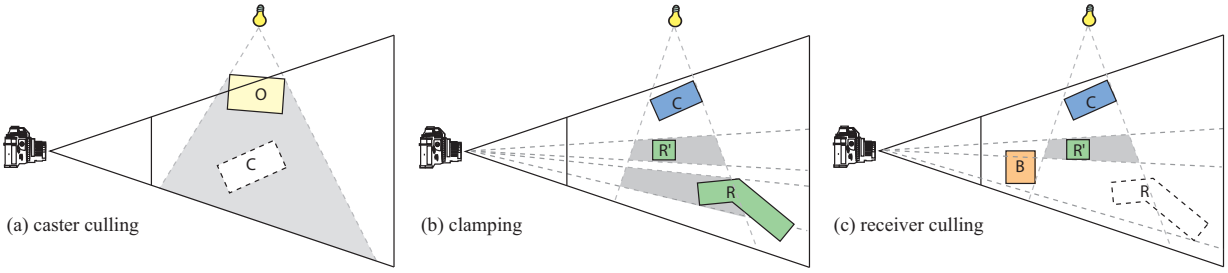


Fig. 8.8 : Shadow Volume Culling and Clamping

*The principle of shadow volume culling and clamping (a) shadow caster **C** is fully in the shadow of **O** so its shadow volume can be culled (b) the shadow volume for **C** need only to extend through regions containing shadow receivers (c) if a shadow receiver **R** is not visible from viewpoint, the shadow volume for **C** does not need to be rendered around it.*

CC shadow volumes is a technique introduced by [LWGM04] to reduce the fill rate incurred by rendering shadow quads that do not contribute to any shadow in the current view. There are three situations to consider, illustrated in Figure 8.8. Shadow casters that are fully shadowed can be culled, as any shadow they would cast will be created by what shadows them. For that, [LWGM04] test if a caster is visible from the light source by testing it against a shadow map using occlusion queries. For non-culled casters, the shadow volumes need to be rendered only around receivers that are visible from the light source (including the caster itself). To find those potential receivers, [LWGM04] simply test them from the depth buffer of the observer's view using occlusion queries. To clamp the shadow volumes, the observer's view frustum is cut in n_l layers by planes containing the viewpoint and oriented according to the light direction. The reason for this is that the intersection of front- and back-facing shadow quads with a layer projects on exactly the same trapezoid in the observer's view, so there is no need to project shadow caps on the layer's delimiting planes. For a given layer, [LWGM04] render the potential receivers with the two delimiting planes as clipping planes. The projection (from the light source) of each caster on the furthest delimiting plane is then rendered with a depth test. If no fragment passes the depth test, the shadow volume can be clamped for consecutive layers. The test is performed using occlusion queries; the receivers are thus rendered n_l times. Note that each of these renderings is not very costly because the clipping planes discard many primitives in the transform stage, but the geometry needs to be sent n_l times. The casters are also rendered n_l times, leading to a total cost of $n_l(n_r + n_c)$ where n_c and n_r are the number of non-culled casters and receivers.

Another method for performing CC shadow volumes is presented in [Déc05], but it cannot do multiple clamping and is not yet fast enough to compete. Slicemaps contain all the information to perform CC shadow volumes in a more efficient way than both previous methods. In this section, we first describe an ideal algorithm that is based on a proposed hardware extension that we believe is simple to implement on a chipset. We then describe a less efficient implementation that works on current hardware and performs better than [LWGM04]. We also show two ways of further clamping shadow volumes.

3.1 Ideal Algorithm

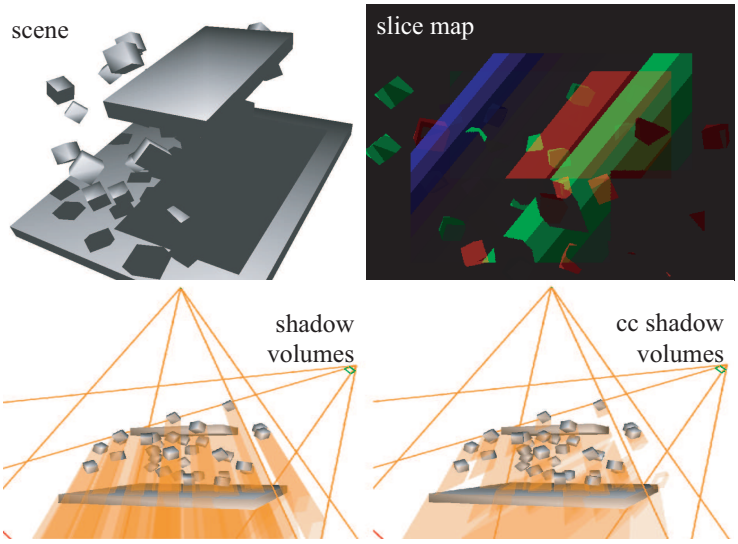


Fig. 8.9 : *Shadow Volume Culling and Clamping*

As in CC shadow volumes [LWGM04], we first cull casters and receivers by testing them against a shadow map and the observer's depth map respectively. We then compute a uniform slicemap with the potential receivers. The computation is a bit different than in Section 3 because the slices must follow the layers instead of being perpendicular to the light direction. To find the cell containing a fragment, we no longer use the cell-mask texture; instead, we project the corresponding 3D point into the observer's view and compute its distance to the projection of the light

along the projection of the light's direction. This distance can be computed at vertices and correctly interpolated for fragments.

To find where a caster's shadow volume should be clamped, the bitmasks of all texels of the slicemap on which the caster projects need simply to be OR-ed. This could be done by extending the occlusion query's mechanism so that for every fragment rasterized, the content of the framebuffer is OR-ed with the value of a special register (initialized to 0). OR operations are order independent and can be done concurrently, so it will not offend the SIMD nature of GPU. The API would then return the value of the register similarly occlusion queries, that return the number of pixels that passed the z-test. The demand for this kind of reduction register to treat order-independent activities has existed for quite some time, but in our case the structure would even be simpler, as the value would not even need to be locked while the modification occurs. With such an extension, clamping a caster would only require a single rendering of the caster, no matter the number of slices. The cost for the slicemap is independent of the number of slices, resulting in a total of $n_r + n_c$.

3.2 Practical Algorithm

Without the proposed hardware extension, we can still use a slicemap for clamping. A first solution would be to perform the OR by hand using a matrix-reduction method similar to [BP04]. This is a bit tricky to set up and might be too slow. Instead, we propose to use occlusion queries such as [LWGM04]. We run through each slice and use a fragment program to test if the covered texels have the bit for the current slice set to 1. If not, we discard the fragment. We use an occlusion query to discover if at least one fragment is not discarded and decide whether to clamp or not. The cost of our approach is n_r for the slicemap generation and $n_l n_c$ for the clamping. So the total cost is $n_r + n_l n_c$, which is less than for CC shadow volumes.

3.3 Improved Culling and Clamping

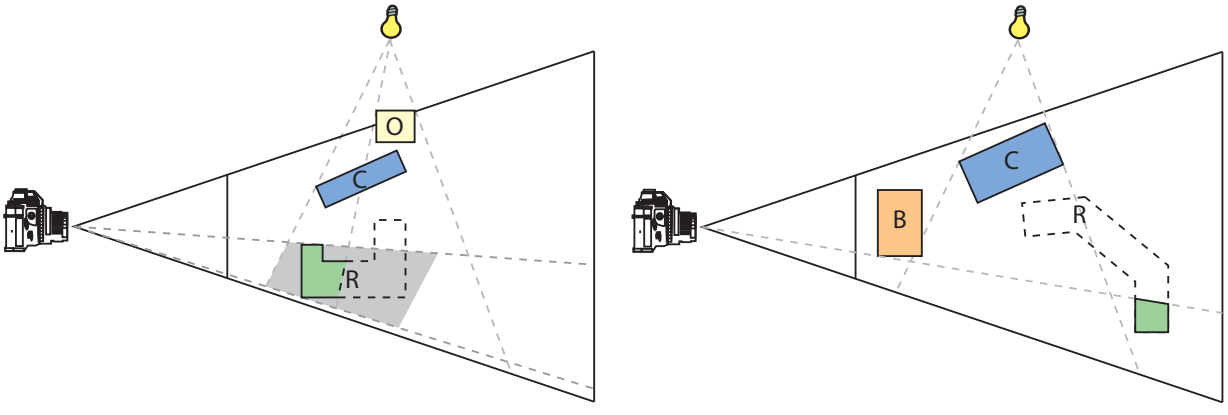


Fig. 8.10 : Improved culling and clamping

The fast extraction of the scene configuration allows pixel-wise instead of object-wise precision

The algorithm we described so far does not perform optimal culling and clamping. Figure 8.10 shows two cases that are not handled. On the left, the algorithm would cull the shadow volume of C around the whole receiver R. But since a part of R (the one dashed) is actually shadowed by another caster O, the shadow volume can be clamped tighter. To account for this situation, when testing a caster for a slice, we compare the depth of each fragment with the depth in the shadow map. If it is strictly greater, we can safely discard the fragment and ignore its bitmask. In other words, we do pixel-based caster culling as opposed to object-based caster culling. The rightmost example shows a receiver that is visible by the observer but cannot actually receive shadows. To account for this, we use the litmap approach described in [Déc05]. The litmap produces a depth map from the light that does only contain visible fragments from the current viewpoint. When generating the slicemap, we discard any fragment that is not visible from the observer. Thus, this fragment does not generate any one bit in the slicemap. Once again, this amounts to pixel-based instead of object-based receiver culling. Note that these two improvements would work straightforwardly with [LWGM04], although they are not described there.

3.4 Results

We implemented the algorithm to test its feasibility. Figure 8.9 shows the results obtained on a simple scene. A validation on more complex scenes and exact measured comparisons could be of interest, but our main intention was to show the applicability and possible gain of our method (especially when the proposed hardware extension becomes available) with respect to previous approaches.

4 CSG and Inter-Object Intersection

To test object penetration, many works rely on complicated data structures or numerous occlusion queries. With solid voxelization, general CSG is straightforward. Each object is voxelized in a separate slicemap. We then render the first over the second and blend both with the desired Boolean operation. Figure 8.11 shows an example.

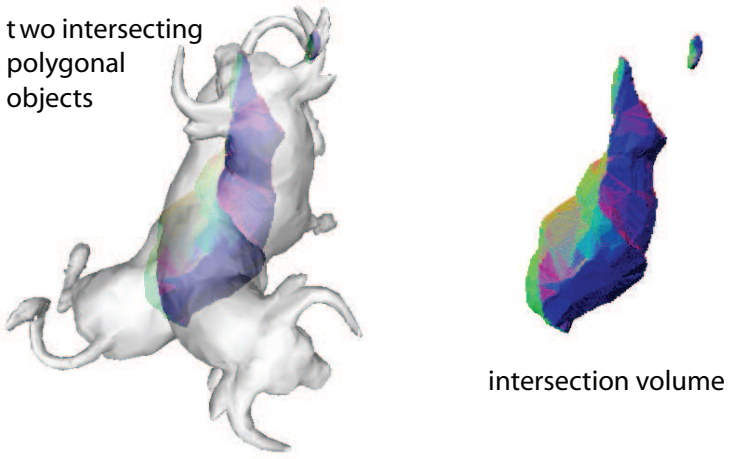


Fig. 8.11 : CSG Operation - *Discretized intersection of two complex meshes. The cost is directly linked to the voxelization.*

We want to emphasize that this is simple, but works at a precision of several *billion* (10^9) voxels in real time with low dependence on the input geometry's complexity. Furthermore, the density computation we presented in Section 4 allows us to rapidly recover the intersection's volume. This is useful for collision detection or haptic feedback. An extension to CSG trees is possible by storing intermediate results (intuitively $\log n$ where n is the height of the tree). Rearranging the tree could optimize this number [HR05].

5 Particle Collision

We apply our technique to a GPU particle system similar to [Lat04], although we have not optimized the particle rendering and simulation. We detect collisions and make particles bounce based on the solid voxelization with normals. Figure 8.12 shows some examples.

Fig. 8.12 : Particle Collision Simulation

Two examples we used to show the benefit of using our voxelization in the context of particle simulations. The stone in the left example contains several tunnels and holes. The right example shows the high resolution of the voxelization as the fine geometry of the toboggan is captured.

Our approach treats concave regions seamlessly, such as the complex toboggan scene (Figure 8.12, right). Dynamic deformation is possible since we recompute density and normals at every frame. The whole scene is queried via a single representation and the computation is efficient: normals are evaluated only when a particle enters into contact with a surface. Surprisingly, the actual bottleneck is the particle display via billboards. The simulation runs entirely on the GPU. The precision is high and even particles crossing a boundary (due to high velocity or large time steps) can be detected since we represent a solid volume. In these situations, we perform back-integration and estimate the actual impact point. Of course, theoretically, particles can still cross very thin volumes.

This being said, it shows notable advantages compared to previous work. Voxelizations from depth peeling might not have a sufficient resolution or holes in certain directions and clipping plane approaches have only restricted information along the z-axis. Consequently, these simulations rely on particles having some privileged direction, whereas our voxelization is uniform in the sense that all directions share the same quality. None can easily perform consistent back integration. They do have the possibility of capturing object motion directly. On the other hand,

our solution can be combined with a movement extraction step such as in [Dro07]. Motion can typically be sampled at very coarse levels. Rigid motion is a constant and can be passed directly into the shader.

Table 8.2 shows the timings for our DX10 particle demo including the collision response and normal derivations.

Table 8.2 Particle system collision response timings. It is voxelgrid independent (here 512^3) because computations are local per particle.

Nb. particles	256^2	512^2	1024^2
Collision management	0.32 ms	1.0 ms	4.0 ms

6 Mathematical Morphology

Finding an eroded interior is useful for many applications such as path finding or visibility (e.g., [SDDS00, DDS03]). Dilation creates a hierarchical structure, which allows rapid queries on whether a neighborhood contains filled voxels. Even for data smoothing or simplification, these representations are of interest. Usually, these representations are obtained in a lengthy preprocess [SDDS00]. In a binary context, erosion/dilation are simple logical operations (AND/OR). They are separable, so one can first erode/dilate along the x, y, then along the z direction. Finally, larger sizes are obtained by iteration. This implies that arbitrary rectangular kernels are possible. Bit shifting (such as in Section 4) allows us to treat columns efficiently. Care must be taken, as some bits are needed from adjacent integers. Figure 8.13 shows an example.

Fig. 8.13 : Mathematical morphology

Erosion (left) and Dilation (right) can be obtained directly from the solid voxelization, which represents the only step involving the actual scene geometry making the solution fast because it is purely image-based.

Summary:

In this chapter we showed a large variety of applications. As mentioned before, this might just be the tip of the iceberg. Algorithms for advanced collision detection could benefit from this representation. Particle approaches like [BYM05] or ray-tracing algorithms (e.g., for refraction and photon mapping) could make use of our hierarchical representation (Mathematical Morphology, Section 6).

Many of the presented algorithms could already be of interest to the community. Our *transmittance shadow maps* allow emulation of simple deep-shadow maps, even on older hardware, with acceptable precision and frame rates. Several effects can be achieved, based on the information of the object's volume. Our approach combines the advantage of exact shape information in the form of depth maps and approximated volume in the form of voxels and results in a good overall estimation. We have shown that this information could e.g., be used to create visual effects, shadow volume culling and clamping, CSG operations, CT Visualization, particle collision detection, and mathematical morphology.

Part IV

Visibility-Related Queries

"The formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill."

Albert Einstein

Chapter 9

Querification

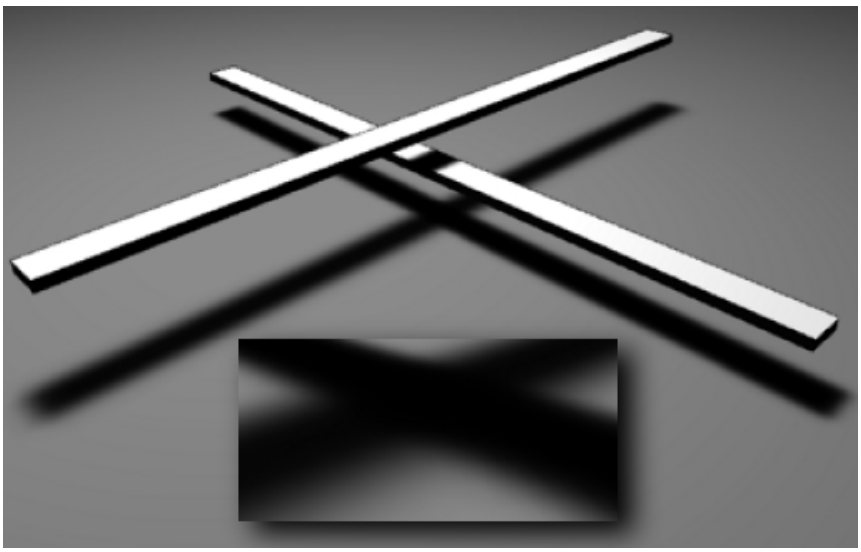


Fig. 9.1 : *Simple configurations can give complex (quadric) shadow borders (image: [ED06c])*

In this part of the dissertation, we will investigate how to reduce the complexity of a task by reformulating the initial problem to an equivalent or approximate solution. In both cases, this can have a substantial impact on the performance. This is something very particular of research in computer science.

In math, it is usually less important how a proof is done as long as it is correct. Of course, short proofs are considered more beautiful, but it does not change the importance of the initial result[1]. For many questions in computer science, efficiency plays an important role and the way something is solved can be more important than the solution itself.

The problem we investigate in this part is visibility, which was already the motivation for some of our other contributions. We will examine it in general and in the context of shadows in particular. In fact, the two are directly related as we will show in Chapter 10. We added this chapter on previous work because all our contributions in this part of the dissertation also find application in the context of shadows.

The results go in two very opposing directions: approximate soft shadows, and accurately sampled visibility. Similarly to our point of view on Preprocessing and Transformation, two possibilities exist. Either compute approximate shadows, but then aim at the highest performance possible, or use a little more resources, but deliver high accuracy.

[1] **{TODO: In some cases it might also be a question of honor... people still try to provide a proof to the fundamental theorem of algebra without resorting to analysis.}**

The approximate solution benefits from a formulation of visibility as a filtering process. This allows an implementation that is highly adapted to graphics hardware. We further show how the filtering process can be achieved in various ways resulting again in a trade-off between efficiency and accuracy. Finally, we reformulate occluder fusion, that is the combination of several blockers, as a probability problem.

One major reason for this approximate method to be efficient, comes from the fact that geometry is only involved during a simple rendering step, not during the computation of the shadows and the approximate, yet often convincing, heuristics. In consequence, the dependence with respect to the geometric complexity in the scene decreases substantially. In comparison to other approaches, we will see that the number of triangles has very little influence on performance. A side product of working on an image-based representation is that we can apply the algorithm to any rasterizable representations that produces a depth.

Starting with Chapter 12 the goal is to *sample* visibility between two regions. Sample points S_i and R_j are spread respectively on both. The query one needs to answer is:

$$\forall_i \forall_j \forall_k [S_i, R_j] \cap T_k \neq \emptyset,$$

where T_k denotes a triangle of the scene. The $\forall$-operators commute and, depending on the order, different algorithmic choices result. Due to the symmetry of the two regions, there are basically three different combinations. If we start with two point S_i and R_j, we obtain a segment that is then tested against the scene, which is typically the case for ray-tracing. If we start with only one point S_i and then take all the triangles, results in a case similar to shadow/occlusion mapping, against which we then test the points R_j. Finally, there is a third formulation [LA05], which is to start with a triangle and go over the entire set of sample combinations. What we realized is that this last formulation has one major advantage, it is well adapted to the GPU because the geometry only needs to be traversed once and each triangle is treated separately. This illustrates well what we mentioned already in the introduction, which is that one problem can be solved in very different ways leading to very different computation times.

We encounter the same principle in our last contribution in Chapter 13. Here, the main idea is to see a shadow map no longer as a collection of depth samples, but rather as a collection of pixels seen from the observer, that need to be tested for occlusion [AL04]. On a GPU, such an approach is challenging because the data structure cannot be directly created. We found a way to use a fixed size array in order to represent dynamic lists, which allowed the implementation. We further addressed other issues and finally propose an accurate soft shadow algorithm in real-time with 256-1024 treated light samples in general scenes.

Once again, we see that approximations, if well chosen, can lead us closer to where we want to get. Outside of an offline process, exact computations are not always needed and sampling helps boiling down the (originally impressive) complexity to a linear process. Although the number of samples is limited, for practical applications, the result is often sufficient.

"Beware lest you lose the substance by grasping at the shadow."

Aesop-The Dog and the Shadow

Chapter 10

Shadows

Fig. 10.1 : *The XYZ-dragon under a large light source*

One focus of our work is on shadows. We will present several techniques that allow real-time, soft-shadow computation in complex scenes. Often, each method presented has advantages and inconveniences. This is a good reason to take a close look at the actual problem definition and previous work in this area.

In particular, the last state-of-the-art report [HLHS03] dates from five years ago and focused solely on soft shadows. Much has changed since then. Soft shadows really emerged. The article surely awakened many people (at EG 2007 no less than three techniques were presented revolving around this topic). Hard shadows also received attention since the state-of-the-art report by Woo et al. [WPF90] which dates back 18 (!) years.

To understand our contributions, this chapter is not needed and could be skipped. We encourage the reader to at least take a look at the introduction to understand the usual shortcomings for shadow computations and the reasons why this particular topic was motivating for us.

In the context of the thesis, shadows represent a situation where, basically, at each point, the entire scene has a potential to cast a shadow. This makes it very challenging to create a convincing image. Pixel-accurate hard shadows, and especially soft shadows, are examples of the difficult problems we addressed with the solutions in the following chapters.

A very important point is that shadows, before all, have to appear plausible to an observer. It is actually possible to get away with stronger approximations, as long as the essence of a shadow is captured.

In this chapter, we will discuss shadows and common approximations. We will start by defining what we understand by *shadow* and explain why shadows are a substantial element in almost all computer generated-images. We will then give an overview of previous work, starting with approaches that compute shadows in the presence of a point light (Section 5). We then address

soft-edged shadows to anti-alias the result (Section 6) before continuing with area- and volume light sources (Section 7).

Our contributions range across a wide spectrum; from *very approximate but really efficient* to *definitely slower (but still efficient) and very accurate*. Some of our contributions are focusing more generally on visibility rather than solely on shadows, and we show how this information can be of interest in applications other than realistic rendering.

1 What Is a Shadow?

This is a good question and surprisingly even dictionaries have trouble defining it exactly. *Webster's* states: *Shade within clear boundaries* or *An unilluminated area*. By looking at Figure 10.2, one realizes rapidly that this definition is not accurate enough. But then, what is a shadow? A better definition is given in the *free dictionary* [The], which states: *An area that is not or is only partially irradiated or illuminated because of the interception of radiation by an opaque object between the area and the source of radiation*. This definition brings us closer, and coincides more readily with the definition in [HLHS03], *Shadow [...] is the region of space for which at least one point of the light source is occluded*. There are two catches though. First, this only considers direct light, light bouncing of a surface is ignored. Second, occluders are considered opaque, which is not necessarily the case in the real world[1]. In general, this problem remains challenging and we believe that it will continue to occupy future generations.

Fig. 10.2 : *A very large source leads to soft shadows. All points on the floor are actually lit*

But even when restricting ourselves to opaque objects, the definition for the "real world" is not as simple as the above descriptions lead us to believe. Shadows are different from shading, meaning different from light attenuated by the surface's reflectance function. Take a look at Figure 10.3 (left): do we see shadows in this picture? Without knowing what is depicted, most people would say yes. But actually, this picture shows a zoom on a leaf just like the one in Figure 10.3 (right). If one presents only the right, most people will tend to argue that this picture is without shadow.

In the same way, as we pointed out in Chapter 1, what we see and how we interpret what we see depends highly on the scale at which we look at things. There is a fine line between shading and shadows [HDKS00].

In our artificial world, details are usually omitted, but unfortunately, their impact on appearance can be enormous. A CD is a typical example of this: if you look at its back you see a rainbow of colors due to the fine surface structure that is used to store data. In practice (in our virtual reality), we cannot work at the scales necessary to capture these effects, and as a consequence approximations are necessary. Many approaches modify the appearance of a surface using techniques that simulate detail that otherwise would be lost due to the coarse representations of our models. Even more, virtual objects are often boundary representations (at least in the case of triangles). Real-world objects are much more complex and many effects take place underneath the surface; light is scattered, attenuated, or diffracted. To overcome this limitation, a great deal of research focused on simulating these events approximately on the surface. Examples include texturing, bump mapping, normal mapping, general BRDF (see next section), BTDF, and BSSRDF functions. Even more advanced shading models such as Cook-Torrance include terms

[1] We have already seen a solution for a case where we compute transmittance effects while disregarding scattering in Chapter 8

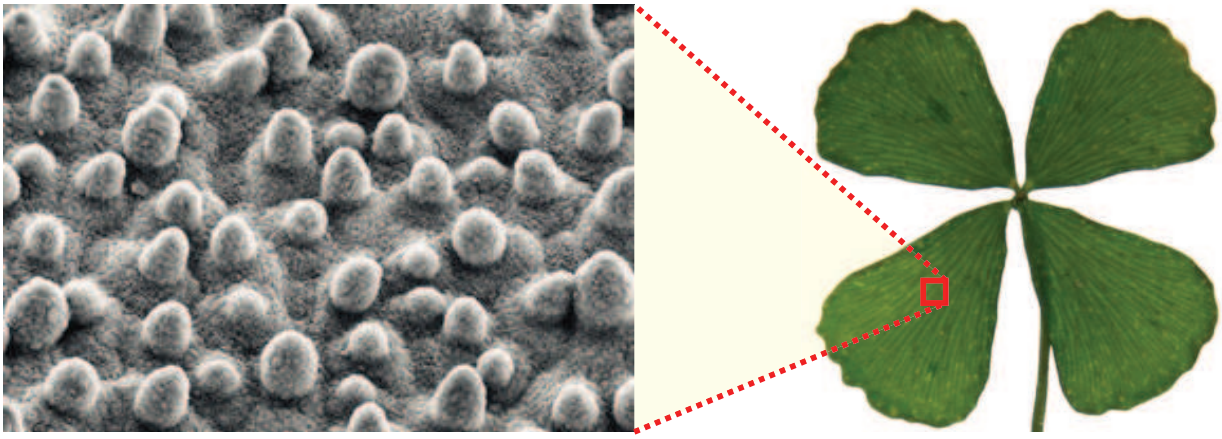


Fig. 10.3 : Ambiguity of the shadow definition

What we define as a shadow depends upon the scale at which we look at objects. In the real world, the definition is thus very ambiguous; in a virtual world, described by a mathematically accurate framework a precise definition is possible and meaningful.

that take the visibility and orientation of small scale details into account. This leaves us with an interesting situation. In the real world, shadows might have all kinds of ambiguities. In our artificial universe, a definition such as the one given in [HLHS03] is mostly sufficient; at least as long as opaque objects and direct light are concerned, which we will assume for the remainder unless otherwise stated. In this mathematical construct, details are limited, and shadows are described independently of scale and purely in terms of visibility.

Notations: A point P lies in shadow if and only if there exists an open segment from P to a source point[2] on the light L such that it intersects the scene $\mathcal{S}$. This assumes that light travels along straight lines (even though in some situations this is not a valid approximation, e.g., atmospheric scattering, black holes). Formally, P lies in shadow if and only if:

$$P_{blocked} = \{Q \in L | (P,Q) \cap \mathcal{S} \neq \emptyset\} \neq \emptyset$$

Let P lie in shadow, then: if $P_{blocked} = L$ then P lies in the *umbra*, or otherwise in the *penumbra*. An object that can intersects segments from P to the light is called an *occluder* (also called *blocker* or *shadow caster*) for P. Generally, we refer to an object as an occluder if there exists a point P such that it is a blocker for P. A receiver is the element upon which the shadow is cast; in the above example it contains the point P. There are situations where receivers and blockers are distinct, or where each receiver is only shadowed by a subset of casters. Notably, some algorithms do not allow self-shadowing (caster and receiver being the same object).

[2]Artificial constructs can have *source points* at infinity, e.g., environment maps.

1.1 The Rendering Equation and Approximations for Soft Shadows

Further Reading: *A more detailed and accurate derivation of the radiosity equation can be found in [SP94]. It also contains an exhaustive presentation of the entities and units of all function components. This section aims is to provide very high level insights into what we compute and where it comes from originally.*

So far, we have clarified where we can find shadows. Now, we will discuss their actual influence on the appearance of a scene. We will make use of one of the fundamental equations in computer graphics, the so-called "rendering equation" introduced by Kajiya [Kaj86][3].

$$L_o(P,\omega) \;=\; L_e(P,\omega) \;+\int_{\Omega} \rho(P,\omega,\hat{\omega})\; L_i(P,\hat{\omega}) \;<\hat{\omega}|\vec{n}_P> d\hat{\omega}, \tag{10.1}$$

where P is a point, ω a direction, $\vec{n}_P$ the normal at point P, and Ω the hemisphere above the surface at P.

- L_o is a function of position P and direction ω and describes the outgoing radiance. Simply put, the light (direct or indirect) leaving a point in a given direction.
- L_e is the emission of radiance. Simply put, the energy produced at a given point for a given direction.
- ρ is the bi-directional reflectance function. It is a function representing the ratio of incoming to outgoing light for a given point and ingoing as well as outgoing directions. This function can be very complex but here we will give only the major intuitions needed for our purposes.
- L_i is the incoming radiance. We will see right hereafter that it relates to L_o.

This equation is physically based and describes the equilibrium of energy in a scene. It is a very good model of the illumination exchanges, but solving the equation is analytically difficult (except for a few uninteresting cases). Its inherent dependency upon itself ($L_i(Q,P-Q) = L_o(P,Q-P)$ if P, Q are mutually visible) is one of the hurdles. Photo-realistic rendering aims at efficient ways to approximate and populate this equation. The integration over the directions corresponds to an integration over a sphere centered at P on which all the surrounding geometry is projected as seen from P. Performing a change in variables to integrate over the surfaces of the scene instead of the directions leads to:

$$L_o(P,\omega) = \; L_e(P,\omega) \;+\int_{S} \rho(P,\omega,P-Q) L_i(P,\omega)\; v(P,Q) \frac{cos(P-Q,n_Q)\; cos(Q-P,n_P)}{\pi||P-Q||^2} dQ,$$

where the newly added binary function v encodes the visibility. It is one if P and Q are mutually visible, if not it is zero.

One simplification is to assume that all surfaces in the scene are perfectly diffuse. This is typically done for radiosity [SP94], a technique to compute global illumination in a scene. *Perfectly diffuse* means that radiance is reflected uniformly in all directions. As a consequence, the bi-directional reflectance function becomes a piecewise constant function ρ only depending on position. Further, the angular dependence is removed and leads to a uniformly emitted energy per surface area, the

[3] Kajiya introduced the equation in a different formulation, but for our explanation this equivalent form is more appropriate.

radiosity, $B(P) = \int_\Omega L(P,\omega)\ cos(\omega,N_P)\ d\omega$, replacing L_o, L_i. Equivalently, the uniform E replaces L_e. As a result, we obtain the so-called *radiosity equation*[4]:

$$B(P) = E(P) + \rho(P) \int_S B(Q)\ v(P,Q)\ \frac{cos(P-Q,n_Q)\ cos(Q-P,n_P)}{\pi||P-Q||^2}\ dQ \tag{10.2}$$

This equation is a good trade-off between approximation through modelization and accuracy, but for soft shadows, we are only interested in direct illumination. This removes the equation's dependency on itself. Consequently, for all points in the scene, the integral evaluates to zero except for those locations lying on a source. It follows that the term E can simply be omitted and added back later. Also, the additivity of the integral allows us to treat several lights sequentially and add their contributions. We thus assume that there is only one source in the scene, and that it has homogeneous directional radiation over its surface. The latter reduces L_i to a simple function of position $L_c(Q)$, which in case of a uniformly colored source reduces to a constant and can be removed from the integral (this case is very common and we will explicitly mention if $L_c(Q)$ is not a constant and can remain in the integral). Let us define:

$$geom(P,Q) := \frac{cos(Q-P,n_P)\ cos(P-Q,n_Q)}{\pi||P-Q||^2}$$

and we obtain the *soft shadow equation*:

$$B(P) = \rho(P) \int_L L_c(Q)\ v(P,Q)\ geom(P,Q)\ dQ \tag{10.3}$$

Radiosity *Radiosity methods discretize the Equation 10.2. Using the Galerkin method, the solution is assumed to be representable as a linear combination of basis functions.The idea is to then search for the coefficients B_k that lead to the best approximation. For this, the equation is projected onto the basis functions. In the special case, the functions are piecewise constant and associated to surface patches. One can derive a simpler recursive equation:*

$$B_i = E_i + \rho_i\, B_j \int_S v(P,Q) \frac{cos(P-Q,n_Q)\ cos(Q-P,n_P)}{\pi||P-Q||^2}\ dQ$$

The integral is referred to as form-factor or coupling coefficient and is very costly to compute. Rewritten, the equation shows that its solution is given by a fix-point equation:

$$B = E + M\ B$$

With the von Neumann series, it follows $B = \sum_i^\infty M^i E$. This well-known result has been used in several approaches, e.g., very recently in [KTHS06]. It nicely reflects the intuition of light transfer. The solution is initialized with the emitted energy from the sources. The matrix M then reflects the energy once and gives the new energy repartition in the scene. The complete solution is thus the sum of light from the sources and its subsequent reflectance passes.

This Equation 10.3 is considered to be the value that one should compute to derive a physically based soft shadow [ARHM00]. In practice, a still close result can be obtained when simplifying the equation further. If the distance of the light to the receiver is relatively large with respect to the light's solid angle (the angle an object subtends in three-dimensional space for a given point), and the light's surface is well behaved, the geometry term varies little. This allows for a separation of the integral.

$$B(P) = \rho(P) \int_L geom(P,\hat{Q})\ dQ \int_L L_c(Q)\ v(P,Q)\ dQ$$

$\rho \int_L geom(P,Q)\ dQ$ can be considered the direct illumination or shading of a surface. This approximation depends on the correllation between the two functions and Soler delivers an error discussion in his dissertation [Sol98].

[4]Sometimes *radiosity equation* might be used to denote the transformation of this equation for a basis of surface patches. Then the coefficients correspond directly to the radiosity of the corresponding patch. More detail can be found in the sidebox on radiosity.

It is interesting to know that for $\int_L geom(P,Q)\,dQ$ analytic solutions exist even for the case where P is a polygonal region and we integrate over P [SH93]. This is typically the case for radiosity computations. On the other hand, the exact formula, although an important theoretical contribution that remained unsolved until 1993 (despite earlier attempts such as Lambert's in 1790) seems too complex for practical applications. Equation 10.2 sometimes allows analytical solutions as well. But only for particular BRDFs and visibility functions. This typically leaves only the sampling option (e.g., Monte Carlo).

The remaining *visibility integral* modulates the direct lighting and represents the true shadow component in the equation (assuming $L_c \equiv c$):

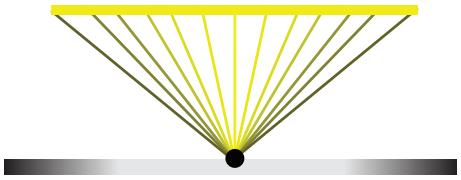
$$\int_L L_c(Q)\, v(P,Q)\, dQ = L_c \int_L v(P,Q)\, dQ \quad (10.4)$$

Usually, for real-time applications, Equation 10.4 is meant when talking about soft shadows. This often leads to misunderstandings when having conversations with people from offline global illumination. To be precise, all of our solutions aim at solving Equation 10.4. Nevertheless, we will see that two of our methods provide a sampled visibility function and not just the integrated quantity. This information allows us to remount to Equation 10.3 at a supplementary cost. Another important remark is that for point lights 10.4 simplifies to a simple binary visibility query.

It should be noted that Equation 10.4 is far from being physically correct and the approximation can be quite different from a reference solution based on Equation 10.2. Only the hidden amount of the source is evaluated, not where the blocker projects onto the source. Because of the term $geom(P,Q)$, the influence of the source on the point P is not uniform and falls off with distance and orientation. The irradiance thus depends on the hidden part of the source, which is not captured in the separated integration. Often people neglect to mention the conditions under which the above approximations are valid. Even [HLHS03] classifies [AAM03,SS98] as *physically* accurate for a convex occluder even though only the visibility integral is evaluated.

A Word on Direct Illumination *In many cases, the direct illumination integral is approximated by replacing L with a point light. This diffuse lighting model (Lambert) and Phong-Blinn, are the current standards on graphics cards. (For more details, We refer to the excellent survey by Schlick [Sch94] and for efficient solutions to map them to older graphics cards, we suggest [HS99]. Modern hardware often facilitates this task via shaders).*

$geom(P,Q)$ is obviously related to a squared distance (Light travels along straight lines and the energy is conserved. In consequence, the energy on the surface of a sphere around the light source should be constant.). Interestingly, OpenGL makes it possible to attenuate the light's power with distance using a general quadratic polynomial. This may sound strange, and makes many people smile when they hear about this option for the first time, but it actually makes some sense to include this supplementary degrees of freedom.



The situation is depicted in figure above. Far-away source points will have very little influence on the final result due to their orientation with respect to the receiving point. This is reflected in the $cos(Q-P,n_P)$ $cos(P-Q,n_Q)$ term of $geom(P,Q)$. In consequence - if the light is large with respect to the current receiving point - moving the light a little will have almost no impact on the received illumination. In other words, leaving the source at the same position and looking equivalently at adjacent receiver points, we will observe basically the same energy. From a certain distance on, orientation will be mostly constant, thus attenuation behaves in a quadratic manner as predicted. In between, the behavior passes through a linear stage. The more general polynomial that OpenGL offers, mimics some of this behavior.

2 Why do We Care About Shadows?

In general, people probably don't care much about shadows, except that we might want to avoid them to get tanned. But in graphics, shadows are of major importance: they provide clues concerning the spatial relationship of objects in the scene and the shape of a receiver, and reveal to some extent information hidden from the current point of view. Several experiments exist in

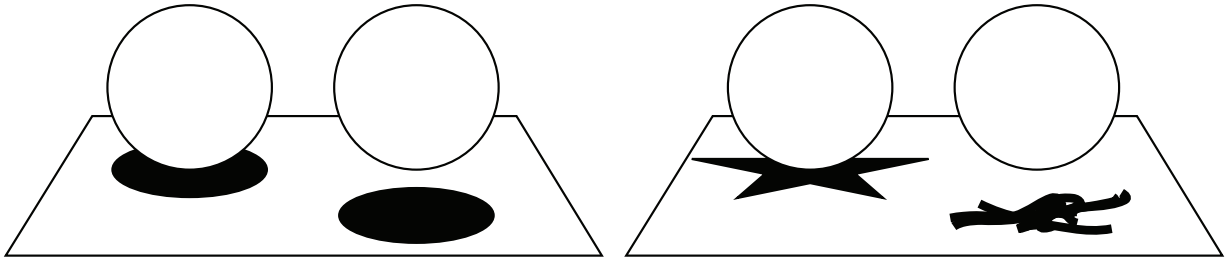


Fig. 10.4 : Position perception is altered by shadows

Shadows have an important influence on the interpretation of spatial relationships in a scene (left). Nevertheless, even coarse approximations can achieve the same effect. (right)

which the importance of shadows has been underlined. Even though this is not in the center of our interest, Kersten et al. [KKMB96] investigated the influence of shadows on perceived motion. In their many experiments, they also displayed a sphere above a plane, not unlike Figure 10.4 (left); just as you can see in this image, the trajectory and position of the shadow influence the perceived position. If the shadow moves up in the image, the observer will have the impression that the sphere moves to the back of the box towards the ground. This simple example is often cited to underline the importance of shadows, and it proves this point well. These experiments lead many to the conclusion that we should thus compute realistic shadows.

Fig. 10.5 : *We associate even deformed shadows to casters (Courtesy of Lucasfilm Ltd.)*

To my eyes, this latter consequence is wrong. My opinion stems from the fact that even the most approximate shadows can often provide sufficient information to interpret the spatial relationships. Take a look at Figure 10.4 (right). We understand the scene just as before, but the shadows are far from realistic. In fact, this motivates seeing shadow deformations as a means to convey messages (see Figure 10.5). These manipulations have now even found their way into NPR [DCFR07]. Other experiments [NBA04] showed that it actually suffices to have something dark underneath the object. An observer automatically makes the connection and accepts the *shadow*.

Interestingly, Kersten et al. [KKMB96] found that soft shadows can lead to an even stronger motion cue than hard shadows. But again, strong approximations suffice. Another interesting observation is that a moving light is very disturbing and we tend to interpret the shadow movement induced by light as an object's movement.

It is not necessary, then, to create an accurate shadow to explain a scene. But the question is how far we can go and this is very difficult to decide, even for artists. Cavanagh [Cav05] mentions several perceptual problems when ap-

Fig. 10.6 : Drawing shadows is not easy

Artists also have trouble with shadows. The left example looks very convincing, but closer examination reveals that the lighting is incoherent. The shadows of the people in the foreground indicate light coming from the left, but if that is the case then the shadows on the ceiling are impossible as the room is closed to the left. We realize that something is incoherent when shadows interact (middle) and, even though we seem to have some tolerance, too much approximation fails to convince and even confuses (right). Image excerpts: Fra Carnevale (left, 1467) (left), Signorrelly (middle, ≈ 1488), Moore (right, 1893).

proximating shadows and other physical phenomena. Figure 10.6 (left) underlines the fact that we are bad at estimating light directions and do not automatically realize problems of incoherent lighting. The ceiling's shadows are completely incoherent with the shadows created by the people in the scene. Conversely, the moment that inconsistencies are in close spatial relation, as in Figure 10.6 (middle), these shortcomings are evident. This underlines the problem of too-coarse approximations. Even though we obviously would like to benefit from the limited perceptual capacities of the human visual system, this is very difficult for dynamic scenes where an acceptable, approximated configuration might change into an obvious visual deficiency. As a direction for future research, however, this is a promising field. This also concerns the degree to which approximations can be applied to shadows. Figure 10.6 (right) shows such an example. For an observer, it is hard to identify the shapes at the feet as cast shadows.

Finally, some artists exploit the fact that we often make unconscious assumptions concerning the caster's shape based on a shadow. Surprisingly, this can incredibly fail, as shown in Figure 10.7.

Fig. 10.7 : *Dirty White Trash (with Gulls) 1998 (Courtesy of Shigeo Fukuda)*

So, do we need to care about accurate shadow computation after all? Not always. I remember the story of a student who had worked on soft shadows, and during his presentation, one member of the jury (not related to graphics) asked: "But is this a real problem? I played Rayman a long time ago and they had soft shadows!" One has to know that Rayman only shows a disc with a slight gradient underneath the player. Memory can play tricks on us and perception is relative. Shadows are use-

ful to increase realism. Physical computations are not needed to explain spatial relationships; other means are sufficient, like a game character such as PacMan is sufficient to describe the player's location and orientation. The same holds for shadows, if we simplify the scenario: light only from above, toon-style environments, etc. Coarse approximations suffice to indicate spatial relations, and simple solutions rarely result in any incoherence. In a realistic image, it is much harder to *fake* shadows. Lights and objects are dynamic and the realism of the scene would suffer dramatically from incorrect shadows. Further, many applications need sufficiently realistic computations.

In architectural design, light transport plays an important role and often even involves a complete global illumination of which soft shadows are just the first step. While an architect is often capable of imagining the final illumination in a building, a potential customer is not. Recent works on global dynamic relighting [KAMJ05, KTHS06, LZK*07, DSDD07] underline the importance of decoupling direct from indirect lighting. Indirect lighting can often be well compressed, or coarsely computed. Usually, energy in the scene is initialized with a direct light pass and then efficient approximate computation or a precomputed global transfer function is used (compare sidebox on radiosity on page 144). Direct light is constituted high energy resulting in high-frequency shadows. As consequence, approximations are more visible, and achieving a realistic composite necessitates an accurate first pass.

For movie productions, even direct lighting is costly because the sources need very accurate sampling. This is especially true for shadows from the light sources because the transferred energy is relatively high and fewer approximations are possible than for indirect lighting.

Finally, realism can be important if the observer is investigating a realistic environment. Nevertheless, in this scenario, we should take advantage of the fact that accurate shadows are not needed to convince of realism. But, we should be warned that incoherence can destroy the immersion in this virtual world. In some situations, accurate shadows might even be part of the gameplay, such as a player who casts a shadow around a corner, revealing his position. Furthermore, if the degree of realism is high enough, this might allow for the deduction of identity or equipment.

The keywords in this context are *plausible and convincing* shadows. We should provide *sufficient* realism, not necessarily exactitude. Unfortunately, it is not easy to achieve this goal. Ultimately, only Equation 10.3 seems to be foolproof, but Equation 10.4 is sufficient in a large number of cases. Any further approximation is likely to fail in some configurations. This is a major point: ultimately we should aim for approximate solutions, but in practice, only physically based shadows seem to be convincing in all situations. In the following, we will motivate our work in this context and illustrate the main failure cases that make soft shadows such a challenging topic.

3 Why Is It Difficult to Compute Shadows?

Figure 10.8 shows how drastically soft shadows influence the appearance of a scene. A single hard shadow results in an unrealistic image. Even though a large amount of light is impinging in the room, the fruit basket casts a shadow that is overly dark and large. In nature, we would never encounter a small object that could block the light of the entire window. This is a situation where even a novice realizes that something is not as it should be. This can be particularly disturbing in an animation because even small objects can block visibility of a point light, bathing the entire

scene in darkness. The soft shadow image, on the other hand, does not exhibit these artifacts. Contact shadows stay sharp and the scene receives a realistic amount of direct light.

Fig. 10.8 : Hard Shadows can lead to unrealistic images

This scene shows one example of the importance of soft shadows in obtaining a convincing and realistic-looking scene. On the left is the result using a single hard shadow sample, whereas the right shows the outcome of a soft-shadow computation.

One difficulty of soft shadows is that treating occluders separately is not simple. Even if for each occluder an accurate scalar value is derived that indicates the blocking contribution for a particular object, it is generally not possible to derive an good estimate of the visibility integral. It is true that these values can be used to derive upper and lower bounds for the exact visibility integral, but not more. Let O_{i0}^n be a set of objects and B_{i0}^n the visibility integral for a given point P. Then, the following inequality holds.

$$max(B_i) \leq \int_L v(P,Q)dQ \leq \sum_{i=0}^{n} B_i \tag{10.5}$$

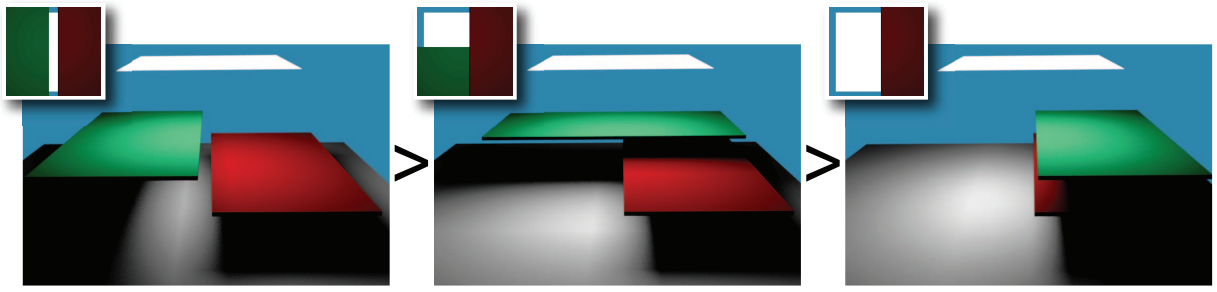


Fig. 10.9 : Occlusion of blockers cannot be easily combined

The figure shows different blocker-fusion scenarios for a view-sample in the center of the scene. On the left, both blockers occupy separate parts of the hemisphere their occlusions should be summed. In the middle, they partially overlap; here, it is a multiplication. The example on the right depicts one blocker being entirely hidden by the other. The maximum of both occlusions values would be the right choice. (Inlays show the source as seen from the center view-sample)

Figure 10.9 shows an illustration of different cases. The lower bound is produced if all blockers fall in the same part of the hemisphere; the upper bound if all their projections are disjoint. Many solutions have been proposed to combine different blockers, including the two extremities [AHT04, AAM03], as well as an average [SS98]. None of these approximations are valid in all situations. That problem, as for visibility, is referred to as inaccurate *occluder fusion*. Figure 10.10 shows an extreme case for a very simple game character that illustrates how shadows can become very unrealistic if blocking contributions are not combined properly.

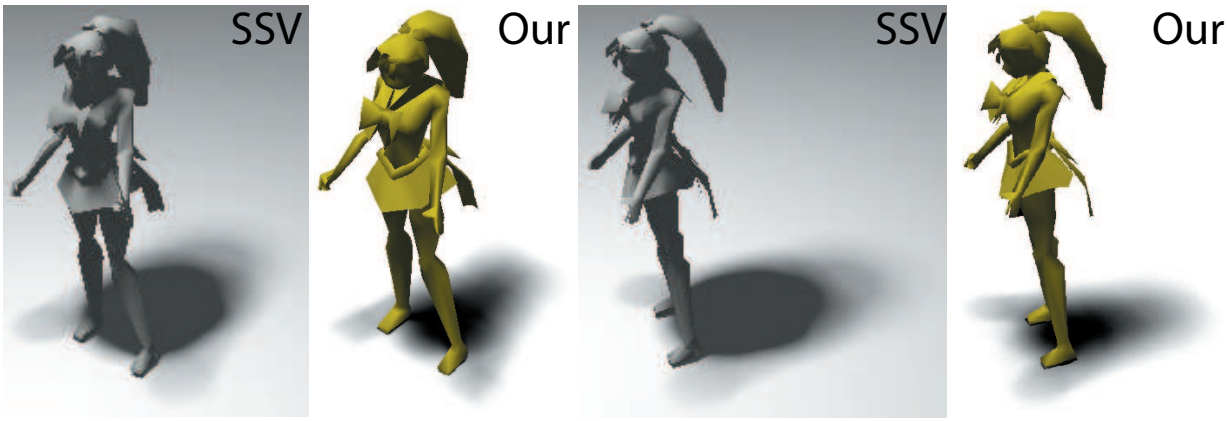


Fig. 10.10 : Inaccurate occluder fusion

Even for typical and relatively simple game characters, classical approximations (silhouette from the center, additive occlusion) can cause noticeable artifacts (here the result with [AAM03] is shown). The umbra is overestimated. In comparison, accurate visibility evaluation leads to convincing shadows (the reference was computed using our approach [ED07b]).

The intricate relationship between soft shadows and visibility introduces other implications. One cannot rely solely on objects visible from a single point on the source to compute shadows. This is illustrated in Figure 10.11. The right image is convincing, but on the left, only faces visible from the source's center intervene in the shadow computations (the computation for the visible geometry is carried out with highest accuracy). One can see significant problems. The shadow on the sphere is lost and especially half of the shadow below the sphere seems to be missing. In fact, this is the case because a single depth layer will not recover the hidden half of the sphere. Although this part of the scene is not visible from the source's center, it has an important impact on the shadow. In a simple scene like this already four layers interact. Further, the notion of a layer is ill-defined for faces aligning with the source's center. During extraction these would usually be missed or unadequately captured.

The same observation holds if one uses silhouette edges as seen from the center [AAM03, AAM02, ADMAM03, CD03, WH03]. Even when including those hidden from the light it can create artifacts and even lead to temporal incoherence.

On the one hand, it is surprising to see how much attention is needed when evaluating such a simple equation as 10.4. On the other hand, a robust solution is to sample the equation.

$$\int_L v(P,Q)\, dQ = \sum_{i=0}^{n} v(P,L_i), \tag{10.6}$$

where L_i are uniformly placed samples on the source.

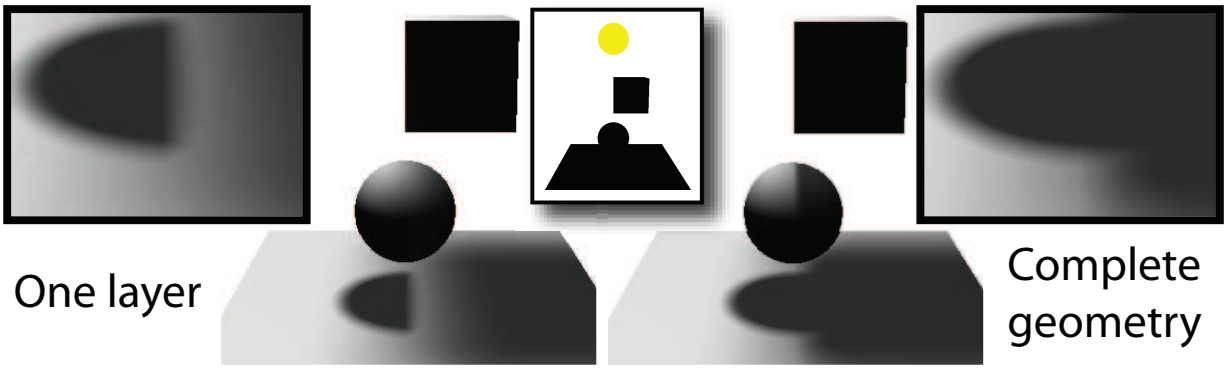


Fig. 10.11 : The first depth layer might not be enough for convincing shadows

Left: One depth layer and accurate shading. For accurate shadows, in this case, four layers would need to be extracted. Further, it has aligned the face with the light, which can be problematic when rasterized. Right: Accurate shadow solution computed with our approach in Chapter 13

We have already encountered this term $v(P,L_i)$. It is the visibility for a point light source at L_i. Soft shadows can thus be computed by covering the area/volumetric source with sample lights. Integration in this way corresponds to a one-by-one evaluation, where energy contributions are simply added. Even Equation 10.3 can be solved directly in this way. We thus have a direct link between hard and soft shadows. Shrinking an area light to a point leads to hard shadows and covering an area light with sample lights results in soft shadows. Unfortunately, as we will see in the next section, even shadows for a single light remain an issue, and for convincing soft shadows a high amount of samples is needed. 256-1024 are standard for medium-sized sources, but large area lights might necessitate even more. Each sample will need to process the geometry of the scene. For 1,000 samples, the cost of a brute force computation will thus be roughly 1,000 times higher than for a single point light (see sidebox). Different solutions are necessary and we will describe them briefly in the next section.

Why Not Accumulate Hard Shadows? *The **XBox 360** has a fill-rate of about 16 billion pixels per second and can process up to 500 million triangles per second. This sounds like a lot, but let's assume that the resolution of our view is* $512^2 = 262,144$ *pixels. If we assume that the evaluation of a shadow has approximately twice the cost of rendering this view, and multiply by a factor of* 256 *samples, we obtain* $134,217,728$ *pixels. If we want 60 fps the fill-rate goes up to* $8,053,063,680$*. This seems to work out, as it is approximately half the specification of the card, but we have not yet processed any geometry nor have we evaluated any of these maps. Regarding the specifications, it is worth noting that fill-rate and geometry-processing are measured independently and in artificial conditions (e.g., non-shaded triangle strips, no blending operations, etc.). In particular, seeing that the architecture has general stream processors (compare Chapter 2), the workload of the two processing stages is no longer independent. In practice, only a few frames per second are possible even on simpler models.*

4 Previous Work on Shadows

What this overview is <u>not</u> about:

This chapter will focus less on ray-tracing approaches and only mention a few (mostly, those that we considered interesting because of their usage of image-based representations or adapted computations), but the presentation is by far not exhaustive.

Solutions for particular cases such as terrains, cities, crowd animation, or trees are also ignored.

*We focused solely on local illumination, not environmental lighting. A large body of literature exists on the latter topic and it is a very interesting field. Even though ever new evolutions in the direction of completely dynamic scenes under these conditions exist [ADM*08], most often lighting precomputations and basis functions such as spherical harmonics (e.g., Precomputed Radiance Transfer) are used. One particular approach by Kautz et al. [KLA04] will be discussed briefly in Chapter 12. It shows similarities to our work [ED07b] and is therefore of particular importance.*

There are two good surveys on soft shadows [WPF90,HLHS03] that describe some techniques in more detail than we will here.

Shadows have a relatively long history in the young science of computer graphics. One of the earliest methods, shadow volumes [Cro77], was published in 1977. It nevertheless, took more than 20 years before it was finally applicable in the context of real-time rendering [BS03]. The algorithm in [BS03] was very fast for its time, based on many very particular representations in order to adapt the computation to a graphics card. Today, this solution is mostly historical. More direct and efficient implementations are possible on the latest generations of cards. But even today, the best solutions [SWK07], which exploit a precomputed hierarchy on a static scene of 500,000 triangles, only achieve interactive rates.

Isn't Ray Tracing the Answer to All? *Ray tracing is currently emerging, but soft shadows will pose a significant thread in this context as well. According to reports by Intel regarding ray tracing, a single P4 3.2Ghz is capable of 100 million rays/sec on average models, but this mostly in unrealistic scenarios and necessarily static scenes. Even the latest eight-core systems usually achieve under 83 million rays per second on average-sized scenes (in demos, as of the end of 2007, shown by Intel themselves [Shr]).* 512^2 *resolution times 60 fps leaves only 5 secondary rays per pixel. Even if one assumes that a more powerful solution with 450 million rays per second existed, only 28 shadow rays would be possible. Furthermore, the timings are usually measured without shading/texturing, which has a significant speed impact [WIK*06]. In the long run, ray tracing could be a solution. Nevertheless, I personally doubt that we will have the needed processing power to compute enough secondary rays for sufficient quality any time soon and primary rays will have problems to beat rasterization.*

In this section, we will concentrate on some important publications on shadow rendering. Over the last years, many contributions have been made in the field of shadows. They are now in almost any newer computer game. But at the time of this document, even though we have come a long way, accurate soft shadows cannot be

obtained in sufficient quality in real time, and we are only on the verge of achieving pixel-accurate hard shadows for geometrically complex scenes at acceptable speed[5].

Both areas still have room for improvement, especially as a scene with a single point light is not a very common scenario. In modern movie productions, hundreds of point lights are placed to create a wanted illumination.

In the following we will analyze different kinds of shadow algorithms. Starting with hard shadow methods in image and object space in the next section, we will then discuss anti-aliased shadows in Section 6. The latter differ from soft shadows in that they produce coarse approximations of the visibility integral, if at all. The goal of these methods is to provide a smooth shadow boundary, which is visually more pleasing and often leads to a *penumbra-like* effect. We make the distinction to *real* soft shadow algorithms based on the capacity whether a small object lifted towards a large light source will have a shadow that dissolves into a penumbra. If this is not the case and an umbra region remains, we classify the technique as *anti-aliased*. The last part of this overview will deal with real soft shadow algorithms. In this Section 7, we will again make the distinction between geometry- and image-based solutions. A special case of soft shadows is ambient occlusion, discussed in Section 7.1. In this case, visibility is seen as accessibility. This is equivalent to the assumption that light is coming from all directions.

5 Hard shadows

For a first time, we will discuss shadows from point lights: hard shadows. In ray tracing, shadows can be computed by shooting a second ray from the point of impact to the light source. This test indicates whether a point is hidden from the light and thus lying in shadow. It does not sound very problematic, but it can more than double the rendering cost because the secondary rays are less coherent.

Currently, rasterization, implemented on graphics cards, is available to the majority of users. Unfortunately, even though very powerful,

Ray tracing/Rasterization: *GPUs now allow to execute ray tracing on standard hardware. Currently performance remains below the results obtained with an optimized ray tracer. In the future, this might change, but my guess is that really efficient ray tracing will only become possible with adapted hardware (maybe Intel's Larabee), not standard graphics cards.*

The spirit of both approaches is very different: rasterization uses local coherence to accelerate computations, whereas ray tracing is highly parallelizable due to the independence of the rays. It is interesting to see that currently both directions merge but do not seem to meet at the same spot. Ray tracing has led to frustum tracing, where coherence of ray bundles are exploited. This almost resembles a small rasterization window that is shot in the scene. On the other hand, GPU implementations start creating complex data structures that are queried by rays - artificially grouped, sorted, and aligned in a texture to pretend a rasterization operation. Local ray tracing is increasingly interesting: rays are shot against texture-encoded, geometry-like height fields or volumes and ray travel is restricted.

No one knows what the future holds in store, but I believe that the two solutions will approach more closely. Rasterization will remain adapted for primary rays, but local computations and ray tracing against alternative representations will be interesting for secondary rays. A coexistence of both, like vertex- and fragment shaders, could be possible.

[5]Id Software is an interesting exception. The shadow casters were often much simpler models and the scene is frequently excluded as a caster. In this configuration stencil shadow volumes are applicable at higher speed. It results in pixel-correct shadows for relatively simple characters.

secondary rays are a difficult task for these architectures and adapted solutions are needed. In the following, we will give an overview of techniques related to shadow computation. Mostly, we will focus on solutions related to graphics cards as they are part of the context in which this dissertation was written.

5.1 Image-Based Hard Shadow Approaches

Point light sources create no penumbrae because of the direct relationship between hard shadows and point-point visibility. A very simple solution to derive shadows in a scene with a set of receivers, a point light, and a set of occluders between the two is to create a binary image of the occluders from the light. One can then test whether a receiver point is falling in an occluded pixel or not.

This observation inspired Williams [Wil78] to extend the map and store, for each pixel, the nearest depth encountered on a straight ray from the light. This *shadow map* is created in a first rasterization step from the light. A second step then applies the shadows from the viewpoint. Each rasterized pixel from the viewpoint, which we will call *view-sample*, is projected into the light's view. There, its resulting depth is compared to the stored value. If it is farther away than the stored reference, it is considered in shadow else lit.

The technique is particularly interesting as it is usable with almost arbitrary input, as long as depth values can be produced. Further, the fact that both steps involve standard rasterization gives it a huge potential for acceleration on graphics cards. In fact, OpenGL does provide extensions to perform the algorithm without shader intervention (today, most people would just use shaders, which is more convenient). Currently, shadow mapping and variants are the most popular technique for creating shadows in games. Nevertheless, several problems were introduced with this method.

The fact that a frustum is needed to rasterize the depth map implies that this technique is mostly aiming at spot lights. The typical way to handle omnidirectional sources is to create, for example, six spots (one for each side of a cube) that together cover the entire sphere of directions. This solution is currently standard, but implies that faces need to be rendered several times. Recently, geometry shaders can perform this projection on a cube map in a single pass, but the fact that geometry is duplicated for each face introduces a significant penalty. Instead of a cube, Brabec et al. [BAS02] point out that a parabolic mapping [Hei98] allows for extraction of the entire field of view with only two renderings. Furthermore, lookups in these maps are very cheap. The fact that the definition range is not rectangular and that the sampling ratio might vary by a factor of four are two minor reasons why this technique has not yet received more attention. The main reason is probably that creating these maps is difficult. Lines need to be transformed to curves, which is incompatible with the standard rasterization pipeline. The solution in [BAS02] is to transform only vertices in a vertex shader to the correct position and assume that the scene is tesselated enough to provide the correct solution. Recently, Gascuel et al. [GHFP08] proposed to compute the deformed and curved elements on graphics hardware, but the algorithm remains costly.

There are two more important problems related to shadow mapping: depth precision and aliasing. The test whether a point is farther away than the reference in the shadow map requires some depth bias. Otherwise, numerical issues lead to so-called *z-fighting* which results in visible shadow

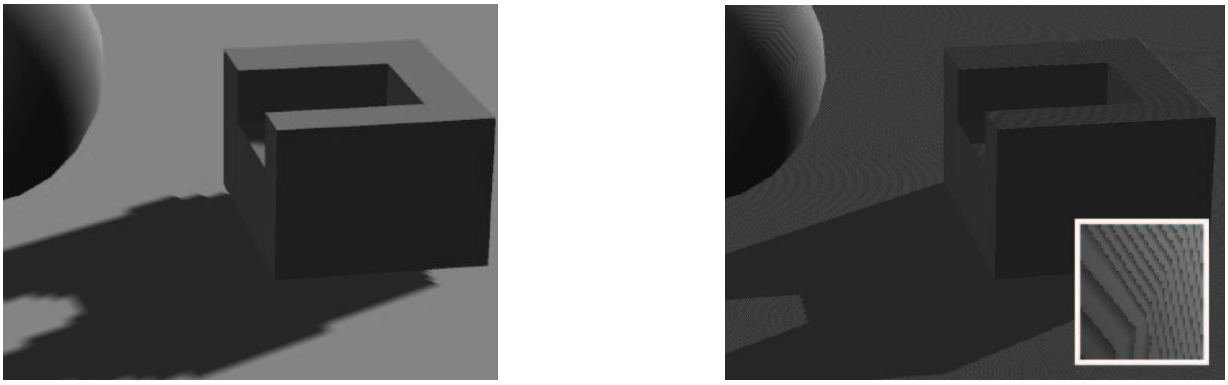


Fig. 10.12 : Shadow mapping artifacts

Shadow mapping suffers from two major problems: aliasing due to insufficient resolution (left) and depth fighting (right) because the values are discretized. The basic solutions to address these problems are as follows. For aliasing one can increase the shadow map resolution. To avoid depth fighting, an offset depth bias is introduced.

sparkles on lit surfaces. Introducing this depth bias is more problematic than it might seem. If a face mostly aligns with the view, a much larger bias can be necessary because, otherwise, the surface might shade itself. On the other hand, for a very short triangle, too much offset would not make any sense, as then the depth would show no more correlation with the actual geometry. This might still sound solvable, but then what about a tesselated planar region? A similar problem presents itself for curved surfaces with low curvature. Finally, the depth buffer on graphics cards is non-linear. This makes sense for hidden surface removal because it puts more precision on nearby elements, but is not necessarily a good choice for shadow mapping. A region far from the light can actually be very close to the observer, and thus have limited precision where most precision is needed.

The fact that the shadow map has only limited resolution leads to aliasing artifacts. A cast shadow will reflect this discretization with an aliasing (stair-stepping) artifact because several view-samples can project into the same shadow map texel. In consequence many modern games use shadow map resolutions that exceed by far the window size: 4096^2 to 8192^2 are typical choices.

In the following, we will quickly discuss approaches to address both: bias and discretization/aliasing. Both of these problems are illustrated in 10.12.

5.1.1 Bias

We will only quickly review the most famous suggestions to lower the bias-concerns.

A straightforward solution to increase depth precision is to better fit the near and far plane of the light's frustum. This can be based on the scene's bounding box, but smarter solutions [BAS05] will be presented in Section 5.1.2 that also address aliasing. Further, this paper describes a solution to linearize depth (an idea introduced in [Hei99]). This further increases precision, but can, today, be achieved directly in the shader, further one can rely on output textures of higher precision.

A classical suggestion is to use indices for each polygon instead of depth samples [HN85]. This eliminates the bias because exact indices are compared. Nevertheless, today, this technique is particularly difficult to use, because many triangles have sub-pixel size, but only one index can be stored per pixel. If alternative representations are used, the attribution of indices is difficult. This led to hybrid visibility determinations that are really slow; the index is used to initialize a search on the original mesh.

A different, more practical solution has been presented by [Woo92]. Two, instead of one, layers of depth are extracted. The shadow map can then be safely set to the average distance. The first layer cannot be shaded, the second will still be shaded due to taking the average. The method does require two passes to extract the first and second layers.

In the same spirit, Wang and Molnar suggest using only the second layer of the shadow map [WM94], but the discretisation of the shadow map can lead to imprecisions (several view-samples fall in the same shadow map texel). In this case the difference seems more robust, but it has its problems.

As Weiskopf and Ertl [WE03] point out, the distance between samples in the two depth maps can be quite large at silhouettes, here problems can occur if the sampling of the depth map was insufficient. Their simple solution is to choose the minimum of a fixed distance and the actual depth distance. In such a way the offset is always limited. In addition, they discuss the possibility of back-face culling for closed/watertight objects, because back-faces are black due to shading. We will exploit this observation in Chapter 13, where we will present a simple solution in the case where the scene consists of watertight elements. We also obtain higher precision by using the actual geometry of the triangular casters instead of a simple depth sample.

In this context, I would like to mention one final technique, even though I could not find any actual reference. The idea is to store a plane or other surface approximations in the depth map, which is easily doable with modern hardware. Apparently, early SGI computers used depth and a plane equation in each pixel, but this information is based on private conversations.

5.1.2 Aliasing

One article concerned with the practical implementation of the shadow mapping algorithm is [BAS05]. The paper uses many graphics-hardware extensions-that might now be considered deprecated-to clamp geometry to the near plane and to linearize depth. Today, according extensions exist and the linearization can be done in a shader. An interesting contribution is the creation of a low-resolution texture from the current viewpoint where colors are used to encode sample positions in light space. This low-resolution texture is then read back on the CPU to best fit the light's frustum to those samples, we present a similar method executed entirely on the GPU in Chapter 13.

Removing artifacts via different constructions: Even though originally not aiming at this goal and mostly of theoretical interest, Zhang et al.'s forward shadow mapping [Zha98] prepared the way for other algorithms in real-time global illumination that use similar strategies [DS06]. The idea is to splat the texels seen from the light to the view. This inverses the usual test which would look up values in the shadow map. Further, it allows us to apply some smooth degrading splat that leads to softer transitions between the samples. In [DS06], a subset of the shadow map texels is used as point lights that are locally evaluated in the view using deferred shading. Each

light renders a quad that englobes the attenuation distance. Every touched view-sample evaluates its distance to the light and adapts its shading. Similar principles are applied in [LZK*07].

A method that reconstructs piece-wise linear shadow boundaries was presented in [SCH03]. It is actually surprising to me that offsprings are still not used in current games. The method is relatively simple and it could be implemented on current graphics cards. The idea is to use two maps: A shadow map that is shifted by half a pixel with respect to a *silhouette map*. The latter stores a point of the silhouette edges in each pixel. To determine the shadow of a view-sample it is projected in the silhouette map and five samples are recovered: the center and its four neighbors. Virtual edges are then added between these sample positions. For each quadrant an evaluation of the shadow map decides on whether the region is considered shadowed or lit (this explains the half-pixel shift of the shadow map). Finally the view-sample is tested against these sectors. The idea to deform the grid locally based on centers stored in pixels has also been used in their follow-up work [Sen04].

Major limitations are that each pixel only stores one center. In areas where two silhouettes project close to each other, noticable artifacts can appear. Nevertheless, the quality of the method is better than for standard shadow maps, although the method remains resolution-dependent. Further, the creation of this map involves a silhouette determination step and a rather costly conservative rasterization that creates the silhouette map.

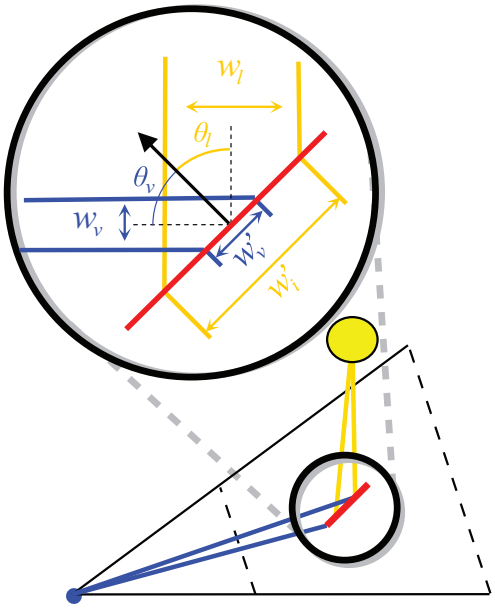


Fig. 10.13 : Perspective and Projective Error - *Aliasing artifacts appear if several view-samples project into the same shadow map texel. This mismatch is related to the shadow map resolution with respect to the projected view pixel size. The general error depends on the scene configuration.*

Shadow map warping: The method in [Sen04] can already be seen as a deformation of the grid, but starting with a uniformly sampled solution. Numerous approaches have been proposed to warp the grid before recording the shadow map to better repartition the samples. A good example is the work by Chong and Gortler [CG04]. They aim at producing high-quality hard shadows on a planar surface. The major insight is that a homogenous transformation can be established that assures a one-to-one pixel-correspondence between the shadow map and the current view. This can be done easily by remapping the intersection points of the view frustum with this plane into the light's view (this is a technique used in vision to rectify the image of a projector on a tilted plane). The result is an optimal sampling and thus, an artifact-free result.

In the same spirit, but without the low-resolution rendering, frustum deformations can be applied to reduce the aliasing artifacts in general scenes. The interesting feature is that these deformations come at no cost because they can be described as a projection matrix. Our work benefits from these adaptations, but we do not provide any contributions in this area; therefore, we will only mention the most important ones.

In Chong's master thesis [Cho03], he considers general scenes and optimizes the frustum by evaluating a low-resolution rendering. But even without this supplementary rendering, which is not assured to deliver any accurate prediction due to

its sampled nature, one can provide optimizations of the light's frustum. Following [SD02] and the notations in Figure 10.13, the goal is to achieve a reparametrization that leads to a good resolution match of the shadow map and the view pixels without much prior knowledge of the scene. The goal corresponds to keeping the following ratio close to 1 in every pixel:

$$m = \frac{w'_l}{w'_v} = \frac{w_l}{w_v}\frac{cos\theta_v}{cost\theta_l} \tag{10.7}$$

The right-hand-side of Equation 10.7 is in fact comprised of two factored error terms. The first w_l/w_v is referred to as the *perspective aliasing*, and the second $cos\theta_v/cos\theta_l$ as the *projection aliasing*. $cos\theta_i/cos\theta_l$ depends on the visible elements in the scene, making it difficult, though not impossible, to optimize. Lately, many algorithms rely on a pre-rendering that is used to make quality adjustments based on this information [DBD*07] and it is likely that, in the future, this option will become increasingly interesting. w_l/w_i is independent of the scene and thus allows a general examination. In most cases, the error analysis assumes the light position to be perpendicular to the viewing direction. This seems reasonable because, most of the time light comes from above (e.g., sun) and our view is usually directed perpendicularly and mostly parellel to the ground. In all algorithms, the reparametrization degrades to standard shadow maps when the light aligns with the view. The latter is actually no problem because, in this particular configuration, a one-to-one mapping is trivially achieved. An interesting article by Zhang et al. [ZXTS06] casts many of these methods in a unified framework and provides a more general error analysis for the non-perpendicular case.

Unfortunately, it turns out that the optimal distribution of sampling points on the shadow map planes is logarithmic [WSP04]. This cannot be implemented easily with the current hardware, but Lloyd et al. [LGMM07] present a solution that describes hardware modifications to achieve logarithmic and thus near-optimal sampling.

Fig. 10.14 : Perspective Shadow Maps - *warp depth maps (inlays) for more precision near the observer*

The perspective shadow maps by Stamminger and Drettakis [SD02] laid the groundwork for approaches in this area. The deformation they use is based on the perspective deformation of the scene. The camera frustum is deformed to yield a unit cube (which is the standard deformation in the rendering pipeline). Then, this deformed geometry is used to compute the shadow map (see Figure 10.14). Unfortunately, mapping singularities can occur and many cases need to be distinguished. A practical implementation guide can be found in [Koz04]. Perspective shadow maps might oversample nearby regions and undersample the scene in the distance. This motivated the work by Wimmer et al. [WSP04]. Instead of using directly the view frustum deformation, their solution is to derive a good warping frustum. First a *light space* is derived, where the y-axis is the inverse of the light direction l. For a point light the scene is first warped, according to its projection matrix, to make it a directional light. The z-axis is chosen orthogonally, but lying in the plane defined by the view vector v and l. The x-axis finally complements the construction. Then a new warping frustum is built.

This warping frustum is defined to englobe the intersection of the scene's bounding box with the convex hull of light- plus view frustum. This volume, called *body*, contains all the geometry that can intervene in the shadow process for the current configuration. The warping frustum's view vector is chosen to be orthogonal to the light direction and the near/far planes englobe the body. x, y-coordinates are then chosen to obtain a roughly symmetric situation, by taking the x-coordinate of the transformed viewpoint and the y-coordinate as the mean of the minimum/-maximum y-coordinates of the body. Only a single degree of freedom remains: the distance of the camera center to the near plane. This parameter influences the warping strength. Most work can be classified via the choice of a value for this distance [LTYM06, ZXTS06] - like trapezoidal shadow maps [MT04], which define the distance such that a focal/interest region maps to around 80% of the shadow map content. In [WSP04], a maximum error (based on their improved error metric with respect to [SD02]) is minimized over the z-range. Even perspective shadow maps can be explained in this way by keeping the distance the same as for the original frustum. A complete error analysis is provided in [LTYM06]. The authors further discuss aliasing in the image plane not only along distance and explain how to take advantage of shadow map partitioning.

Adapting the Shadow Maps Warping is one solution, in this short paragraph, we will present work that increases the resolution of the depth buffer virtually by tiling it into several depth maps, until artifacts should no longer be visible.

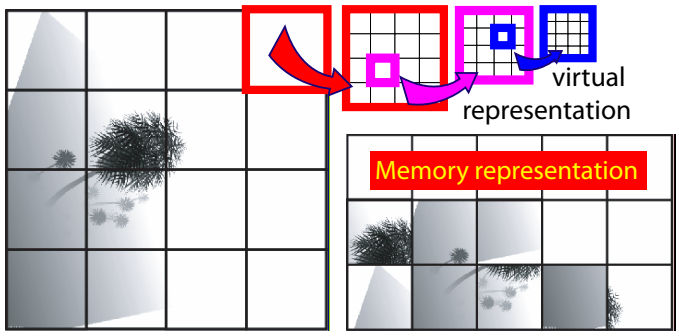


Fig. 10.15 : *Adaptive shadow maps tile a SM. The complex memory layout is hidden by GLIFT*

Adaptive shadow maps (ASM) [FFBG01] was one of the first algorithms to tile a shadow map and thus reach virtual resolutions of impressive size $131,072^2$ [LSK*05] (figure 10.15). Nevertheless, the approach had to execute many steps on the CPU making it rather costly. A GPU implementation from scratch is very challenging due to the complex data allocation and structuring. GLIFT [LKS*06] is a generic, multi-purpose GPU data-structure library which facilitates enormously the implementation of adaptive shadow maps. All important steps of the algorithm can be executed on the GPU. This leads to interactive rates on complex models. The ASM algorithm works as follows. Instead of a fixed shadow map, a quadtree will be sequentially refined. The first rendering step creates a low-resolution shadow map at the base. A Sobel edge detector then identifies shadow boundaries. For each boundary pixel, the mismatch between view and shadow map is found based on its shadow map-coordinate derivatives. These pixels will initialize a refinement. They are grouped in a dense list and read back to the CPU. Here the new quadtree nodes are allocated and the process continues until convergence. If the light does not move, the tiling can be cached for the next frame to optimize performance.

Lefohn et al. [LSO07] point out problems regarding performance and quality. The boundary detection based on derivatives can fail and the read-back (although pixels are grouped) is costly because duplicates (pixels asking for the same refinement) are not eliminated. To ensure quality, they allow all pixels to request a shadow map update. Of course, this would make read-back

even more costly. To decrease the data, they keep only those view-samples whose left and lower neighbors request differing refinements. This eliminates many elements because coherence in the view often implies coherence in the shadow map, an insight we also exploited in our plausible work presented in Chapter 11. The extracted samples are then sorted and all duplicates are elmininated, minimizing data efficiently on the GPU. Further reduction can be achieved by downsampling. Often, the so-missed shadows are likely to be small and on unstructured receivers where the artifacts are less visible. The decreased CPU interaction leads to a substantial speedup. Scenes with 66,000-88,000 triangles still run at around 16-20 Hz.

We will propose a different method in Chapter 13 that might not always lead to similar, but comparable performance (we measured > 60 Hz for 44.000 triangles). Conversely, the implementation effort of our solution is smaller and we achieve guaranteed alias-free hard shadows without any resolution limit (Lefohn et al. [LSO07] usually impose $32,768^2$).

Queried shadow maps by Giegl and Wimmer [GW07b] provide a smart way to decide on subdivisions with minimal CPU interaction. The scene is rendered with a first shadow map applied. Next, the light frustum is tiled and its children each create a new shadow map that is applied to the scene. The basic principle of the method is to measure the impact on quality that a newly rendered tile has on the final result. By using occlusion queries enables counting the differing pixels by discarding all fragments that are similar to the previous shadow-mapped result. Very detailed scenes of up to five million triangles receive high-quality shadows at interactive rates. Nevertheless, the method stays heuristic and the result might not be accurate.

Another work by the same authors *guesses* the resolution for the shadow map tiles directly based on the view-samples. Each view-sample analyzes its imminent neighbors and derives an approximation of the necessary shadow map resolution due to its position variation in light space. This map is then read back to the CPU. Its scattering capabilities help in associating the resolution demands to a fixed grid of shadow map tiles. A hierarchy is then created on these tiles. Starting at the root, the algorithm descends this tree and either subdivides a node (if its resolution is too large for the graphics card), or produces a shadow map and applies the result. An interesting effect is that splits can be along any axis leading also to rectangular shadow maps. The method is much faster than [GW07b], but needs more CPU interaction. Here, the result is not accurate, but a very good estimate.

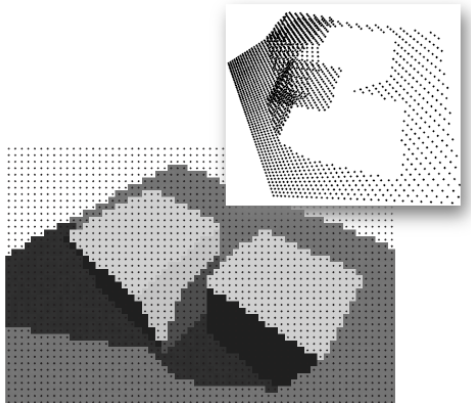


Fig. 10.16 : *Uniform view-samples have a non-uniform light space pattern.*

Per view-sample accuracy Aside from shadow map improvements, other approaches invert the process and determine shadows for each view-sample.

Alias-free shadow maps [AL04] project all samples into the shadow map (see Figure 10.16) and create a hierarchical structure subdividing all projected view-samples in groups of around 64 points. The geometry is then rasterized in this tree using a software solution. To accelerate computations, each node further stores the maximum depth extent of the inner points, allowing for a quick skipping of scene parts that lie behind a given triangle. Once a sample is found to be completely in shadow, it can be safely excluded from subsequent pro-

cesses. The method can treat semi-transparent materials directly. The problem is that its implementation on hardware is not, at the moment, feasible, and a CPU solution is too slow for real-time applications. A thousand triangles take about half a second to be shaded. Fortunately, the method scales well with geometry and 1 million triangles can still be shaded in about ten seconds on a 1.6 GHz Pentium 4, which was not even a high-end machine at the time of the article.

Our GPU method in Chapter 13 also delivers view-sample accuracy, but is adapted to graphics cards. Our work is inspired by Johnston et al. [JLBM05]. They propose a hardware modification to store chained lists in each shadow map pixel. Then all view-samples are projected in the shadow map and inserted into the corresponding list. Each scene triangle is then rasterized in the light view and the samples stored in the intersected pixels are tested for occlusion. Finally, when drawing the scene, the result is scattered back from the list into the image. The key insight here is that there is a simple way to structure computations by combining the regular grid imposed by pixels of the shadow map with more advanced data types (chained lists). The latter is a hardware extension that is not likely to exist on current hardware designs in any near future. We show in 13 that extension are no longer necessary, and obtain a more efficient solution based solely on standard hardware.

A novel alternative for improving shadow quality over time has been presented in [SJW07]. The algorithm works just like standard shadow mapping, except that shadow information of the previous frame is remapped into the current view. A weighting decides on the impact of the old value on the new pixel. This gives a temporal shadow smoothing. The main insight in this paper is to weigh according to *confidence*. The shadow confidence for the current view-sample is its distance to the pixel center when projected into the shadow map, motivated from the fact that pixel centers deliver the accurate depth (compare Section 2). From one frame to the next, the light is slightly perturbed and the same steps are applied again; this shifts the projection positions and leads to new depth samples. For a static view, the weights will usually be higher, leading to a better result but slower temporal behavior, whereas while the camera is moving, the weights are decreased. The main limitation is that light and scene basically need to be static.

5.2 Geometry-based Approaches

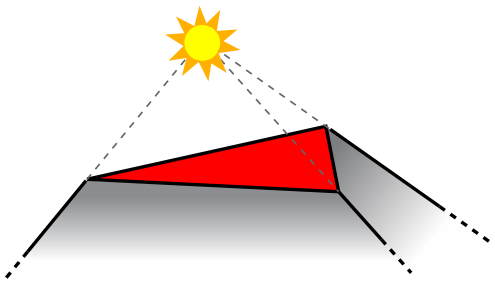


Fig. 10.17 : *A Triangle's Shadow Volume*

One of the earlier solutions to computing shadows was introduced by Crow in 1977 [Cro77]. We have already mentioned it very briefly in chapter8. For completeness, we will present this technique again and mention related and more advanced work. To simplify explanations, we will suppose that the model is *watertight* or *closed*, as defined on page 109. General models were originally discussed by Bergeron [Ber86] and an implementation is given in [SWK07]. The shadow volume is the region in space that lies in shadow, thus all points hidden by the light. For a single triangle, this region is delimited by the triangle itself and its extruded edges. The extrusions are defined by four points each: two corresponding to the edge's extremities, and two to their respective projection from the point light to infinity (see Figure10.17). A point P lies in the

triangle's shadow if it lies in this infinite volume. One realizes that adjacent triangles lead to an inner boundary that can be omitted. Ultimately, only silhouette edges as seen from the light need to be extruded.

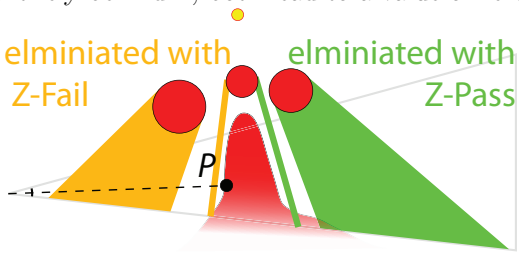
To perform the containment test efficiently, a ray can be shot from a *reference point* outside the shadow to P. A counter is incremented every time the ray enters a volume, and decremented when leaving. If the final number of intersections is even then P is lit else in shadow. This is a direct consequence of the Jordan Theorem.

Efficient implementations [Hei91] send rays from the eye instead of an arbitrary center, making rasterization possible. The stencil buffer can then be used to count the volume intersections. In a first step, the depth buffer is filled from the viewpoint. Then the fragments of the shadow volume geometry increment/decrement the stencil values according to their orientation (front/back-facing) with respect to the eye. This serves as a parity counter for the intersections. Because the depth buffer blocks all shadow volume fragments further than the impact point P, the final stencil buffer entry correctly reflects the number of intersections from the eye up to P. The moment that these operations became usable on graphics hardware represented an important step forward. Earlier approaches clipped the geometry by planes along the shadow volumes to divide the model in lit and shadowed parts. This often involved complex structures like a BSP tree (or even two [CF90]) and although a moving object could be "efficiently" removed and reinserted [CS95], light position changes were almost infeasible.

Nevertheless, there are several problems with the stencil solution. If the camera viewpoint is in shadow, the intersection count will be wrong. A very simple observation is that the depth test can be

Split Shadow Volumes *It seems that many people found the description in [Lai05] a little difficult to follow, even though the basic idea is simple. This short explanation hopefully helps in the undestanding of this work.*

The main observation is that, from one object *to the next, we can toggle between z-fail and z-pass if we assure that the stencil buffer is modified in a coherent way. This can be easily visualized in 2D and generalized to 3D. We will explain the situation in 2D. The stencil manipulation is set such that, if one side of the shadow volume lies in front and the other behind the first visible (impact) point P, both methods (z-fail and z-pass) deliver the value one. This means: z-pass will increment the stencil for the visible front-facing and z-fail for invisible back-facing quads. If a shadow volume lies entirely in front or entirely behind P, both lead to a value of zero.*



The idea is now to eliminate unnecessary stencil updates. This is possible with early z-culling. For z-fail only fragments are drawn that lie behind P. Thus z-fail should be used, if it is more likely that the entire volume lies in front of P. In the opposite case z-pass should be chosen.

To select an appropriate strategy per object, an additional value z_{Split} is computed per pixel and object (the paper defines two ways to do this, but basically it just aims at drawing a quad that approximately splits the shadow volume). It defines a barrier that toggles between the two methods. Based on the depth of z_{Split} and the depth of P, a per-pixel choice is made. If P is closer than z_{Split}, the volume is likely to be behind P, thus z-Pass is chosen (culling all fragments behind P). In the opposite case, z-fail is used.

Of course, one would not gain much if z_{Split} was rasterized at full view-resolution because then each object would draw one supplementary quad leading to an again increased fill-rate. But because the z_{Split} values have no influence on the correctness of the result, only on performance, it is possible to render them in a much lower resolution buffer. Optimally, one whose resolution matches the highest level in the hierarchical z-buffer [GKM93] to ensure the best early culling behavior.

inversed [Car]. In this situation, all shadow volume fragments are counted that lie behind P on the viewray from the eye.

This technique is referred to as z-fail because counted fragments fail the z-test. In this situation, the reference point is no longer the eye, but a point at infinity. One problem is that now *dark-caps* (faces closing the shadow volumes on a plane at infinity) become necessary. For this, it suffices to project light back-facing triangles on the far plane. Graphics hardware provides the nv_depth_clamp extension to clamp depth values between zero and one, thus prohibiting a culling of the primitives beyond the far plane. This allows us to simply send the points to infinity using homogeneous coordinates.

It turns out that z-fail, though more robust, is often slower than z-pass because it is more likely that fragments lie behind the first visible surface, leading to a large amount of updates to stencil buffer. The idea of ZP+ [HHLH05] is to project the scene from the light onto the camera's near plane and thus initialize the stencil buffer from the view with the correct values to allow the application of z-pass. The algorithm is cheap and theoretically simple, but, unfortunately, numerical precision might lead to cracks for single pixels and the correction of these cracks is rather costly, involving a specialized vertex shader. Further, some special cases need to be tested, which makes the code more complex.

The observation that speed can be gained by fewer stencil updates motivated split-shadow volumes [Lai05]. It is an ingenious idea, but in practice, still lacking hardware support and thus currently not being efficient. More details on this algorithm can be found in the sidebox.

In general, shadow volumes show two major costs: silhouette extraction and high fill-rate. Shadow quads can be numerous. For example, for vegetation, almost all edges (borders of leaves) are silhouettes (compare sidebox below) leading to significant overdraw.

5.2.1 Culling and Clamping

Several approaches aim at reducing the overdraw. To mention two important ones: Chan and Durand's solution [CD04], and Lloyd et al.'s CC Shadow Volumes [LWGM04]. We have briefly discussed the latter in Chapter 8 where we proposed a more efficient variation for the GPU implementation. Chan and Durand suggested to have two shadow passes. They first apply a shadow mapping algorithm where all pixels in the view will be marked if they lie on shadow boundaries (detected as discontinuities in the depth map). The marked pixels will be the only ones updated during the following shadow volume pass. To achieve this, the authors make use of the early Z capabilities of the graphics card. By setting the depth buffer to block all fragments outside the marked region, the hierarchical depth representation in the hardware takes care of discarding large areas of fragments. In contrast to Lloyd, who create shadow volumes geometrically where necessary, Chan and Durand cut shadow volumes on a per pixel basis. A combination of both methods

Shadow Volume Problems With Large Models *A large number of accumulated shadow volumes can make the stencil buffer overflow (there are only 255 distinct values). This leads to visible artifacts even when using the wrapping option - 255+1 becomes 0 - and the simultaneous in/decrement extensions on modern GPUs. The silhouette derivation should be performed on the GPU. A brute-force CPU implementation led to a speed difference of a factor* > 10 *for a* $\approx 100,000$ *triangle model. Fast silhouette-extraction methods could be considered, e.g., [OZ06], but this costs CPU performance. The fill-rate also reaches the card's limits if many small polygons create large shadow volumes.*

seems fruitful in practice, but the fact that Chan and Durand's method is so much simpler to implement makes it surely the first choice. A variant has been presented by Aila and Akenine-Möller [AAM04a] who compute intersections of the shadow volume with pixel tiles and mark them as boundaries. They then restrict shadow computation to one tile per region plus these shadow limit tiles. The solution is accurate, but currently slower.

An approach to reduce the actual number of necessary shadow volumes is described in [SWK07]. Here a scene hierarchy is exploited and only visible shadow volumes intervene in the shadow computations; this is conservatively estimated based on bounding-volume hierarchies.

It is also worth noting the approach by McCool [McC00] which uses a shadow map to build shadow volumes from detected discontinuities. It replaces the silhouette extraction step by a simple image-filtering operation. The advantage is that the so-obtained volumes are all star shaped and do not overlap, but might have common boundaries that are rasterized twice. This means that only a parity bit instead of a sum is needed in the stencil buffer. Clear disadvantages are that the shadow volume quads are numerous, and it remains unclear how to better approximate shape. Creating a volume per pixel leads basically to a shadow map equivalent and the potential advantage of geometric shadow boundaries vanishes. Shadow silhouette maps [SCH03] seem a better choice.

Summary:

Hard shadows are now the minimum for a realistic simulation. Image-based approaches suffer from aliasing artifacts for which either costly solutions exist, or pixel-accurate shadows cannot be assured. Very good heuristic approaches and especially very cheap reparameterizations are available but interact with the CPU.

Geometry-based approaches deliver pixel-accurate shadows, but are costly because they tend to create supplementary primitives that result in a high fillrate.

As a consequence, in our work we investigate image-based interactions only for the view-samples (which finally constitute an image), and deliver accurate shadows based on the original blocker geometry. To make this efficient, we try to keep the fillrate as low as possible and do not extend any supplementary geometry.

6 Anti-Aliased Shadows

Several approaches try to make shadows more pleasing to the eye by applying some kind of blur to the shadow boundaries. In the case of image-based methods, this also has the potential to hide the jaggy appearance if the shadow map resolution is insufficient.

6.1 Image-Based Anti-Aliased Shadows

One early approach to hide the jaggy boundaries of shadow maps is percentage-closer filtering (PCF) [RSC87]. Its simplicity and relatively satisfying quality makes this approach particularly

interesting (see Figure 10.18). Standard shadow mapping compares a single depth sample to the depth of the view-sample, which we will refer to as *reference depth*. PCF performs several such evaluations in a small window. The final shadow is the average of lit and shadowed results. A larger window leads to smoother shadows, a smaller to hard shadows. Due to its success, PCF is even available as a function in DX.

Unfortunately, the result can look good, but are not soft-, nor physically plausible shadows. The evaluated rays (corresponding to the shadow map samples) do not leave from the view-sample, but always from the light. This is illustrated in Figure 10.19 (left) and visible in Figure 10.18 because the cast shadows have too large umbrae.

Almost PCF Algorithms *Brabec and Seidel [BS02] use an even coarser approximation than PCF and radially search for the nearest blocking/not blocking shadow map pixel if the view-sample is lit/shadowed. The final shadow is given via a falloff function depending on this radius (similar to [PSS98], see Section 6.1.1). This method can lead to strong over- and underestimations of shadows. The same holds for Kirsch et al. [KD03] who employ a similar idea. Their algorithm derives a width map by iterative filtering of a black and white map of the occluders. This allows them to derive some approximate distance to the shadow boundary. During rendering, they further rely on a difference of the projected view-sample's depth and the shadow map entry it projects in. Both values combined define the shadow's intensity. In this approach, outer penumbrae are missing and the computation is only applied if the pixel is in hard shadow. The major quality problem is that the solutions is easily biased by outliers and overlapping geometry.*

Also related, but more general than PCF, are multiple-depth shadow maps [PCdON04]. If ever up to date, this is currently out of date, but the idea is intriguing. Instead of using all samples in a neighborhood of size l, a best selection of k elements is chosen. These samples are then evaluated with PCF. If k is small compared to l, then there is a potential gain. Unfortunately, the selection of the k elements is usually costly and the paper further limits the choice to $k = 2,3$. Nevertheless, I believe that this selection process is an idea to keep in mind for the future.

One has to pay particular attention to the depth bias. It was problematic for standard shadow maps. For PCF it is worse. Imagine a point on a plane; using its depth to compare against the shadow map results in the plane shading the point itself (Figure 10.19, right). One solution is to estimate a derivative at the current view-sample to get a reference plane for the testing step [Sch06]. This fails in area of high curvature. Brabec and Seidel [BS02] use an index map [HN85] to avoid samples of the same object, but a per-object index removes self-shadows, a per-triangle index can be similar to depth tests.

A generally important drawback of a larger window is that more samples need evaluation. This quickly becomes costly.

Fig. 10.18 : *Luxo* *Courtesy of Pixar [RSC87]*

Brabec and Seidel [BS02] proposed to linearly interpolate between view-samples if the result is similar. A better solution to optimize the sample evaluation is presented in [Ura05]. It is more adapted to newer cards than the work by Brabec and Seidel [BS01], which was optimized for previous generations. In a first step, a small set of samples is used. If the shadow result is not unanimous, only then are more samples evaluated. This leads to a strong speedup and samples are added mostly in the penumbra region, where they are needed. As in many sampling scenarios, banding can occur when several

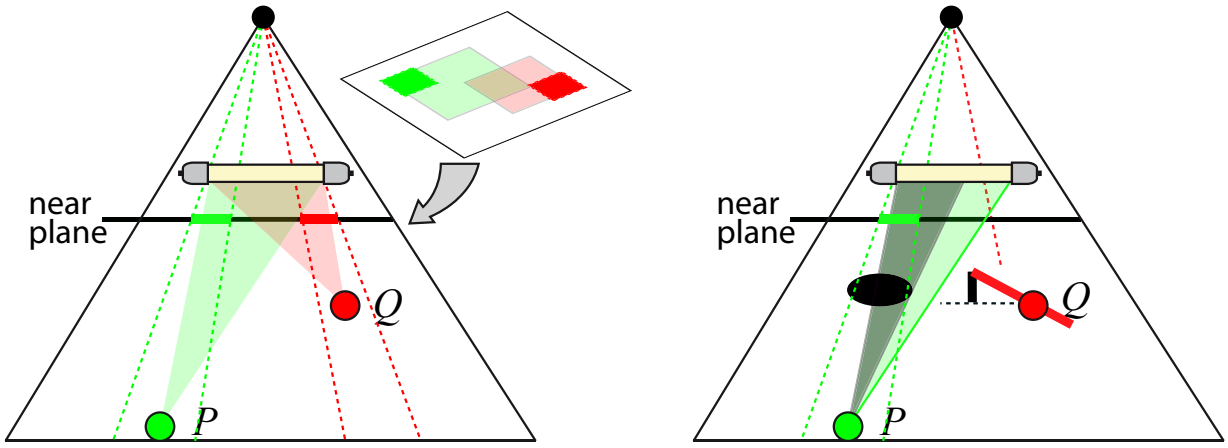


Fig. 10.19 : Percentage-closer filtering

Percentage Closer Filtering evaluates, for each point of the scene, a constant window around the projection into the shadow map (dashed border). This is a coarse approximation. First, these windows should have different sizes for different distances to the light (transparent areas); second, the rays leaving from the light are not the rays that should be integrated. In fact, for each point, only one of the evaluated rays is correct, namely the one passing through the point itself. Therefore, this often creates umbrae where a physically based shadow would not show one (right, P). The right-hand-side illustrates a point Q that is half-shaded even though it is visible to the light. This results from testing the shadow map texels with a constant receiver distance, thus the local surface shades the point.

neighboring pixels share the same sampling pattern. Choosing the sampling pattern on a per view-sample basis replaces these artifacts with less objectionable noise [Ura05].

It would be so much easier if one could pre-blur the values for PCF. Unfortunately, preblurring does not make much sense on depth values, even though there are methods, that use a blurred depth map to smooth the boundary of shadows [LD04]. In these cases, the shadow step function is usually replaced by some smoother, continuous variation. With a lot of parameter, tweaking those shadows can be visually pleasing (we used them in [BEDT08] to achieve more comic looking shadows, which seemed nicer in the context of toon shading), but they are not physically explainable, and can be very far from any physical reference. Unfortunately, this is impossible because of the dependence on a reference depth. Interestingly, Donnelly and Lauritzen [DL06] found a way to introduce meaningful pre-blurring operations. Their Variance Shadow Maps are a beautiful example of a simple mathematical formula that leads to a powerful algorithm. Their observation is that probability can be used to derive a lower bound on how many depth samples will be behind a given reference depth. In other words, they provide a lower bound on the brightness of the shadow. In the special case of one planar receiver and one planar occluder, this bound is even exact. More precisely, they rely on the Chebychev inequality:

$$p(x \geq t) \approx p(t) = \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} \tag{10.8}$$

The equation is valid only for $t > \mu$, otherwise its value is set to 1. The key is that this simple, rational function depends solely on the reference distance and the variance/mean (σ,μ) of the depth distributions in a given window. The latter two can be precomputed efficiently with

filtering operations applied to the depth and a squared-depth map. The resulting images show that effects similar to percentage-closer filtering with large kernels can be achieved, while avoiding the usually related high cost. The major disadvantage is that the bound can be very approximate. If the variance is high (*one occluder on top of another*), light leaks can occur. This is somewhat coherent with the missing information from the depth map (*does the lower occluder have a hole where the upper is hiding it?*).

Layered Variance Shadow Maps [LM08] aim at solving this issue. The observation is that strongly differing depth samples result in a large variance. Therefore, depth samples should be closer together, particularly because PCF evaluation for a reference depth of t delivers the exact same result if all depth samples closer to the light are moved to depth $t-\varepsilon$. In other words, the actual distance from which the depth samples are away from t is not interesting, only whether they are above or below. This insight led to the idea of a warping function. If all samples are warped by

$$\phi_t(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0 & \text{else} \end{cases}$$

, then the result for a reference depth t with Variance Shadow Maps would be equivalent to the accurate result of the PCF. The warping functions cannot be chosen per view-sample. Instead, the scene is sliced into depth intervals $\{[p_i, p_{i+1}]\}_{i=0}^{n}$. Inside each such layer, the depth is warped linearly.

$$\phi_i(x) == \begin{cases} 1, & \text{if } x \geq p_{i+1} \\ (x-p_i)/(p_{i+1}-p_i) & \text{if } x \in [p_i, p_{i+1}] \\ 0, & \text{if } x < p_i \end{cases}$$

This leads to smaller variance estimates and, thus, better shadow behavior. To choose the separations for these layers, an estimation for light bleeding is performed on the current depth map. The method is very efficient because during evaluation each sample only needs to test the layer into which it is projecting. The algorithm leads to much better results than standard Variance Shadow Maps, but to avoid light bleeding completely, many layers are needed. The paper gives an outlook on non-linear mappings that seem to perform much better and need many fewer layers, but an accurate evaluation is missing.

The idea to slice a scene and flatten the casters on the fly was also what drove our work in Chapter 11. The difference is that our shadows are soft shadows and do not need a fixed window size. Further, we extract our layers directly from the scene, thus keeping hidden geometry, not only a single depth map. Missing fragments are one of the major reasons for strong light leaks. On the other hand, our representation has a strong depth discretization. In [ED06c], we also show that summed area tables [Cro84] can be applied in the context of shadows. Lauritzen recently applied SATs to improve Variance Shadow Maps [Lau08].

Convolution shadow maps [AMB*07] follow the same spirit as [DL06], the goal being a way to allow pre-filtering to approximate PCF. The main difference is that instead of basing the result on probability, Annen et al. derive an equivalent representation using Fourier expansion. This expansion has unlimited coefficients, but truncating leads to an approximation of the final result. One major difference to Variance Shadow Maps is that the result converges toward the exact solution. They proceed as follows:

Let P be the point to be shaded and P_s its projection into the shadow map. PCF computes:

$$shadow(P) = \sum_{y \in Neighborhood(P_s)} \omega(y-p) SM(P(z), d_y),$$

where ω is a filter kernel, SM is the shadow test function comparing $P(z)$ (the distance of p to the light) to d_y (the depth sample at location y). This looks like a convolution, but, as mentioned before, the problem is that the depth test needs to be performed *before* filtering. To overcome this problem, SM is replaced by a combination of basis functions that *linearize* the depth test and seperate the dependencies of the variables.

$$SM(P(z),d) = \sum_{i=1}^{infty} a_i(P(z))B_i(d) \tag{10.9}$$

Using Equation 10.9 one can conclude:

$$\begin{aligned} shadow(P) &= \sum_{y \in Neighborhood(P_s)} \omega(y-P)SM(P(z),d_y) \\ &= \sum_{i=1}^{infty} a_i(P(z)) \sum_{y \in Neighborhood(P_s)} (\omega(y-z)B_i(d_y)) \end{aligned} \tag{10.10}$$

This is now a weighted sum of convolutions, where each one can be computed efficiently. By looking at the terms, we see that it amounts to applying the basis function B_i to the depth map and blurring the result values with the ω kernel.

The big question that remains is how to choose B_i and a_i. The authors tested several possibilities and conclude that Fourier analysis is the best choice. $SM(p(z),d_y)$ can be seen as a step function $step(p(z)-d_y)$. This function can in turn be approximated by a smooth function, e.g. $0.5sigm(p(z)-d_y)+0.5$, where $sigm$ is a sigmoid (s-shaped) function. A Fourier expansion leads to the equation of form 10.9.

In practice, the sum is truncated at 32 coeffcients. Any Fourier truncation can lead to ringing artifacts reflecting the sinusoidal nature of the representation (Gibbs phenomenon). To hide this artifact, the coefficients of higher frequencies are attenuated, but there is no guarantee that the ringing

Link to Image Processing *Convolution shadow maps show a resemblance to Paris and Durand's work [PD06] and their follow-ups focusing on image processing with the bilateral filter [TM98, SB95], that found many applications (e.g., [PSA*04, BPD06, ED04]). I think it is interesting to link PCF to existing accelerated filtering processes and it gives a new way of looking at the problem.*

The goal of [PD06] is to provide an approximation for the bilateral filter.

$$bilateral(x) = \frac{\sum_{y \in Neighborhood(x)} I(y)g(y-x)h(I(y)-I(x))}{\sum_{y \in Neighborhood(x)} g(y-x)h(I(y)-I(x))},$$

*where $I(y)$ describes the value of the image at position y. g,h are usually Gaussian kernels. The filter thus combines pixels that are not only close in distance (g) but, further, have similar color (h). Let us quickly show the similarity to [AMB*07]. Paris and Durand compute the nominator and denominator separately. Let us first look at the denominator. We see that $g(y-x)$ takes the place of ω_y and further $SM(p(z),d_y) = step(p(z)-d_y)$. So if h is chosen to be a step function, we obtain the same equation. Paris and Durand then use a similar key insight to replace h by an approximation via basis functions. $h(I(y)-I(x)) = \sum_i \delta(I(x)-I_i)h(I_i-I(x))$. Where delta is measuring the similarity between $I(x)$ to the fixed samples I_i. This equation is already of the same form as Equation 10.9. The computation is thus separable and both approaches perform similar computations. For shadow computation, this weight is already the final output as it describes how many pixels intervene in the computation (thus, let light pass to the source). However, the derivation of the basis functions does differ. Paris and Durand rely on a linearization similar to previous work by Durand and Dorsey [DD02] which slices the function with respect to a set of reference values. It could be interesting to see whether one approach could benefit from the other's solution.*

disappears. Further, the approximation always results in a C^∞ function, whereas a step function is not even C^0. This has important consequences. On the exact shadow boundary, we will encounter a value of 0.5 due to symmetry. This leads to shadow and light leaking. The authors decided that light leaking is less objectionable and add offsets and scaling to shift the function appropriately. The method is reasonably fast (around 60 Hz on a G80 GTX) for complex scenes, and gives good antialiasing because mipmapping or even anistropic filtering are meaningful when applied to the basis functions.

One major cost factor of the method is that 32 coefficients, even with only eight-bit precision, result in 8 textures (each 4 color channels) that need to be written per pass. This lead to an extension of their previous algorithm, in which Annen et al. [AMS*08] replace the basis function approximation by a simple exponential:

$$SM(p(z), d_y) = e^{-c(d_y - p(z))}, \tag{10.11}$$

where c is a large constant. In concurrent work, Silva [Sil08] developed a very similar solution that was presented slightly earlier at the GDC 2008. The main insight is that $d_{p_s} - p(z) \leq 0$, meaning that the depth map stores the nearest surface, thus any point in the scene should project on or behind, not in front. With this assumption, Equation 10.11 behaves almost like a step function which is steeper for larger c. In practice, a value of 80 seems to be a good choice. Larger values can lead to an overflow due to numerical inaccuracies. Applying the filtering operation to this function leads to:

$$\begin{aligned} shadow(p) &= \sum_{y \in Neighborhood(p_s)} \omega(y-p) SM(p(z), d_y) \\ &= \sum_{y \in Neighborhood(p_s)} \omega(y-p) e^{-c(d_y - p(z))} \\ &= e^{c\ p(z)} \sum_{y \in Neighborhood(p_s)} (\omega(y-p) e^{-c(d_y)}) \end{aligned}$$

Thus the two terms are again separated, but, a single 32-bit information is sufficient this time.

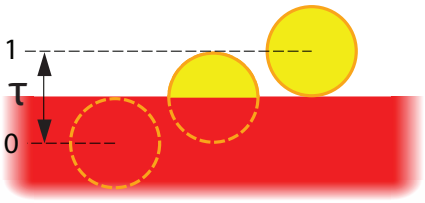
Unfortunately, for a y in the neighborhood of p_s, the assumption $d_y - p(z) \leq 0$ does not hold. As a consequence, large positive values can occur. This is a problem because it invalidates the summation as the exponential no longer behaves like a step function. To avoid this problem, these pixels are detected and all of them resort to a standard PCF filtering (which can be performed with the exponential map by clamping each single value before the summation). Two possibilities are proposed to classify pixels as erroneous. A first strategy tests if the filtered value exceeds one. This solution is very approximate, but fast. A better classification can be performed by precomputing the maximum z-value for each filtering window. This is exact but costly in practice. Although - using the approximate classification - this approach leads to an approximate speedup of $2-3$ over [AMB*07], the performance depends highly on the scene and all presented test cases contain relatively planar surfaces. This is a problem because whenever silhouettes from the light are present, these are marked to be evaluated with PCF. Grass on the ground, unstructured surfaces, or other fine geometry can result in many pixels being treated with PCF up to the point that the gained speed-up can completely vanish (I verified this claim by contacting the authors of the paper). Memory cost, on the other hand, is always improved by an important factor of 8.

6.1.1 Geometry-Based Anti-Aliased Shadows

Parker et al. [PSS98] presented a very simple technique to achieve a soft-shadow-like effect in the context of ray tracing. The main principle is to derive shadow intensity from a single sample ray based on how closely the ray passes to the original object. To find this distance, each object is surrounded by an englobing geometry that is tested for intersection[6]. Many approaches took inspiration from their shadow computation, to name a few: [CD03, MPW07, AHT04]. We give more details on the *shadow measure* in the corresponding sidebox (below).

Haines [Hai01] avoids ray tracing by attaching outer penumbrae to shadow boundaries on a planar ground. He uses the same approximations, but the algorithm is somewhat complex because it uses the z-Buffer to blend different shadow contributions. The darker the shadow, the higher the constructed element, hence the name *plateau*. The camera is placed orthogonally to the planar receiver. Each projected vertex and silhouette are transformed in a 3D shape that, projected, delivers the quad for an edge and a circular approximation for a vertex with correct shading. A big problem is that the construction of these volumes is not easy and they can contain hyperbolic surfaces if, for example, two vertices adjacent to an edge are not at the same distance from the light. Shading in graphics hardware makes the solution somewhat obsolete and the z-buffer method can be replaced by a blending operation.

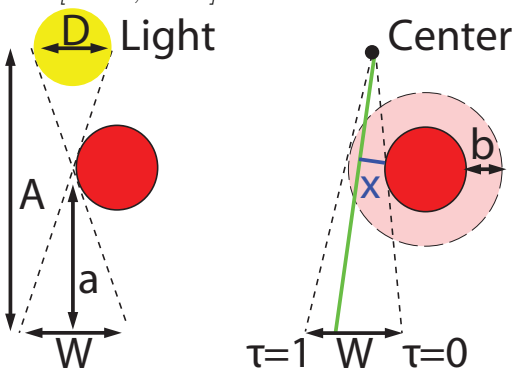
Parker et al.'s One Sample Soft Shadows *The assumption is that the surface is locally a plane hiding a part of a spherical source, as illustrated in the figure below.*



Parker et al. derived the following formula that describes the amount of the source that is hidden with respect to the source's relative position:

$$s(\tau) = (1 + sin(\pi\tau - \pi/2))/2$$

They point out that a simple polynomial matching (of values and derivatives at the extremities) gives the simple expression: $s(\tau) = 3\tau^2 - 2\tau^3$. *To further reduce computational costs, the value can be precomputed and stored in a texture [Hai01, CD03].*



Another contribution was the way to approximate the penumbra region. This is illustrated in the figure above. The goal is to provide a smooth variation of size equivalent to W on the left side. The source is replaced by a point light and each view-sample shoots a single ray toward it. Inside the hard shadow, the illumination is still assumed to be zero (only an outer penumbra is added). Otherwise, the shortest distance x of the ray to the object is computed. b should be aD/A because $W \approx aD/(A-a)$. τ can then be set to x/b to achieve the wanted behavior.

Chan and Durand [CD03] propose a related solution for arbitrary receivers exploiting newer graphics hardware. They make use of additional primitives, the *smoothies*, produced per

[6] The text in [PSS98] states that each triangle is enlarged per ray. This sounds unnecessarily costly, many people thus assumed/proposed that bounding shapes are chosen conservatively in advance.

frame and attached at each silhouette and vertex. These small quads are of constant size in the light's view and orthogonal to the light direction. Each such *fin* encodes the distance from its edge/vertex with colors. Rendered from the source this results in a *smoothy mask*. To deal with superposition of smoothies, a *min blending* is used to keep the shortest distance per pixel. The resulting mask can then be queried via simple texture lookups. In a second rendering step, a depth buffer for the smoothies is derived. In the third and final rendering step, each view-sample is then classified as in shadow (standard shadow map) or in "penumbra" (smoothy depth). For the latter, a soft transition is computed based on the distance stored in the smoothy mask. Using this distance directly would lead to constant sized penumbrae even when the object is close to the receiver. Therefore, the authors use the same ratio formula as in [PSS98]. One problem is that this method does rely on a silhouette extraction step that can be costly. Further, all smoothies are rasterized twice and recreated for each step and need to be sent to the card. The algorithm also inherits the aliasing artifacts of shadow maps where a blocker is close to the receiver. Here, smooth transitions are too small to hide the aliased hard shadow. A follow-up work added interior smoothies to produce soft shadows and will be discussed in Section 7.3.2.

Wyman and Hansen [WH03] make the good observation that, like for smoothies, only an outer penumbra is added, the entire penumbra region is visible from the light. This allows us to rasterize the soft shadows directly into a texture that can then be applied just like a shadow map. The algorithm starts by filling the depth buffer and computing a depth map. In a second pass, the penumbra map is created. Each vertex on the silhouette is extruded to a cone corresponding to the spherical light source projected through the vertex. Silhouette edges are transformed to sheets connecting these cones tangentially, which is a similar construct as used for the accurate penumbra region determination (compare Chapter A for more details on the exact computation in the presence of spherical light sources). While rasterizing these sheets in the light's view, a shadow intensity is derived and stored in the *penumbra map*. To compute this value, they rely on an ad-hoc formula that combines the current height of the rasterized geometry, the receiver point from the underlying depth map, and the distance of the occluding element from light (this one is passed via texture coordinates). The approach is not very fast because of the silhouette extraction and geometry production. It delivers overestimated shadows and uses a discretized map despite the geometry extraction.

Summary:

Anti-aliased shadows are a good compromise to hide aliasing artifacts. Currently, these solutions are usually more efficient to compute than real soft shadows but less realistic.

For most game applications, this seems to be currently the best choice. But even in movie productions, percentage-closer filtering sometimes finds applications (e.g., *Luxo* by Pixar). One less-cited problem is the fact that most efficient filtering solutions are not compatible with shadow map reparameterizations. Therefore, higher resolutions are needed to capture the same amount of detail as with standard hard shadow mapping.

Geometry-based approaches allow us to trade off pixel computations against triangle processing. Nevertheless, many of these project the results into image space. Aliasing artifacts can remain.

The degree of realism can be very high for smaller sources, but as soon as the light becomes larger, the results are no longer convincing.

7 Soft Shadows

In this final section, we will investigate the soft-shadow algorithms. They all share the goal of creating convincing and varying penumbrae. In particular, algorithms in this section lead to more realistic penumbrae, and all share the criterion that the shadow of an object smaller than the source starts disappearing when it approaches the light. Even though this might sound like an arbitrary choice, it is actually a very good criterion. Other methods aim at smoothing the boundary of a shadow. With this alone, the previously described effect remains impossible.

7.1 Ambient Occlusion

Fig. 10.20 : Ambient Occlusion (Images by Malmer et al. [MMAH07])

Ambient Occlusion often looks very convincing and is used in the industry for feature-film productions. It delivers very smooth and visually pleasing shadows (left, middle). One drawback is that, based on the "light from hemisphere" assumption, shading on the objects and "cast" shadows on the environment might look incoherent. In the lower image, the zoomed region would have received much more light if there had been a real caster that produced the shadow on the ground.

Ambient occlusion is a method to compute shadows based on the assumption that light is impinging uniformly from all directions of the hemisphere at a point. This sounds like a crude approximation, but, surprisingly, it leads to very convincing and especially smooth shading. Ambient occlusion can be seen as an accessibility value [Mil94]. The less a point in the scene is accessible from the outside, the darker it appears. The motivation behind this is that in the real world, we do not have a single point or area light source. Instead, light bounces off all surfaces; it is coming from "everywhere". An approximation such as ambient occlusion thus has a look of indirect illumination.

This alone would not be reason enough to mention it in this chapter where we focus on interactive cast shadows. The real reason is that "shadows" can "bleed" onto nearby elements of the scene. Newer approaches allow us to evaluate either ambient occlusion on the fly [Bun06,HJ08], or *attach*

ambient occlusion values to the space around an object. In the latter case, visibility information is determined in a preprocess. Thus, the geometry of each object needs to remain static.

Kontkanen and Laine [KL05] use a cube map surrounding the object where each texel entry stores occlusion information in form of a quadratic rational function of distance. It is evaluated for each pixel to derive its shading. They further combine several occluders based on a heuristic.

A faster and simpler solution was presented by Malmer et al. [MMAH07]. They store ambient occlusion directly for points in space in a 3D texture surrounding the object. Theoretically, this results in an infinite support because some light should always be blocked by an object even when far away from the impact point. To obtain a finite support, they clamp the shadow values and ensure that outside of a fixed radius no more shading influence exists. Figure 10.20 shows an example of their results. The shadows are generally very blurry, which gives the images a nice feel.

Both solutions mentioned here are extremely fast. To our knowledge, no other current method leads to penumbrae in comparable size at a comparable speed, with the exception of our solution presented in Chapter 11 that delivers plausible soft shadows for an area light source. Another strength of ambient occlusion is that the methods are usually very temporally coherent. No high frequency defects appear in the image if the result is precomputed. They can occur for on-the-fly evaluations like [Bun06] due to geometric approximations. Unfortunately, incoherences can always show and shadows might look incorrect and "attached" as illustrated in Figure 10.20.

Realism can be increased by involving a direction of minimum occlusion for each vertex during shading [Lan02], but only rather smooth shadows are possible with these techniques. Occluder fusion (as for many approaches) remains problematic in all these per-object approaches.

7.2 Image-Based Soft Shadow Approaches

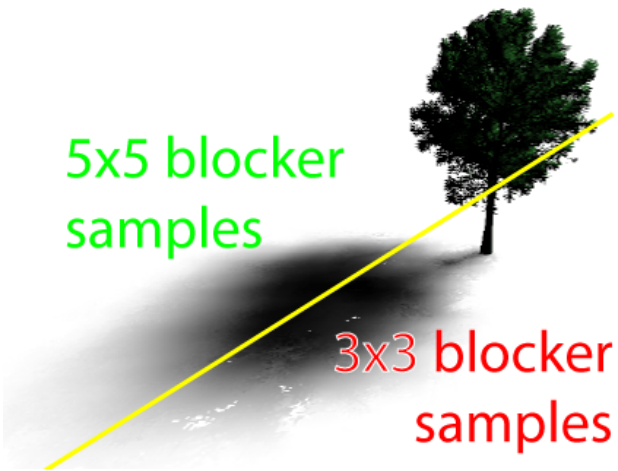


Fig. 10.21 : *Imprecise blocker estimation leads to artifacts* *(Based on Images by Bavoil)*

One natural question after having seen all the anti-aliased shadow solutions is whether there is not a way to adapt them to produce soft shadows. Actually, it is relatively straightforward to increase the realism.

Fernando presented the very simple percentage-closer soft shadows [Fer]. The idea is to use a fixed-size window to test the surrounding depth values of a given pixel. The depths closer to the light are averaged to derive an average occluder distance d. Like Parker et al. [PSS98], he then derives the penumbra size s for a half-space blocker situated at distance d. This size s is used to guide the kernel size of a standard PCF process. This method is compatible to many approaches, e.g., [AMB*07, AMS*08, LM08, Lau08]. Unfortunately, deriving the average blocker distance is as expensive as PCF (usually up to 36 samples are used), but approaches such as [AMS*08] might allow adaptations to address this. The solution can be very approximate and costly and is prone

to noise. If the blocker samples are not chosen wisely and sufficiently two neighboring receiver pixels might compute a very different average distance and, thus, a very different window size and shadow (see Figure 10.21). In the case of the small ball approaching the center of the light, the average will be very high. In consequence, the PCF window will almost the entire screen. This leads to an elevated cost for each pixel in the scene and a approximate soft shadow.

7.2.1 Backprojection

A more accurate solution was presented by Atty et al. [AHL*06] and initiated a variety of papers going in a similar direction. In Atty et al.'s work, the scene is assumed to be seperated into a height field receiver (as seen from the light) and a set of casters. First, a depth map is derived involving only the occluders. This map is then read back to the CPU, where for each pixel a corresponding quad, called a *micro-patch*, is created and passed to the GPU. There, the extent of the penumbra for this single quad is computed and projected on the receiver. Then, a fragment shader is executed which projects the quad back onto the source from each of the touched height field samples. The shader computes the covered area of the source and blends this coverage with the stored value from previous micro-patches on the receiver. The final shadow is a result of additively accumulating the blocking contribution of all these quads. Due to the fact that the pixel quads are chosen to be aligned with the rectangular source, the intersection computations can be simplified aggressively and correspond to simple lookups. The obtained shadows can be convincing, but can also fail often. Furthermore, the CPU intervention leads to a significant overhead. Due to the basis of this approach, only the first depth layer of the scene is exploitable. The authors point out that they could use more, but this introduces a performance penalty. Furthermore, overlapping reprojections of micro-patches are not detected and thus shadows would be further overestimated. Inversely, if projections create gaps significant artifacts can occur because of an underestimated occlusion. The shadows often lack realism if the depth map resolution is too low, and aliasing becomes obvious. Increasing resolution is difficult due to the CPU bottleneck. Even though today this step could be implemented in hardware, the method can be considered outdated. It nevertheless represents a milestone because the idea of interpreting the depth-map texels as micro-geometry opened up the road for a variety of approaches.

The work by Guennebaud et al. [GBP06] shows significant similarity. They lift many of the restrictions that applied to [AHL*06]. Namely, gaps and overlaps of micro-patches are somewhat addressed, the CPU interaction was eliminated, and the caster/receiver separation lifted. The key was to derive a conservative window for the depth map that contains all samples that project on the source from a given point p. For this, a *hierarchical shadow map (HSM)* based on a min/max mipmap hiearchy is created. In a first step, the search window is initialized as the intersection of the cone p towards the light source, with the near plane. Then the HSM is used to lookup the minimum z-value d in this window. Based on this result, a new window is derived by intersecting the cone with a plane at distance d. The process can be continued until the change can be neglected or after a fixed amount of iterations. Then, for all depth samples in this region, the reprojection on the source is computed. To account for overlapping and gaps, neighboring samples are investigated and the quads slightly modified to join properly. This does not eliminate all of the artifacts, but does a good job locally. The blocking contribution is then evaluated by backprojecting the samples on the source and computing the blocked region. The value is then blended with the previous values at the receiver position p. To accelerate the

method, a maximum number of backprojections can be chosen. In this case, an appropriate level in the HSM is selected as a function of the search area. Although the speed of the method is acceptable for small sources, it can become really costly (as low as 2 Hz for a single quad on a GeForce 7800) if the search area is too large. The hierarchical solution leads to a speedup (in the above case even up to 24 Hz), but can create discontinuities.

In the following, we will examine several much similar approaches. The important points to keep in mind during the discussion are:

- **micropatches** how are depth samples interpreted and adapted to fill gaps, this also includes dealing with discontinuities of a hiearchical solution.
- **search window** how to determine the search window size
- **blending** how to improve occluder fusion of mircropatches
- **acceleration** how to accelerate computations and decrease the number of view-samples.

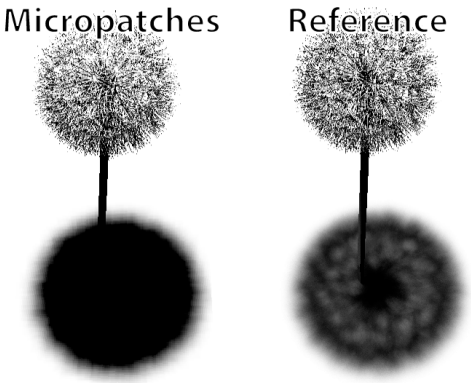


Fig. 10.22 : *Gap-Filling Problem (Image by Bavoil et al. [BCS08])*

Bavoil et al. further [BCS08] point out two possible ameliorations of [GBP06]. The previous ***gap-filling*** can lead to overly dark results for fine structures (see Figure 10.22, thus, instead, they propose to perform depth-peeling and recover several layers that they process consecutively, without any adaptation. It closes many gaps, but might lead to overly dark shadows as well. Another improvement concerns the bias. They use the method in [WE03] and test whether a view-sample is shaded using some depth comparisons in a fixed window. The shadow computation is evaluated only if this is the case (similar to [Ura05]). It avoids some of the surface acne, but might lead to incoherencies.

In my opinion, Guennebaud et al.'s follow-up paper [GBP07] better addresses some of their shortcomings. One major focus was on ***speeding up*** their previous work. The first observation is that a lower shadow map resolution is leading to fewer samples, but then also to aliasing. Therefore, they use a strategy to improve the shadow map interpretation, not unlike [SCH03]. But, they skip the sampling of the silhouette and directly reconstruct borders in a marching cube fashion, based on the outcome of the depth test. In other words, they evaluate the depth map in 4×4 subwindows and create a polyline along the boundaries, which replace the ***micropatches***. They further introduce an offset parameter which allows them to shrink or extend these boundaries continuously by moving the edges in or outwards. This is important when they ***blend*** with higher levels in the HSM map because its construction, by taking the minimum value, overestimates blockers and thus usually darkens the image. Shifting the occluder boundaries inwards allows them to lighten the result. Of course, this does not solve issues due to limited details arising from a low resolution and is purely heuristic. They address the ***discontinuity problem*** of the level selection, by blending between two levels whenever a slight change in distance would result in a different level. This is potentially costly and as we will see later, this is still insufficient to obtain a smooth transition. To integrate the resulting polylines, they use the algorithm by Assarsson et al. [ADMAM03] (see section7.3.1). They do not face the problem of superposing occluders because (as for [McC00]) the reconstruction of the occluder boundary is based on the depth map.

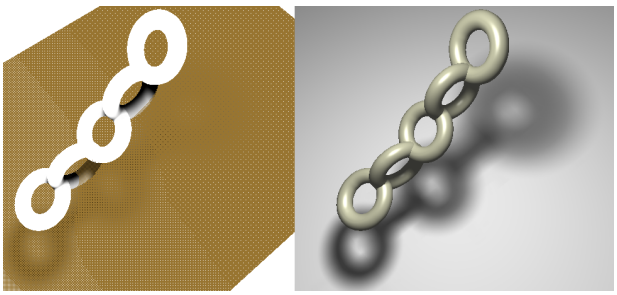


Fig. 10.23 : Skipped samples (orange) - *are filled in a discontinuity respecting filtering process (right). [GBP07]*

Another major and smart contribution is to select only a limited number of ***view-samples***. This idea was already presented in different contexts before: Segoiva et al. [SIMP06] proposed a GPU approache, related to [KH01], to evaluate many light samples. Here, deferred shading is organized by regularly subsampling the image and grouping the result in coherent blocks. The second step is a reconstruction that uses a filtering process respecting discontinuities. In [GBP07] a similar process is applied to shadow computations. A simple heuristic estimates the penumbra variation as seen from the viewer. A fitness value is computed per pixel, which involves the projected screen size of a sample area based on the local normal and penumbra width (derived from the mean depth in the search window). To select a view-sample subset, they apply a dithering mask successively on the image and keep only the values that are inferior to the mask's thresholds. The final pixels are packed and then successively evaluated. The result needs to be splatted on the final image. For this, strong depth and normal differences are detected and an iterative method performs a value propagation whilst respecting these discontinuities (see Figure 10.23). The method delivers much better quality than the previous solution, but the result remains approximate.

Various spin-offs appeared approximately at the same time. Many researchers seem to have had similar ideas. The major difference is usually on how to interpret the micropatches. In [BS06], the solution is to work with discs as ***micropatches*** in the presence of a spherical light source. The rounded shape has a nice effect on the shadows, but the accumulation is additive, leading to overly dark shadows in the case of overlapping occluders. Similar to the others the search window uses only a fixed amount of depth samples selected from an appropriate mip-map level to achieve an acceptable frame-rate. This can create temporal incoherence.

One straightforward way around ***discontinuities*** due to the hierarchical selection, is to always use a constant sized ***window*** [ASK06]. All reprojected samples are thus depth map texels that are blended additively. Nevertheless, this significantly restricts the light's size.

Schwarz and Stamminger [SS07] recently presented an approach using bitmasks to maintain knowledge about visibility. It provides two major improvements: the ***blending*** strategy of the occluders, the interpretation of the ***micropatches*** and the ***search window*** determination. This solution was presented concurrently to our work in Chapter 12. Whereas we compute an accurate solution based on geometry, Schwarz and Stamminger rely on an image-based approach and use bitmasks to solve the problem of overlapping backprojected micropatches. Another difference is that we present a way to perform the blocked sample lookup for arbitrarily placed samples. In the case of bitmask shadow maps, this is not possible. The bitmask of blocked samples is computed *manually* by mapping the axial-extents to bit segments. Therefore, general sample positions are not possible without a strong performance drop. Basically, only uniformly spaced light samples are acceptable. Their solution to obtaining a pseudo-jittering is to decompose the light samples into four shifted groups of uniform and equally spaced samples and perform the bitmask computations independently. To hide the sampling aliasing, they propose to compute shadow values obtained with [GBP06] and interpolate (guided by the bitmask shadows) these smoother results between neighboring pixels.

Another modification is the use of microquads instead of micropatches. This corresponds to a different way of looking at the depth map. A blocking element is considered to be a quad consisting of four corners placed on adjacent depth samples. Only if all four samples are in front of the current view-sample the backprojection is performed. This can create holes if fine structures are present. Further, it is difficult to perform the backprojection for such general quads. Therefore, they are obliged to derive axis-aligned bounding rectangles instead, which they can transform into bitmasks. This can degrade the solution.

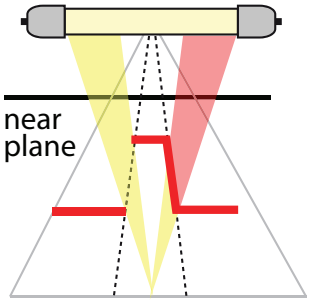


Fig. 10.24 : Ambiguities in a depth map - *Light leak, or blocked? Even depth peeling encounters problems because the face almost aligns with the light direction.*

Rasterization issues left aside, the main benefit of bitmasks would be if depth peeling is used to take the entire scene into account. But this is not only costly in the peeling step, it also forces the shadow computation to be performed for each layer separately, slowing down the method. Also, due to the representation as microquads, it is unclear what should happen, if two occluders overlap. Figure 10.24 illustrates this problem. For a pixel on the ground, the two might receive a connecting wall, which is not necessarily correct, or this wall is eliminated, in which case a gap occurs. Even depth peeling cannot necessarily correct this step. These cases show why really accurate shadows usually need to involve either the entire geometry or really high resolution maps.

Other improvements include the choice of the mip-map level at which to evaluate the structure. An NBuffer [Déc05] like structure is used to prone the search area, but the microquads are constructed from a mipmap representation based on a fixed budget, just like Guennebaud et al. [GBP06]. In consequence, discontinuities can arise whenever adjacent view-samples do not share the same level. The only workaround is a blending operation between two levels, which significantly decreases performance.

The problem of blending between mip-map levels is discussed in Schwarz and Stamminger's follow-up paper [SS08]. They provide many very technical contributions to increase performance and make the previous approach much more practical. The strong performance dependency on the light source size remains, but they improve the algorithm on several levels. The ***search structure*** is split into a hierarchical and an NBuffer part to save storage space. As in [GBP07], they perform ***subsampling of the view-samples*** and introduce the idea to decrease shadow quality where it is masked by texture[7]. They provide a simple scheme sampling the neighboring pixel's shadow results to remove ***aliasing*** and finally introduce a better ***micropatch*** approximation: the so-called microrects. Microrects are stored via a center and rectangular extents. The representation is built from a shadow map in a greedy fashion by merging most similar adjacent texels. This constructs a mipmap hierarchy, where each microrect has certain extents that can leave the original pixel boundary. This makes their shape a better approximation than standard mip map structures, but they can result in significant overlap. In consequence, it becomes necessary to perform bitmask light sampling [SS07]. The timings are very convincing whenever the budget of the window pixels that are used to test visibility, are set to a small amount. For a 5×5 window and a single depth layer, the performance can go up to 80 fps on a 8800 GTX. For 12×12, the timings decrease to 23 fps. The major drawback that remain are the dependency on the light's size and the single depth layer, but this is the case for almost any method of this type.

[7] We proposed this very idea concurrently in [SEA08] (Chapter 13)

7.2.2 More Than Meets the Eye (or the Light's Center)

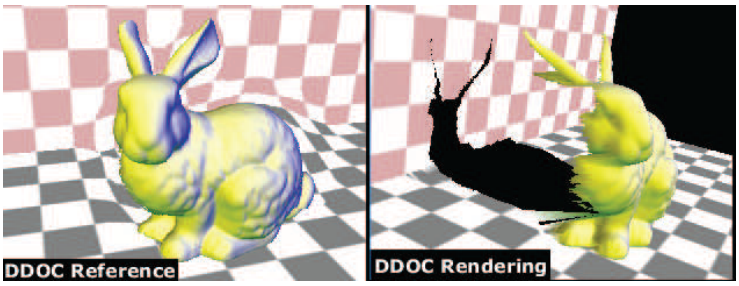


Fig. 10.25 : Occlusion Camera - *Left: Hidden pixels are distorted to become visible. Right: Larger regions can thus be captured. (Courtesy of Mo et al. [MW07])*

All backprojection methods rely usually on a single depth layer. Otherwise the speed impact would be too penalizing. Unfortunately, as illustrated in figure 10.11, this can introduce significant problems. It is thus interesting to have more information in a depth map than just what is visible from the light. This was one motivation for our work in Chapter 11. In the same spirit, Mo et al. [MW07] introduce a non-pinhole camera that sees *around* objects to the shadow community. Even though this idea sounds very tempting, the rasterization of such an image is costly. Instead, the authors only use the distortion information that would be needed for construction to derive their shadow value. This distortion is an approximation of how much rays need to bend around the surface to see the sample. This distortion can then be used in an approximation like in [PSS98]. To avoid light leaking, the distorted position is still tested for visibility from the light's center, otherwise, the sample is assumed to be completely shaded. This is important, as otherwise any two superposing objects would introduce light leaks. To create a distortion map, all light-silhouette edges are transformed to quads and rasterized in the light's view. These quads encode a pixel movement that indicates how to squash the data locally to allow occluded parts to be discovered behind the model. See Figure 10.25. In practice, two maps are used: one to shift pixels from inside the hardshadow region, one to shift pixels from outside.

Because space is limited, the distortion needs to be bound, which is probably why many scenes are rather shallow. Other problems arise from overlapping distortion quads: A heuristic is used, which distorts and displaces a pixel successively as long as its new position falls into a different distortion quad. All encountered blocking contributions are multiplied. In practice, three such jumps are used. Nevertheless, many artifacts arise whenever more than two silhouettes overlap and details are usually lost. The silhouette extraction can be costly but is directly performed in parallel to the rendering for efficiency. Further, the distorted view is never explicitly constructed (which would be too expensive) thus the potential of recovering hidden surface areas is lost. On a very recent system, 3.2 GHz Xeon with a GeForce 8800 GTX, the framerate is around 84 Hz for 64k triangles, but for a relatively small source.

Heidrich et al. [HBS00] discover more information from the scene by using a small set of depth maps. They present a soft shadow algorithm suited for linear light sources. The idea is to interpolate visibility between the two extremities of the light. Of course, this can fail whenever objects between the samples are revealed (for example via a passage), but for smaller sources, it can lead to acceptable results. Although the algorithm deals with general receivers, the approach is motivated using a planar receiver and a set of casters. For the moment, we will assume that only two samples have been created for the extremities of the source. The goal is to derive a visibility map that encodes the percentage of light that reaches a point in the scene. To create this map, edges are detected in the shadow maps and extruded along the light direction, this basically describes shadow volumes. The shadow volumes for one extremity of the light are

rasterized in the view (depth map) of the other. This leads to skins that will be used to define the shadow intensity. It can be shown that if the linear source and the receiver align, the shadow varies linearly in the presence of a blocker. Thus it makes sense to shade the skin from black (at the depth discontinuity) to white (when it reaches the ground) to encode visibility. The same process is repeated for both extremities. The final shading combines the contributions from both additively. The solution is not accurate in general, because any small object (even a quad) can lead to separated or missing penumbrae, but for large aligned occluders, the approximation can become exact [HLHS03]. The algorithm is rather costly due to the geometry derivation from a depth map. For fine geometry, a high resolution is needed, which results in many quads. The shadows can show significant artifacts, but by adding more samples, quality can be improved. Splitting the light source in the middle, is a good choice: the same depth map can be used to construct two visibility maps.

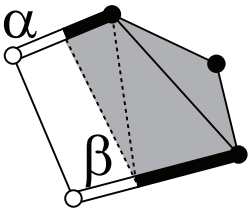


Fig. 10.26 : *The hidden area is approximated from the vertices*

The idea has been extended to area lights by Ying et al. [YTD02]. The light source has to be convex and (I believe) pre-triangulated. Each view-sample P in the scene queries the shadow maps of the light's vertices. Whenever a segment of the light source has one vertex declaring P in shadow, with the other one declaring it lit, Heidrich et al.'s solution is used to derive a fractional visibility α for the segment by interpreting it as a linear light. This value α is then used to approximate the point on the light segment for which P changes visibility status. Figure 10.26 illustrates this situation. The triangulation and these vertices describe lit and occluded areas. The lit part is used as shadow intensity. To accelerate the computation, they show that the lit area can be approximated based on the precomputed areas of the triangulation by using a linear combinations weighted by the fractional visibility. Of course, the approximation of this method is very coarse, and the computation due to the visibility-buffer creation for each segment can be elevated. The light's triangulation should also have an influence on the appearance of the shadows, which underlines that the solution is very approximative.

In general, this the idea to use a set of depth maps would generally be a good solution, but storage requirements and creation time do not necessarily allow us to create the maps on the fly or keep them in memory during execution. Scientific literature provides us with many solutions for tackling this problem, which has been extensively examined. In the following section we will investigate this topic more closely.

7.2.3 Larger Shadow Map Collections

Using n shadow maps, we can compute soft shadows with n samples. This solution was employed in [Hrb]. Obviously, the number of depth maps is very limited and thus shadows will only be acceptable for smaller sources. If the sampling rate is too coarse, banding occurs. We will discuss other solutions to encode a set of depth maps and then query them efficiently.

Besides direct illumination, shadow map sets have even been used successfully for the computation of bounced light [LSK*07]. The well-known property is that light first bounces off those places that are visible from the light (we already shortly mentioned this in Section 5.1.2). If the light only moves by a small amount, and the scene is static, mostly the same areas remain visible for the new light position. Thus, if one places samples in these visible areas and most of these samples will remain in the visible area when the light is moved. If one computes a shadow map

per sample to propagate the energy of the first bounce to the scene, not all of them will need an update in each frame. This works very well for indirect light, but for soft shadows all light samples are on the source. A general movement will thus leave less samples unchanged, and make this direct illumination challenging. Furthermore, the resulting shadows usually have much higher frequencies, thus not allowing a coarse light sampling, whereas for indirect light 256 sources often suffice [LSK*07].

Depth Map Fusion and Real-time Decoding Layered attenuation maps [ARHM00] are a way to compute almost accurate soft shadows by warping together several depth maps. The construction is costly, but the rendering is rather fast. The scene has to be considered static though. For a given set of light samples, the depth buffer pixels are warped into a reference view (typically from the center of the source) and stored in an LDI [SGwHS98]. Samples at a distance smaller than ε are merged. Further, a counter is incremented at this position. In this way, once all the images are warped, each LDI layer will contain a counter of the number of lights that see this particular position in space which gives the shadow intensity. The evaluation of the shadows at run-time becomes relatively simple by looping over the sorted lists and finding the corresponding depth and recover the associated counter. If it does not exist, the pixel was not visible by any of the lights.

St. Amour et al. [SAPP05] use a very similar preprocess with several shadow maps to derive a special penumbra deep shadow map. This shadow map also encodes visibility of the extended light source in a single image. The shadows of the static scene can be applied to any point in space. Thus dynamic objects can be added, but they cannot cast any soft shadows themselves. The representation shows many similarities with deep shadow maps [LV00]. The penumbra deep shadow map encodes in each pixel the shadow variation on the ray towards the light. This is a one-dimensional function and is obtained by projecting the deep shadow ray into each shadow map and encode the change from lit to shadowed. Accumulating the result for all maps leads to the shadow function that allows direct shading of the entire scene volume. This is the major difference to [ARHM00], where the shadow information was stored only in each layer. As for deep shadow maps, the 1D functions are simplified while limiting some deviation. The whole process maps on the GPU and allows many effects. Nevertheless, the light source and the shadow casters are static.

Image-Based Ray Tracing Against Depth Maps We will only quickly review approaches to compute offline shadows involving ray tracing. As we aim at interactive rates, these algorithms are slightly out of the scope of this overview. Therefore, we will only shortly mention them here.

Fig. 10.27 : *PCF vs. Multi-layer Shadow Map ray tracing* *(Courtesy of Feng et al. [XTP07])*

Even though it was not the first paper on image-based rendering with ray tracing, examples are the work by Lischinski and Rappoport [Lis98] or Leonard McMillan's thesis [McM97], Agrawala et al. [ARHM00] clearly showed that soft shadows based on layered depth images can lead to a good quality. Recently, in the same spirit, semi-transparent materials have been added [XTP07].

Keating and Max [KM99] decided to use *Multiple Depth Images*. These are obtained on the CPU, and ray tracing is performed against this alternative representation. There are some similarities to our work and we will have a detailed comparison in Section 2 of Chapter 11.

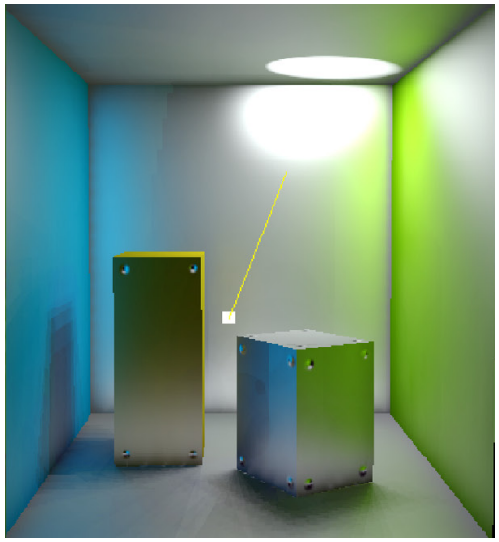


Fig. 10.28 : *Two bounces at 1.3 Hz (Courtesy of Ritschel et al. [RGKS08])*

Ritschel et al. [RGKM07] present a very nice solution to compress a set of shadow maps. If we assume $z_1 < z_2$ are the first two depth samples along a shadow map ray, they use the insight that storing any depth $z \in [z_1, z_2]$ will shade the same surfaces as z_1 would. Therefore, for neighboring depth samples, any sample in the intersection of the two corresponding intervals would do. This process can be continued and enables the derivation of large constant depth areas. This leads to very efficient compression. During rendering, these functions are uncompressed and queried. The method works with lights at infinity, sampled local lights, or what they call semi-local lights (light sources that lie outside the convex hull of the object, because then any light ray can be seen as a ray from an infinite light source). As the tests are purely image-based, the evaluation is fast and reduces the rendering of complex objects from minutes to seconds. In a follow-up [RGKS08], the method is extended to store depth maps on the surfaces of objects. Sources can thus be placed everywhere in the scene opening the road for the application in the context of global illumination (see Figure 10.28). They achieve almost interactive rates through an efficient GPU encoding.

7.2.4 Shadows and Image Operations

Arvo et al. [AHT04] present an approach where the camera view is processed to add soft shadows. This entirely image-based solution detects hard shadow boundaries and adds (in flood-fill steps) penumbrae regions based on depth map information. For this, the hard shadow borders are first detected by applying a Sobel filter. Then, in successive render steps, the detected discontinuities are grown by a single pixel that is added to the interior and exterior of these borders. During this shadow growing, the maximum blocking shadow map sample is propagated. The assumption is that each sample behaves like a half-space which enables the use of Parker's [PSS98] approximation. The cost of the algorithm is directly related to the size of the penumbra on the screen, so it can be efficient for very small penumbrae. But depending on the viewpoint and the light's position, this area can be very large and the flood-fill is not well supported by current graphics hardware. It still remains relatively slow even when using the optimization of jump flood-fill [RT06]. But less than the performance, the quality of the shadows is a major drawback of this method. Temporal incoherence can result from missed penumbra boundaries, for example, where distinct umbras overlap and therefore the shadow border detection fails. Penumbraes of casters that do not project a hard shadow in the image will not be found (for instance, to the left or right of the observer). It is unclear how many iterations are needed to finish the image. The shadow values themselves are almost like a maximum blocker fusion which leads to significant

light leaks and visually unpleasing gradient reversals. The idea, on the other hand, has to be considered really novel and shows a completely different way of tackling the problem.

One of the most inovative approaches to soft-shadow computation was presented by Soler and Sillion [SS98]. They point out that for the configuration of aligned planar receiver, planar occluder, and planar light source, the accurate soft shadow corresponds to a convolution. They then use this information to shade such scenes by convolving planar approximations of the geometry. A major problem is that several occluders cannot be easily combined and, due to the strong approximation, shadows can appear incorrect. Enforcing this very restricting configuration on general casters and receivers leads to strong approximations. Our approach in Section 11 is highly inspired by their solution. We will see that the high benefit of this approach is that, some light sources allow an extremely efficient convolution, thus allowing really large sources. Furthermore, we will show how to encode several filtering kernels which will allow us to lift the constraint of the planar receiver.

Inspired by McCool's [McC00] and Chan and Durand's work [CD03], de Boer [dB06] delivers an algorithm that is entirely image-based. Starting from a depth map, discontinuities are detected to derive a light silhouette boundary. This is then extended, not unlike smoothies, using a flood-fill-like approach, encoding a degrading distance to the silhouette. This leads to a buffer similar to the smoothie buffer, but having interior and exterior smoothies. The shadow is then assumed to vary betwen 0 and 1 in this now-larger region. If a point lies in hard shadow, the distance from the silhouette is added; otherwise it is subtracted to derive a continuous variation from black to white. The approach can lead to significant light leaks when an occluder is situated above another. To overcome this issue, an additional test is used. If all samples in a small disc around the view-sample lie in hard shadow, the point is assumed to be completely occluded. This last sampling approach makes the method relatively costly. The extension via an iterative flood-fill also limits performance and several overlappinng "smoothies" can introduce significant artifacts.

7.3 Geometry-Based Approaches

Geometry-based approaches do not share the aliasing and visibility problems of image-based methods. On the other hand, they are usually more computationally expensive.

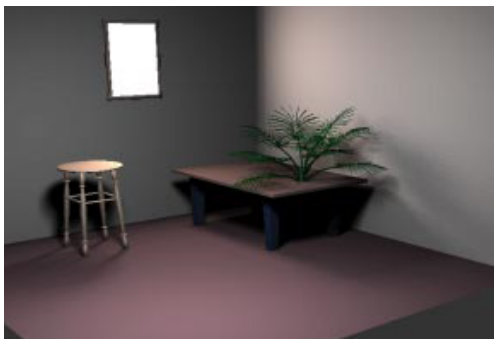


Fig. 10.29 : *Beautiful, but costly (Heckbert and Herf [HH97])*

We already mentioned the possibility of sampling the light source to create soft shadows. For a planar receiver, one can describe this sampling process as several projections. The use of projective geometry allows a simple implementation that is conform with the transformation of graphics hardware, as was pointed out by Blinn [Bli88]. Heckbert and Herf [HH97] showed that this approach allows physically based soft shadows (figure 10.29). In each sampling step, they further include the geometric terms and thus their solution converges toward the correct integral approximation. In their original approach, the accumulation buffer was used, but today, it proves more efficient to rely on alpha blending. Nevertheless, this implies that also the stencil buffer needs to be used in each step because overlapping triangles should only be written once. In consequence, the stencil buffer needs to be cleared after each

pass [Bli]. Fortunately, this is extremely efficient due to the ZClear/Hyper Z technology (Z- (24 bits) and stencil buffer (8 bits) share the same 32-bits of memory). Of course, this method needs to perform one rendering pass per sample and thus scales linearly with an expensive coefficient and in each pass the entire geometry needs to be rendered. In practice, this technique is used at least for standard hard shadows in some applications due to its simplicity. Our method in Chapter 12 avoids several rendering passes and delivers the result for all samples in a single geometry pass.

Chin and Feiner extended their work [CF92] on hard shadows using BSP trees to soft shadows. The approach subdivides surfaces to create a rough segmentation into lit, shadowed, and penumbra areas. Then the penumbra regions are sampled and the result is interpolated on the screen. The method is thus far from real time and the estimation does not perform correct occluder fusion, eliminating almost any umbra regions for objects with smaller triangles.

7.3.1 Penumbra Wedges

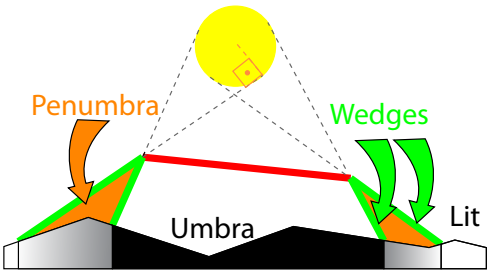


Fig. 10.30 : *Penumbra Wedges are tangent to silhouette edges and the source*

Akenine-Möller and Assarsson [AAM02] launched geometry-based soft shadows into a new fundamental direction and presented a series of soft shadow papers that created a novel trend. The major idea was to adapt the shadow volume separation into umbra and lit regions for soft shadows. The observation was that light-silhouette edges should contain all the information that is needed to derive shadows. Therefore, they start by creating a hard shadow in the scene. Then, wherever a light silhouette is interacting with the shadow, the intial solution is overwritten. To compute the influence region of each light-silhouette edge (determined from the center of the source), the authors introduce *penumbra wedges* (figure 10.30). For each silhouette edge, two tangential points are found on the source that correspond to light-tangent planes containing the edge. The penumbra wedges are then defined by the shadow volumes created from the tangential points on the light. On the sides, two infinite triangles close these tangent faces to a volume. In [AAM02], a first algorithm exploiting this representation was introduced for cards with limited shader support. Consequently, the shadow computation was necessarily simple. The authors decided to linearly vary intensity inside of the penumbra wedges (not unlike [WH03]). To achieve a continuous behavior in this scenerio, care has to be taken for adjacent silhouette edges. The penumbra-wedge side triangles should be shared, and this results in a very involved construction that needs to distinguish many cases. The interested reader is referred to [AAM02] and its amelioration in [AAM04b]. Because these historically important methods can be considered outdated due to their successors, we will not discuss them in more detail here.

More important and still currently of relevance are [AAM03, ADMAM03]. The principle is very similar. Each light silhouette gives rise to a penumbra wedge. The main difference is that they are now constructed independently for each light-silhouette edge or adjacent edge. For each penumbra wedge, shadows are no longer determined by interpolation; instead, a fragment program is executed on the contained view-samples. Each view-sample projects the silhouette onto the light source and computes its blocking contribution. This degradation of penumbra

wedges to simple *markers* of the transitional shadow region allows us to simplify construction substantially because it can be very coarse, they are no longer used to directly interpolate shading. Many of the complex cases (e.g., (almost) alignment with the source) can be circumvented by shifting the wedges' vertices. The construction is as follows: both silhouette vertices are lifted to the same height (the nearest distance to the light), and the front and back quads are constructed as before. The sides, are based on tangential planes to the light, that contain the edge extremity and the cross product of the edge and the vector connecting the extremity with the light's center. The resulting volume is a conservative bound for the penumbra region generated by the light silhouette edge. Nevertheless, their determination is performed from the light's center, which is an approximation and can even result in temporal incoherence when an edge is suddenly becoming a silhouette.

The next step is to compute the actual shadow intensity. This works almost like the approach for computing the area of a polygonal closed shape: choose an arbitrary reference point, and sum up the *signed* areas of all triangles defined by edges and the reference point. The nice observation is that, as long as the edges have a consistent orientation, the final sum can be computed by treating the edges independently and accumulating the result. There are, nevertheless, some problems with this approach. We only want to compute the area inside the light source, and thus edges need to be clamped. This proves rather complicated when using the area described by triangles as above. It becomes much simpler if, instead, one sums the opposites, meaning the sector (described by the two edge extremities and the center) minus the triangle (Figure 10.31 shows an example of the process). The solution works because the infinite area is clamped to the source resulting in a finite value. The edge orientation is chosen according to the view-sample and the light's center. For this, each penumbra wedge is virtually divided into an inner- our outer half-wedge by a hard shadow quad created from the light's center. This classification is equivalent to whether the center sample is in the "blocked" halfspace or not. The orientation of the edge is thus chosen accordingly to either include or exclude this sample. This is also necessary to add-in the umbra region not touched by the penumbra wedges. To compute the covered light area efficiently, a 4D texture can be derived in a precomputation; 4D because it is queried by a pair of 2D endpoints of the projected segments. Then the blocking contribution of each edge boils down to a single lookup. This is especially efficient for textured sources.

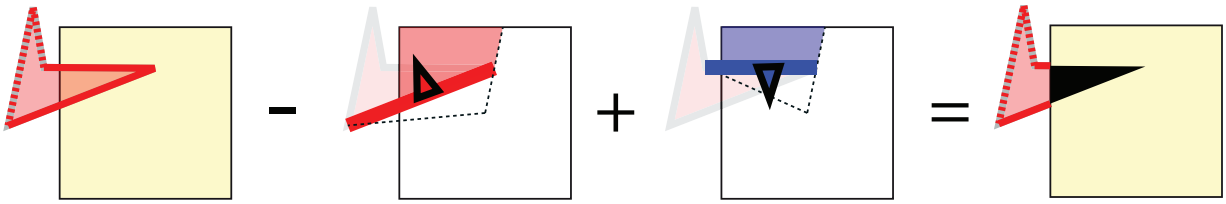


Fig. 10.31 : Computing the blocked area for an occluder at a receiver point

Each silhouette edge is treated separately. The blocking contribution of an edge is the intersection of the source with the sector described with the center of the source and excluding the triangle described by the edge and the center. Depending on the edge's orientation, the area is subtracted or added. This leads to an integration of the surface area once all source intersecting edges are processed.

Some shortcomings of this solution are that occluders are necessarily additively combined, otherwise an expensive method is needed where the shadow information for each occluder is derived separately [ADMAM03]. Unfortunately, this can lead to a strong umbra overestimation, as shown

in Figure 10.9. On the other hand, for non-overlapping silhouettes the method derives an accurate solution. The main reason it has not yet been used in practice is the cost of the approach. It inherits the deficits of shadow volumes: costly silhouette determination and strong overdraw. Furthermore, the computations for penumbra wedges are a little involved.

Several improvements have been presented since. Namely, Lengyel [Len] and Forest et al. [FBP06]. Lengyel presents a way to use orientation and z-tests to optimize the penumbra-wedge rendering. He also passes the plane equations for the inner, outer, and hard shadow plane into the fragment shader to classify the samples based on their plane distances. Further, Forest et al. [FBP06] reduce the previous 250 shader instructions in [AAM03] to 16(!).

Nevertheless, the algorithm remains costly. Even in the most optimized implementation, only 90 Hz are possible with a 762 polygon model, on an NVidia 7800 GTX. For 1524 polygons, the execution time shrinks to 50 Hz before reaching less than 30 Hz with 3048 triangles (timings from [FBP06]).

The quality, on the other hand, can be improved. Assarsson et al. [ADMAM03] mention the idea of cutting the source into smaller patches and to treat them independently. This is somewhat similar to an approximation via several point lights, but here using small patches. The quality can be improved, but the execution time goes up rapidly (even though in [HLHS03] it was claimed that this step has only relatively little impact on performance because the fragment shader was supposed to be the bottleneck). In fact, the silhouette extraction is applied four times and this step governs the computation much earlier. As pointed out in [FBP06], with 17,482 triangles (which can be considered medium-sized), the algorithm does not achieve more than 4 Hz (for a single light patch) and even when adding a second GPU (in SLI mode) no speedup could be obtained. Better silhouette extraction methods exists, but 4 Hz seems like a barrier that is too high to take any time soon.

Forest et al. [FBP06] do provide an amelioration over the several passes (for each region of the light) usually needed to improve the algorithm. They break the light virtually into four regions and compute the blocking contributions during a single pass for all four regions (the key is that the silhouettes are detected for the common corner of all four light regions). Further, they keep track of already-created blocker surfaces by maintaining a bounding box. Whenever a new silhouette is added, the overlap of the bounding boxes is tested and the shadow contribution is decreased by a relative amount, according to the detected intersection. Of course, this is a coarse heuristic, but results in better-looking, lighter shadows.

Finally, Jacobson et al. [JCLP04] present another offspring that claims to derive boundary correct real-time soft shadows. The approach delivers soft shadows for a disc light source. This restriction to a disc leads to a possible acceleration (as also pointed out in [ADMAM03]) that they exploit to compute the light-silhouette edges. A major problem is that their claim of a *correct boundary* (all silhouette edges intervening in the penumbra) seems wrong. They state that the prolongation of these edges to lines cannot intersect the disc source, a property needed during their construction step of the penumbra. Unfortunately, this does not hold. An example is a thin cone. Its side edges intervene in the penumbra and can intersect the light source. The blocking contribution is computed based on the formula in [PSS98]. This relates to the strong assumption that the unoccluded and occluded light regions can be separated with one straight line. This line is found by projecting the silhouette edges on the source and keeping the maximum blocking contribution in the final rendering. Even though this can be faster than the lookup with edges as in other solutions, the quality is often much worse and exhibits the artifacts of max-occluder fusion.

7.3.2 Hybrid Approaches

Smoothies [CD03] can be considered a hybrid approach. It extracts silhouettes and then rasterizes them in a buffer. The original method cannot be considered a soft shadow approach in our strictest sense, due to the fact that it keeps the inner umbra. Multilayered shadow fins [CJW*06] lift this limitation. Instead of creating only outer smoothies, the paper also adds inner smoothies, just like de Boer [dB06], but based on geometric light silhouettes. To solve the issue of light leaking, when one object is on top of another several layers are derived from the scene. The shadow computation is then performed for each layer seperately. Of course, this induces significant extra costs. The authors suggest deriving 1024 layers by placing clipping planes according to the geometry repartition. They abandon this idea for performance reasons and suggest using 16 uniform layers instead. Our voxelization from Chapter 7 has huge potential to be applicable in these kind of scenarios. Occluder fusion is only handled heuristically, by using the minimum blocking contribution for the inner smoothies, and the maximum for the outer smoothies. Both resulting shadow values are added in the end. The fact that silhouettes are combined by a min operation can lead to gradient reversals. Overlapping elements rarely show correctly behaving shadow boundaries. Nevertheless, when using several layers, there is potentially room for more involved combinations (we give one such possibility in Chapter 11). The algorithm is relatively slow due to the silhouette extraction, smoothie rendering, and layering. It runs around 33-14 Hz for 984-9900 triangle models on an ATI Radeon 9800Pro.

7.3.3 Precomputations

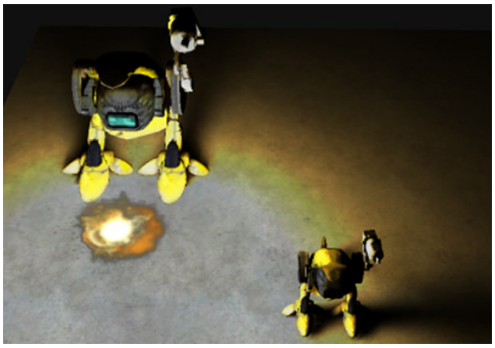


Fig. 10.32 : *A frame from an explosion (Courtesy of Zhou et al. [ZHL*05])*

We have already seen the possibility of precomputing shadows in the context of depth map collections. In this short section, we will focus on other forms of precomputation. As our contributions on shadows mostly avoid supplementary knowledge about the scene, we will only quickly review this topic.

Zhou et al. [ZHL*05] follow a very different strategy: occlusion and emission (for sources) is precalculated per object, and then used dynamically. This is done by creating cube maps around the object. At run-time, for each vertex v in the scene, the illumination is derived sequentially. First, a cube map at this point is initialized that encodes local visibility and BRDF terms. Then all other elements of the scene are sorted according to distance and added sequentially. If it is a blocker, its blocking contribution is looked up from the occlusion field and then rasterized in the map around v. If it is a light source, the corresponding emission field is evaluated against the current cube map and accumulated. To assure efficient storage and evaluation, these occlusion fields are compressed using spherical harmonics for low-frequency representations and wavelets for high-frequency solutions (the input is usually a resolution of 32^2 per cube face and around 16 cubes for highest quality). Wavelets are costly to rotate, and consequently, objects with this representation can only translate. In general, the shadows need to be very smooth to achieve a convincing look. No objects allow deformations and the triple and double products to accumulate blocking and illumination contributions need

important computation time (in 2006, only around 6 objects could be used in real time on a standard machine [RWS*06]). This is also reflected in the storage cost: for an explosion, a six keyframe animation can result in 400 MB (figure 10.32).

Fig. 10.33 : *Approximation via Spheres [RWS*06]*

Following work improved upon many of the shortcomings. In [RWS*06], the computations and storage space have been decreased substantially and deforming objects become possible. The key is to represent elements by a set of spheres. The accumulation becomes much simpler due to this representation. Furthermore, by working in log space accumulating blocking contributions becomes less expensive. The resulting method achieves real-time environmental and local-area light shadows. Nevertheless, it is a necessity that soft shadows are very smooth because the spheres as blockers are a rather coarse approximation (Figure 10.33 shows a high level in the hierarchy). They further need to be precomputed. The shadow evaluation, though more efficient, remains costly. The number of characters in a scene is still quite limited (usually 10-20 objects) and the shapes should not be too detailed.

7.3.4 Offline Solutions

Soft shadow volumes have also been used successfully in the context of offline rendering using ray tracing. Again, this is not in the center of interest for our work and thus we will only quickly review some methods. Laine et al. [LAA*05] showed that the accumulation of edges is nothing but an integration step. They use a map of counters instead of only an occlusion value. After accumulation, the result indicates, up to some offset, how many surfaces are intersected by a ray from the view-sample to each light sample. As a result, only one final ray needs to be shot per view-sample to derive the correct values for all light samples. To illustrate the necessity of this last ray, imagine that not a single penumbra wedge interacted with a view-sample. This does not necessarily mean that the sample cannot lie in shadow; for instance, it can lie deep in the umbra region. To find silhouette edges related to view-samples, they use a hierarchical hemicube-like structure. Then each view-sample finds potential silhouette edges from this structure and then tests which ones of these are actual silhouettes that will be integrated in the process.

In Forest et al.'s most recent work [FBP08], which was concurrently published to our approach in Chapter 13, they present an algorithm that is basically an adaptation of [LAA*05] for the GPU. No hiearchical structures are used, instead penumbra wedges are directly involved to determine the view-samples with which the light-silhouette edges interact. The counters are packed into bits of (currently) floating point numbers, allowing to maintain several ones in a single variable. Nevertheless, care has to be taken to ensure that there is no overflow because it would pollute the neighboring counters. Like our approach [ED07b], lookup tables are used to evaluate the result, but imprecisions can lead to stronger artifacts in this case. The solution achieves interactive performance if the number of light samples is low (typically 16) because of the necessity to maintain counters. With respect to [LAA*05], the algorithm provides a smart GPU adaptation, by providing the reference-ray count based on a shadow volume approach. It is unclear from the article how the silhouette edges are extracted, but a discussion with the authors reveiled that it is done solely from the center of the source, which is inaccurate and can be temporally

incoherent, as we pointed out in our work on accurate visibility [ED07b]. The timings given in the paper do thus not correspond to the computation time of accurate shadows and if all potential silhouette edges were used, some of them would need to be constructed twice with differing orientation or involve a supplementary test where information about the adjacent geometry is needed (compare [SWK07] for a similar situation with hard shadows). The accurate method is thus substantially slower. Unfortunately, text does not discuss this issue, nor does it show scenes where this has an effect, e.g., a large source. For accurate computations and more light samples, the method is thus quickly pushed towards a non-interactive computation.

The work by Lethinen et al. [LLA06] improved upon [LAA*05], they use an important insight, which was that locally clamping penumbra wedges based on the adjacent triangles can significantly reduce their size. Futher, they adapted the hierarchical structures and use a BSP Tree to recover the potential light-silhouette edges that are also silhouette edges for the view-sample in question.

Another approach that relates to ray tracing is Hart et al.'s lazy visibility evaluation [HDG99]. Here, each view-sample shoots a small set of rays to find blockers. These blockers are then tested to determine whether they influence adjacent pixels by employing a flood-fill-like strategy. This combines, somewhat, ray tracing (find triangle) and rasterization (propagate information and exploit coherence).

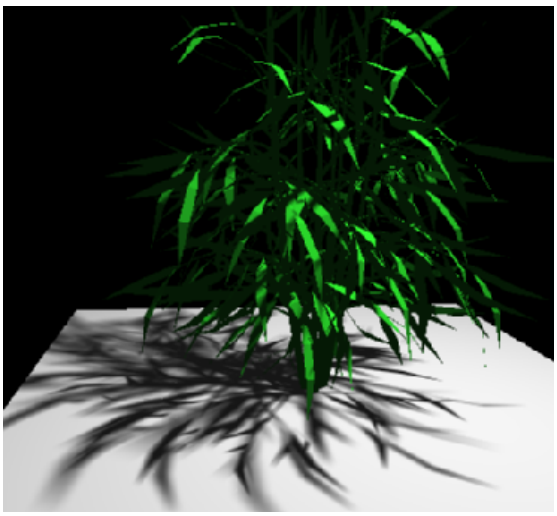


Fig. 10.34 : *Plant With Soft Shadows Overbeck et al. [ORM07]*

In a similar way, this coherence has been exploited by Overbeck et al. [ORM07] in their beam-tracer. Instead of shooting rays, windows are subdivided whenever encountering a discontinuity. This offline tracer is very efficient and highly accurate. It has comparable speed to highly optimized ray tracers, but provides all the benefits of beam-tracing, namely accurate coverage and blocking values. They report very impressive timings on several scenes. Even though considered exact, the results show some artifacts (but this might be an implementation issue or precision problem). Nevertheless, it currently should be considered the state of the art in efficient offline shadow computation.

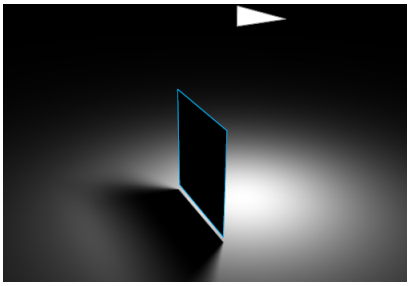


Fig. 10.35 : *Exact Shadows (Stark et al. [SCLR99])*

Stark et al. [SCLR99] provide a completely accurate solution but only for simple configurations (e.g., see Figure 10.35). They show that it is possible to describe shadows for a convex occluder directly and accurately via splines. The method is also interesting as it discusses how to find a boundary for the penumbra region in the presence of a rectangular light source.

Finally, the visibility complex/skeleton [DDP96] has by definition the information about any visibility event in the scene. It has thus the potential to give exact shadow information. Unfortunately, the creation and evaluation is expensive. Aveneau and Mora provide a complete and accurate solution using CSG

in Pluecker space [AM05]. Its exactitude makes this approach very expensive, but the results are precise. This is a must in some applications, for example, for sound propagation.

Summary:

Soft shadows are more pleasing to the eye than other shadows. They increase realism in scenes but are much more difficult to compute. The main directions in this field included penumbra wedges, which create shadow-volume-like primitives to bound the penumbra region. This delivers very beautiful and alias-free shadows for simple objects. Incorrect occluder fusion, on the other hand, can result in very disturbing artifacts. The cost of creating shadow volume primitives makes the methods rather costly and, even on the latest hardware, acceptable run-times are difficult to achieve even for average-sized scenes.

Image-based solutions, on the other hand, provide fast feedback. The varying interpretations of the shadow map allow us to eliminate many of the aliasing artifacts. The shadows look good for smaller sources, but can become less convincing for larger lights because in many cases only a single depth map is used. One such depth layer contains insufficient information about the scene. One possibility is depth peeling, but this implies that one pass is needed per layer, which can quickly outweigh the performance advantage of using image-based approaches.

8 Discussion and Outlook

In this overview, we classified many potential problems. Image-based approaches are much faster, but can result in artifacts. Geometry-based approaches are costly if extra primitives need to be created or extracted. As a consequence, we will discuss here how our work can be situated in this context.

In Chapter 11, we will present an algorithm that is highly efficient (as of 2008, our solution is currently still the fastest local soft shadow algorithm) and allows us to treat geometry that would be hidden for a simple depth map by deriving a layered representation of the scene. A depth map only captures the first visible layer, which is often insufficient to create convincing soft shadows because visibility differs for each light sample. The layers make the algorithm more robust with respect to large sources than a single depth map, and performance is independent of the light's size. This was our main goal: to achieve fast shadows for large local sources in fully dynamic scenes. The major drawback of the method is that the solution delivers only plausible, not accurate shadows. There can be situations in which the result differs substantially from an accurate shadow evaluation. Further, the scenes can only be of limited extent because of the layering process.

In our work on accurate visibility sampling in Chapter 12, we lose much of the performance benefits from pure image-based approaches. On the other hand, in many cases it would still be faster than extracting all depth layers of a scene. By traiting geometry, not only visible elements from the light's center are involved. Our approach relies on the entire geometry to

obtain accurate visibility, but performs computations in image space using rasterization. The main insight is that we can encode visibility using bitmasks that allow for an accurate occluder fusion, a major problem in the soft shadow literature. We present this solution, which is capable of evaluating visibility between two sampled regions in real time. This process is encoded in such a way that the evaluation per triangle is very low cost. The resulting algorithm is applicable beyond shadows.

In an extension of this work in Chapter 13, we describe how to lift the constraint of two patches and deliver accurately sampled soft shadows at interactive rates for general scenes and even volumetric sources. We also address the issue of discretization which usually arises when relying on sampled solutions. This is difficult even for hard shadows. Several approaches suggested reparameterizations to increase shadow map resolution for a given viewpoint. Rarely can these measures assure pixel correct shadows. We discuss our solution to this problem. Arguably, not all situations need pixel-accurate shadows and approaches such as [GW07a] seem to provide sufficient quality. On the other hand, our method delivers pixel-accurate solutions for soft shadows as well, and runs entirely on the GPU.

We developed contributions in two very opposing directions: very fast approximate soft shadows, and highly accurate visibility sampling. Futher, we underlined that, even though the result for the latter is cast in the soft shadow theme, potential applications are many fold. One is visibility-assisted level design, which we present in Section 5 of Chapter 12. It integrates into the context of this dissertation, as it is a means to overcome complexity issues in a preprocess, directly during the design of a model.

"It is the nature of all greatness not to be exact."

Edmund Burke

Chapter 11

Occlusion Textures for Plausible Soft Shadows

Fig. 11.1 : *Our Soft Shadows in a Complex Scene*

This chapter presents a plausible soft shadow algorithm for complex dynamic scenes with rectangular light sources. The result is not accurate in general, in the sense of the visibility integral, but can be in special cases. It also often compares well to reference solutions. Our method makes use of the observation made by Soler and Sillion [SS98] that the integral becomes a convolution for an aligned planar source, caster, and receiver. We use this result to estimate the occlusion at each point of the scene using pre-filtered *occlusion textures*. These are obtained by slicing the scene and storing information about the presence of matter in color channels, not unlike our voxelization in Chapter 7. This dynamically extracted information is then transformed into a hierarchically filtered representation that allows us to rapidly query *blocking contributions*. To extend the result to general scenes we rely on a probabilistic heuristic. The algorithm is very fast and relatively easy to implement, which makes the technique appealing.

Several useful properties can be pointed out:

- shadows are plausible, smooth, continuous, and account for real penumbrae (not just extended umbrae);
- performance is independent of the light's and penumbra's size;
- the scene is only involved once, during a cheap rendering step, which makes the algorithm mostly independent of the scene's complexity, and it integrates well with various rendering paradigms (e.g., vertex shaders, point- and image-based rendering);
- No *a priori* information is needed, and there is no caster/receiver separation.

The major drawbacks are that the light's shape needs to obey some symmetry limitations, and our occlusion retrieval is working best for scenes with limited size, otherwise discretization artifacts

can occur. The method is well suited for indoor environments and performance-crucial situations. It currently seems to be the fastest soft shadow algorithm for local light sources.

Publication notice: *The content of this chapter represents a work with Xavier Décoret and led to a publication at the* ***SIBGRAPI*** *conference 2006 [ED06c]. An extended version was accepted and published in* ***Computer Graphics Forum*** *[ED08a].*

1 Our Approach

Let us quickly recall the visibility integral equation for a point P and a rectangular light $\mathcal{S}$ in the presence of an occluder O:

$$I(P) := \int_{\mathcal{S}} v_P(S) dS, \tag{11.1}$$

where v_P is the visibility function for P defined by:

$$v_P : S \in \mathcal{S} \rightarrow 1 \text{ if } [P,S] \cap O = \emptyset \text{ else } 0 \tag{11.2}$$

As seen before, this function is generally complex and three dimensional. A GPU-friendly approximation is presented in the following sections.

1.1 Planar Occluder

First, we consider a single planar occluder parallel to a rectangular light source. It is fully described by a supporting plane Π_O and a characteristic function δ in that plane:

$$\delta : \Pi_O \mapsto \{0,1\}, Q \rightarrow 1 \text{ if } Q \in O \text{ else } 0 \tag{11.3}$$

Consider the frustum of a point P and the light. There is a bijection between the region where the frustum intersects the occluder plane and the light rays passing through P. The integral of $1-\delta$ over this region, normalized by the region's size, thus gives the shadow intensity at P (Fig. 11.2).

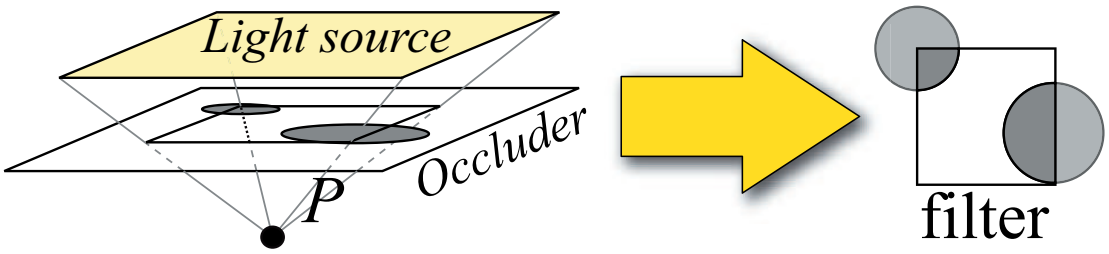


Fig. 11.2 : Visibility as a box filter response

For a rectangular light and a flat-aligned occluder, for a point in the scene the visibility integral is a box filter response.

This region is a rectangle because the light is rectangular and parallel to the occluder. Its size depends on the ratio of distances of P to the light and the occluder plane:

$$s(P) := \frac{d(P,\Pi_O)}{d(P,\Pi_S)} \times size(\mathcal{S}) \tag{11.4}$$

This derivation presents a particular case of the result by Soler et al. [SS98], who showed that for an aligned planar light source, planar occluder and planar receiver, visibility corresponds to a convolution with an appropriately scaled source. In our special case the integral can be computed by filtering $1-\delta$ with a box filter of size $s(P)$. We encode $1-\delta$ as a an *occlusion texture* and process it , as described in the next section. We can then shade a point P by simply computing $s(P)$ using eq. (11.4) and performing a lookup of the appropriately filtered result.

1.2 Accelerated Box Filtering

We investigated three approaches to filter an occlusion texture with a rectangular kernel: Mipmapping, NBuffers and Summed Area Tables.

Mipmapping was originally introduced to reduce aliasing of minified textures [Wil83]. It linearly interpolates between dyadically downsampled versions of a texture. It was a natural candidate because it is widely supported by GPUs. Anisotropic filtering even allows rectangular kernels. Shadows vary smoothly thanks to linear interpolation (Fig.11.3, top). However, it suffers from blocky artifacts that become particularly noticeable when the scene is animated. The dyadic downsampling may combine adjacent texels only at very high levels. Meaning that a slight shift of the filtered function can lead to large variations in the mipmap pyramid.

To alleviate this problem, Décoret introduced the NBuffers [Déc05], which make prefiltering with continuously placed kernels possible. Originally, they were used with a max filter and applied in the context of geometry culling. We use them to compute the mean of neighboring pixels. Each texel, of a level l, holds the *normalized* response of a box filter with a kernel size of $2^l \times 2^l$. Intermediate kernel sizes are approximated via linear interpolation. This does not lead to the exact filtered function, but it significantly reduces the blocky artifacts (Fig.11.3 bottom). The dyadic construction (using four lookups per level) is very efficient.

Summed Area Tables (SAT) [Cro84] allow us to compute the exact filter response for a rectangular kernel. Although an efficient GPU implementation exists [HSC*05], this approach still suffers from several limitations. To avoid precision artifacts, 32-bit textures are required (shifting the values, as suggested by the authors, is not useful in our context, due to the binary nature of occlusion textures). Also linear interpolation for such textures is currently not supported (this is why texels are noticeable on the close-up of Fig.11.3). Moreover, creation and transfer increase the bandwidth. The four texture lookups, needed to get the filter's response, further slow down the approach, when NBuffers require only one. Finally, extra computations are required because the normalization by the kernel size cannot be embedded, contrary to NBuffers. However, the filtering is exact.

We tried the three approaches (Fig. 11.3). It seems that the best trade-off between performance and quality is NBuffers, though SAT may become the preferred solution on future hardware. In particular, a single texture is more cache-friendly than multiple textures as is the case for NBuffers. Currently, SATs are too slow for our purposes, even though our implementation leads to approximately the same timings as in [HSC*05]. The results on a GeForce 6800 Ultra are summarized in table 1.2

resolution	Mipmap (top)	SAT (middle)	NBuffers (bottom)
256×256	< 1 ms	5.1 ms	< 1 ms
512×512	< 1 ms	21.9 ms	1.7 ms
1024×1024	< 1 ms	96.8 ms	7.3 ms

Fig. 11.3 : Comparison of different filtering methods

MipMapping (top) results in many artifacts, FSAT (middle) gives an accurate result, but is costly due to 32-bit textures, and finally NBuffers currently result in the best trade-off (bottom).

1.3 Multiple Planar Occluders

We now consider multiple planar occluders. The shadows caused by each occluder can be computed independently as before. However, it is difficult to combine these shadows. The truth lies between the sum and the maximum of the occluders' contributions (see Section 3). Intuitively, two occluders can cover disjoint or overlapping parts of the source.

Several attempts to solve this problem have been proposed. In [SS98], the use of the mean value between these two extremal situations is suggested. Assarson et al. [AAM02, AAM03, ADMAM03] are inherently bound to combine contributions additively because they constantly add and subtract occlusion contributions based on edges. A different combination method (still not exact) implies a costly clearing step after accumulating the contribution of a single silhouette loop (see [ADMAM03]). Detecting these loops loads more work on the CPU. Arvo et al. [AHT04] need to keep track of the texel in a shadow map responsible for occlusion during the flood-fill process. It is necessary to make a choice when combining the occlusion for two such texels, and they select the one with maximum occlusion.

Additive approaches tend to saturate quickly (occlusion values exceed 100%). The umbra is thus overestimated: shadows look too dark and create unrealistic shadow gradients. Taking maximum values gives more visually appealing results, but tends to create gradient reversals and lead to shadows that seem too bright (in particular if the occluders are rather unstructured, like the foliage of a tree). The average does not seem to be a good choice, because the maximum only takes a single occluder into account, whereas the sum involves all occluders. The ranges of these two values differ too much to be meaningfully combinable.

Our suggestion is based on the observation that the probability of a ray, from P to $\mathcal{S}$, being blocked by the considered planar occluder is exactly $1-V(P)$. V denoting the shadow intensity function given by eq.(11.1). For several occluders with an independent uniform distribution of occlusion, the probability that a ray is not blocked by the union of the occluders is the product of the probabilities. In consequence, we suggest accumulating the shadow intensities of several occluders using:

$$I_{1,\ldots,n}(P) := \prod_{k=1}^{n}(1-I_k(P)) \tag{11.5}$$

This formula inherits the advantages of the sum. Namely, if an occluder does not block any ray ($I_k(P)=0$), the result is not modified and if it blocks all rays ($I_k(P)=1$), the source is declared invisible. In contrast to the maximum (Fig. 11.4), it does combine all contributions instead of selecting just one.

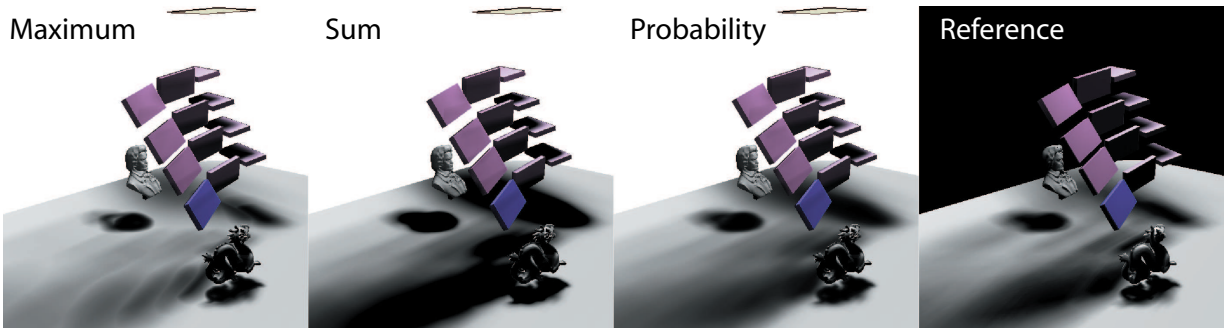


Fig. 11.4 : Comparison of the Max, Sum, and Probabilistic Approach With a Reference

The image illustrates the importance of occluder fusion. Taking the maximum (left) can introduce gradient reversals, and simple summation (middle left) leads to overly dark shadows. The probability-based combination (middle right) results in a solution much closer to the reference (right)

1.4 General Case

In the general case, we approximate the caused occlusion with several occlusion textures. We slice a scene parallel to the light source, and project everything inside a slice on its bottom plane (the one furthest from the light source). The binary nature of this information allows the use of approaches such as the one in Chapter 7 to recover many layers at the cost of a single rendering step.

However, more slices imply more texture lookups to compute the combined shadow. The cost of pre-filtering can mostly be neglected (see table 11.3). Currently, four to 16 slices represent a good trade-off between speed and accuracy. To calculate the occlusion texture representation, each slice is represented by one color channel. Multiple render targets (MRT) give the possibility to write into several buffers at the same time and we can directly associate the slices to the correct color channel. This also accelerates the computation of mipmapping, NBuffers, or SAT because four slices can be treated in parallel. This relates to lightweight methods such as [ND05].

The occlusion texture creation is fast and does not interfere with any CPU- or GPU-based animation. Furthermore, it is compatible with any representation that produces a depth, such as point-based rendering, impostors, or ray tracing on GPU.

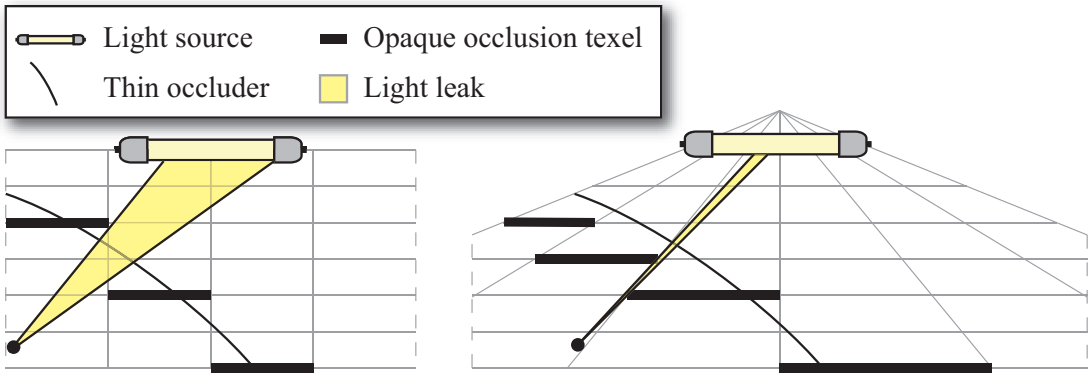


Fig. 11.5 : Orthogonal- (left) create more light leaks than perspective projections (right).

Light leaks are common problems in image-based approaches. This can arise especially in depth-map solutions, where one object lies above another. Nevertheless, our approach is also not immune to this artifact and the problem must be addressed.

[1.0]

The view during this rendering pass is very important as it controls the scene slicing. An orthogonal projection is disadvantageous. First, for an acceptable quality, a higher texture resolution is required because it is more difficult to fit the scene into the camera's frustum. Second, the projection onto slices breaks continuous surfaces into pieces along lines not oriented towards the light's center. Light can easily shine through, where it would actually be blocked by the real surface, causing *light leaks* (Fig.11.5). Perspective projection suffers less from light leaks. Figure (11.5) illustrates this. Nevertheless, if the center of projection (COP) is placed on the light source two problems arise: first, to encompass the scene a large field of view is necessary, increasing texture distorsion; second, during the shadow computations kernels are large and can even jut out of the occlusion textures (Fig.11.6). Interpolation as well as precision issues arise. Placing the COP slightly behind the light (as seen on figure 11.5) at a distance sufficient to englobe the scene, and fitting the near plane to the source leads to much better behavior. Using a projection from a particular COP only affects the way we "x-ray" the scene to approximate occlusion, not the areas where shadows are computed. There is neither a relation to the recovery of a depth map (which would only contain the first surfaces of a scene), nor to the approximation to detect silhouette edges from the center of the light source as in other methods.

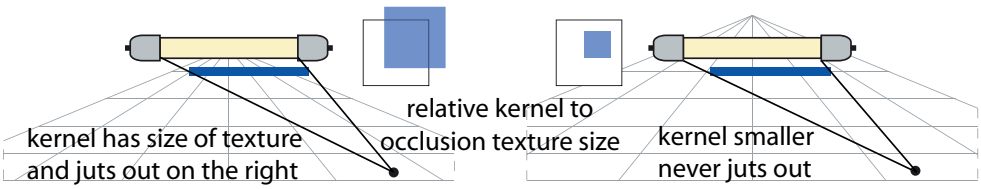


Fig. 11.6 : Offsetting the COP simplifies filtering

When choosing the COP to lie behind the source, the box filter cannot leave the frustum of the light source. This facilitates access and filtering operations.

Light leaks

Perspective projection already limits leaks, but some may still occur. It will become particularly visible in the case of thin geometry, such as the wings of a butterfly (Fig.11.7). For such geometry, we decided to project each occlusion texture on its successor further away from the source. This helps to "close" the discontinuities. This projection is possible at almost no cost: it is sufficient to fill two channels instead of a single color channel during the slice creation. Figure 11.7 shows the effect of the projection. In practice, we did not encounter any leaks using this method although it does probably not handle all situations. It is also useful for animation of thin objects as it improves coherence when geometry changes the slice.

Fig. 11.7 : Light leaks for thin occluders can be fixed

Virtually downprojecting the upper on its neighboring lower layer fixes many of the light leaks.

Shadows are affected by this step; umbrae are slightly overestimated and the shadow gradient differs a little. Consequently, this correction could be applied uniquely for thin objects. Where light leaks are unlikely, it can be deactivated.

Self Shadowing

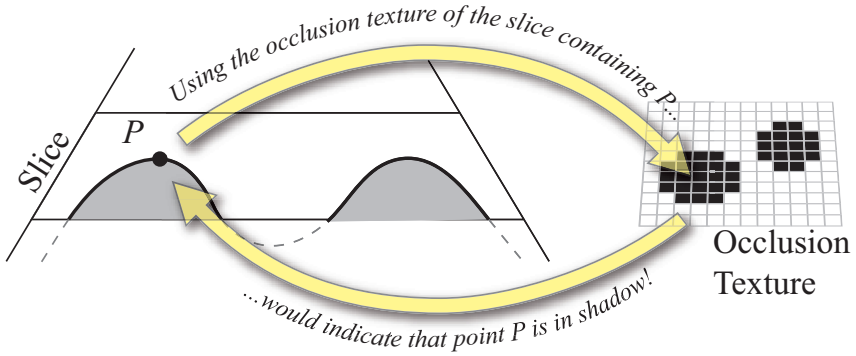


Fig. 11.8 : Auto-shadowing inside a slice

To avoid auto-shadowing we exclude the containing slice from the shadow computation. To avoid visual artifacts a linear blending ensures continuity

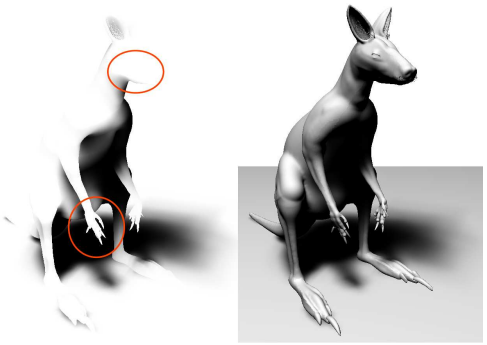


Fig. 11.9 : Missed Shadowing - *Our shadowing (left) may miss close self-occlusions. Diffuse illumination often compensates for this (right).*

There is no shadow caster/receiver distinction in our method. For every point in the scene the shading is computed based on the occlusion textures between it and the light source. The occlusion texture corresponding to the slice that contains the point needs to be omitted. Otherwise the point would be shadowed by its own projection in the occlusion texture (Fig. 11.8). Simply ignoring a slice may cause discontinuities where geometry crosses the clipping planes. Instead, we linearly fade out the slice's contribution based on the distance of the shaded point to the slice's lower clipping plane.

This solution is a very coarse approximation. In practice it often still works well for the following reasons. The occlusion texture actually provides a good approximation for far-away geometry; for nearby elements it is more problematic. Interestingly diffuse illumination often helps to correct this shortcoming. For a watertight object the front-facing faces block light from the back-facing ones. If both fall in the same slice, this effect is missed. However, the diffuse illumination of the back-facing faces is zero and makes them appear dark, as they should (Fig.11.9).

For non-watertight chaotic objects like trees, the diffuse illumination has high frequencies which potentially hide incorrect shadowing. Again, the problem we discuss here only concerns nearby slices.

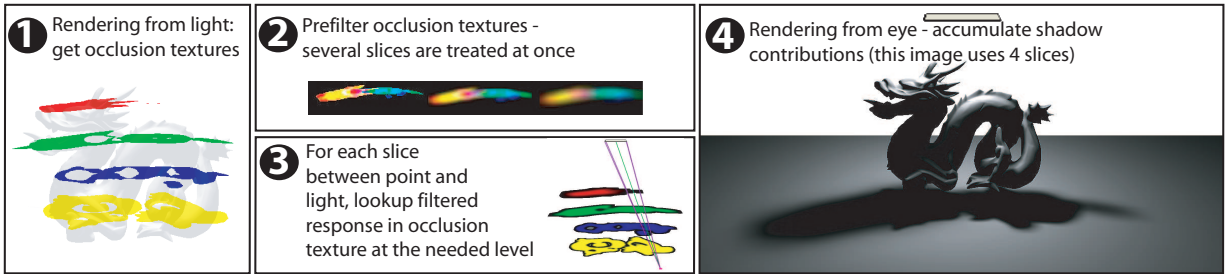


Fig. 11.10 : Overview of our algorithm

Summary:

We have seen that in the special case of a rectangular source and a planar caster, the visibility of the source from a point P is a box filtering process. We investigated several methods to provide a structure that allows fast box filter queries and pointed out an efficient solution based on NBuffers [Déc05]. A heuristic allows us to estimate the visibility in the presence of several layers. Approximating the scene with a set of layers and making sure to avoid artifacts from auto-shading, we obtain a simple and efficient algorithm. There are still several issues concerning the implementation that we solve in the succeeding sections.

1.5 Putting Everything Together

The algorithm is summarized in figure 11.10. Occlusion textures are obtained by cutting the scene in slices. This involves one rendering of the scene from the light's point of view (Sec. 1.4). These occlusion textures are processed and allow us to recover filtered responses for different kernel sizes (Sec. 1.2). In a second step, the scene is rendered from the observer's point of view. At each point P of the scene, the shadow caused by each slice between it and the source is computed. This involves a texture lookup according to the size of the light's projection from P onto the slice. Using the formula (11.5), the shadow contributions are combined. This leads to better results than the maximum or additive approach (Sec. 1.3). The contribution of the slice s_i closest to P is weighted depending on the distance between P and s_i. This results in a smooth inter-slice variation. Finally, illumination based on the material (phong, texture) is combined with the shadow intensity.

Fig. 11.11 : Sunlight: simulating a spherical source

To allow an efficient computation of a circular kernel we use a Gaussian intensity distribution with respect to the center, which allows an efficient filtering.

A modification of our algorithm allows sunlight-like shadows (see figure 11.11). We suppose that the source is at infinity, and consider a constant frustum from each scene point. To capture the spherical shape, we use a Gaussian instead of a box filter. The non-dyadic nature limits the maximum frustum size. In the case of sunlight this is unproblematic as the sun's solid angle is also small in reality.

1.6 Implementation Details

Conceptually, the algorithm is simple (see alg. 1). To shade a fragment, we retrieve the corresponding world point W. Then, for each occlusion texture, we find the kernel position p and size s (implied by W and the light). We deduce the surrounding NBuffer levels l and $l+1$ and lookup the filtered occlusion in these levels. These are then linearly interpolated and accumulated.

In practice, coding this solution is a little more challenging. The first problem is that shaders can only access a fixed number of textures, typically 16. If we use four occlusion textures of 256×256 (thus encoding 16 slices in the RGBA channels), we need $8 = \log_2(256)$ levels of NBuffers. Even on a simple configuration like this, $4 * 8 = 24$ different textures are used. To solve this issue we pack the result densely. 16 slices are encoded in four occlusion textures (one slice per color channel). Instead of generating eight NBuffers for each, we generate eight textures and pack in texture i

Algorithm 1 ideal computation of visibility V

```
1: V = 0
2: W = world_coordinate(fragment)
3: for i in occlusion_textures do
4:     p,s = kernel_pos_and_level_for(W,Light,i)
5:     l = ⌊s⌋
6:     v_lo = lookup(p,nbuffers[l])
7:     v_hi = lookup(p,nbuffers[l+1])
8:     v = (1−(s−l))v_lo + (s−l)v_hi
9:     V = accumulate(v,V)
10: end for
```

the NBuffer level i of each occlusion texture (resulting in 8 textures of $4 \times 256 \times 256$, instead of 24.). With a resolution of 2048×2048, we would only need three more NBuffer levels, thus still fitting the 16 textures limitation (and in particular leaving five textures usable for conventional shading).

The second challenge is that on older hardware we cannot index arbitrary textures in a loop. On newer hardware this is possible: NBuffer levels can be arranged in a 3D texture or texture stacks (DX10). A 3D texture would also make the linear interpolation of lines 12-14 immediate and hardware-supported. Rendering in a 3D texture is a documented functionality, but drivers did not implement it until the GeForce 8. Another interesting aspect is that the GeForce 8 supports 32-bit textures. Several occlusion layers could thus be packed in one color channel. This would make the lookups more cache-friendly and improve performance.

1.6.1 DX9 Hardware Implementation

For DX9 hardware, shaders did not allow dynamic access of an array of textures. Lines 6 and 7 thus cannot be translated to shader instructions.

We get around this problem by sequencing the algorithm so that texture arrays are accessed statically. Each slice i should be filtered with a different kernel of size s_i. The key observation is that this s_i is strictly increasing from the slice closest to the shaded point to the slice closest to the light. Thus, we can loop over the NBuffer levels l in order of increasing kernel size s_l, and increment a current slice index i. This index is initialized with the slice closest to the point and is increased every time $s_i < s_l$. Because of packing, this index is a shift of the horizontal texture coordinate used to access the current NBuffer level. This second version (see alg.2) works because the loop at line 5 is a static one, completely determined by the resolution of the occlusion textures.

Line 8 requires a comment. Because we encode four slices in the RGBA channels of each occlusion texture, increasing the current slice index is a bit more tricky than using $i = i + 1$. Luckily, this can be done efficiently using the swizzle operator of shading languages, and other tricks. We use a `float4` for slice index and implement lines 8 and 9 with `i=i.yzwx` and `delta += i.w*packing_offset`. Then, in lines 12 and 13, when we do the lookup, we get back four slices as one RGBA color, and we extract the relevant one by doing a dot product with `i`. Note that

Algorithm 2 practical computation of visibility V

```
1: V = 0
2: W = world_coordinate(fragment)
3: i = index_slice_closest_to(W)
4: p,s = kernel_pos_and_level_for(W,Light,i)
5: δ = i*packing_offset
6: for l in nbuffers_levels do
7:    while s <kernel_size_for_level(l) do
8:       i = i+1
9:       δ += packing_offset
10:      p,s = kernel_pos_and_level_for(W,L,i)
11:   end while
12:   v_lo = lookup(p+δ,nbuffers[l])
13:   v_hi = lookup(p+δ,nbuffers[l+1])
14:   v = (1−(s−l))v_lo + (s−l)v_hi
15:   V = accumulate(v,V)
16: end for
```

this approach is purely arithmetic and no branching is used (also conceptually, it amounts to tests, and can also be implemented using `if/then` constructs).

A couple of other optimizations can be done, but they are not presented here for the sake of conciseness. In particular, packing offset and kernel position/size actually only depends on the distance from the shaded point to the COP. This simplifies and factors several computations. Deferred shading is used to avoid computations on hidden fragments. This works because the algorithm only needs the world position of the fragment, which can be output in a texture in a first pass.

2 Results and Discussion

Our work is similar in spirit to that of Keating and Max [KM99], but the field of application is completely different. Their approach does not aim at real time and targets ray tracing. It introduces the idea of decomposing the scene into layers, but even without averaging several rays, it is still presented in a form that would not allow real-time performance. It uses a convolution with small kernels mostly to avoid noise, and applies it similarly to percentage closer filtering [RSC87]. Instead, we use convolution for acceleration purposes. We presented several solutions to approximate filtering efficiently. Our method thus treats large light sources without penalty. Occlusion textures are efficiently created on the GPU and we avoid any CPU interaction. We combine contributions differently, based on probability, and obtain convincing results without evaluating several sample rays. Of course, ray tracing produces more realistic images.

We believe that the introduction of occlusion textures for shadows is very beneficial. Depth map-based approaches can suffer from visible temporal incoherences for large sources, even in simple situations. Consider a small occluder close to the light: It might not even create an umbra region, but all objects that are hidden in the depth map will not cast any shadow at all. Therefore, whenever an object passes over another one, artifacts can appear in the penumbra

region. Atty et al. [AHL*06] rely on two shadow maps to overcome this problem. Guennebaud et al. [GBP06] do not have this option, because they would need to separate occluder and receiver. Thus, they are restricted to small sources, not only for performance but also for quality concerns. For small light sources, due to a smaller overhead, the technique is preferable, although in this case percentage-closer filtering works as well. Our method has fewer problems with occlusion because we rely on slicing. Nevertheless, slicing might miss nearby occlusions, thus flickering and oscillations may still appear for vertical movements and grazing angles (as shown in the accompanying video).

We implemented our method using Cg 1.5 shading language and OpenGL. To make it possible to compare our algorithm to others, we used the same test system as in [AHL*06](a 2.4 Ghz Pentium 4 with a GeForce 6800 Ultra). Both the scene slicing and the computations of NBuffers are very fast, thus the rendering cost of our method is dominated by the final render pass. Most of the images we show are levels of gray; this is to emphasize the shadows. Our method works seamlessly with textures, and would even benefit from their presence, since texture maps would mask minor shadowing artifacts. Similarly, most of our examples show cast shadows on a flat ground to ease the perception. An arbitrary ground is possible, and we want to emphasize again that there is no caster/receiver distinction in our method.

In practice it could be advantageous to know that some elements of the scene are only receivers. These could then be simply excluded from the slicing. Nevertheless, we did not rely on any such special treatment.

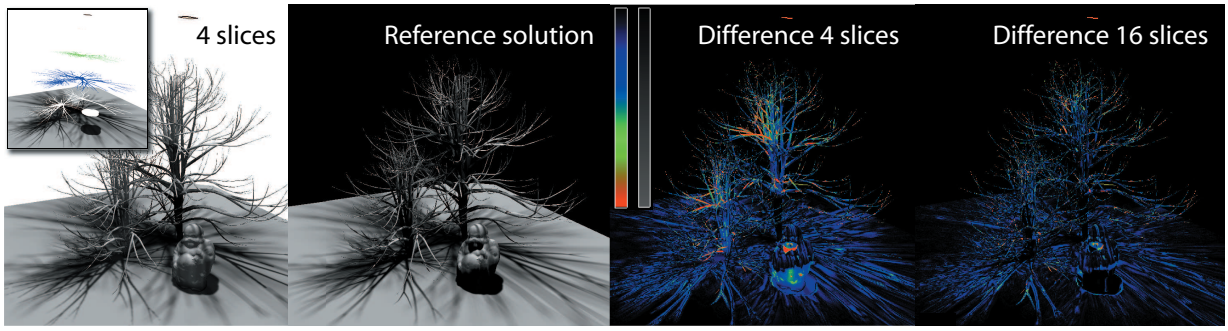


Fig. 11.12 : Quality Comparison for our Shadows

Distant objects are well represented even with few slices. Nearby shadows, on the other hand, benefit from more (e.g., branches).

2.1 Qualitative Analysis

Although our approach is based on a very coarse assumption, the results are surprisingly convincing and correct. We tried different test scenarios to study both, how well the slices capture occlusion, and the influence of texture resolution. We compared our technique to a reference solution based on sampling, and show deviations in false colors.

The tree scene of Fig. 11.12 indicates that, even with 4 slices, we capture important effects. In particular, the shadows cast by the large tree are very accurate on the ground because the further a caster is from a receiving point, the smaller is the error caused by approximation. In other

words, the further away objects are, the more aggressively they could be simplified. This means that if shadows on the ground are the focus, a good choice would be to place more slices close to the ground. Our implementation, however, uses uniformly placed slices because we wanted to make no assumptions about the scenes. The overlapping of branches in the tree scene would severely challenge depth map-based methods. We added a weak ambiant that is only removed by actual cast shadows. This helps to better illustrate where shadows were missed. Diffuse lighting for the tummy of the buddha would be completely black. Figure 11.13 shows an example with a non-planar ground.

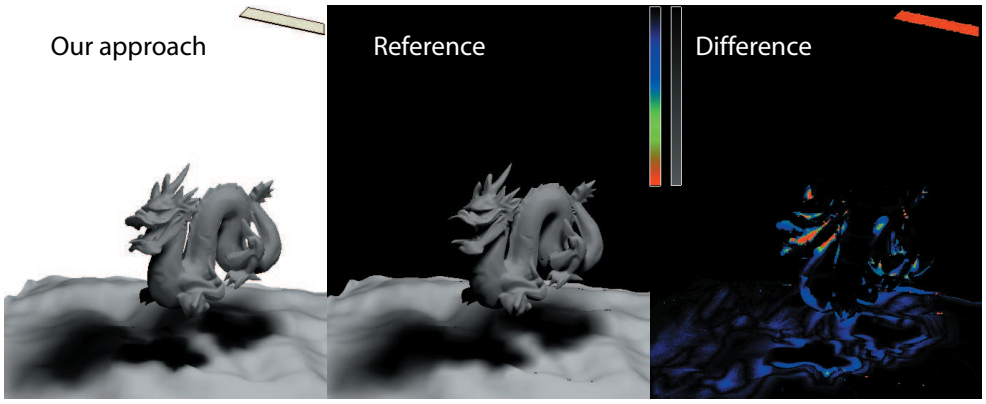


Fig. 11.13 : Non-planar Receiver

Comparison between our approach and a reference solution for a non planar ground.

Even for extreme low-resolution occlusion textures, our shadows are plausible and smooth. Although each single texture is piecewise linear and can look blocky, combining non-aligned textures increases resolution artificially. One can interpret this in terms of frequencies [DHS*05]. Slices can be considered as a decomposition of the shadow on basis functions. The lookup into each slice is done with a distinct filter size and thus represents a separate frequency range. Consequently, low resolutions seem to be sufficient if the light source is large with respect to the scene. Figure 11.14 illustrates this.

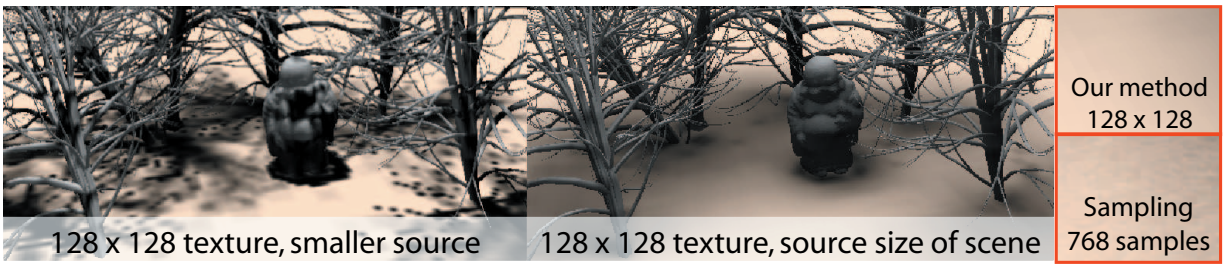


Fig. 11.14 : Quality comparison: Smoothness and Texture Resolution

Low resolution is visible for small lights (left) but acceptable for large ones (right). In the latter case, sampling might even be noisy.

The quality can even beat sampling with many samples because in this case the fine branches add noise when sampling the light accurately. Due to the filtering, our result is smooth, as expected.

In general it is not possible to use low resolution textures. Because the smallest entity is a pixel, the blocking contribution of very fine objects can be overestimated or missed. This problem is common to all image-based methods. Noticeable artifacts can occur during animation (still images are deceiving to judge a method).

2.2 Timings

We compared the performance of our algorithm with the Soft Shadow Volume (SSV) algorithm of Assarson et al. [ADMAM03], and with the Soft Shadow Texture (SST) algorithm of Atty et al. [AHL*06]. We used the Jeep scene from [AHL*06] with a varying number of cars (each is 2032 polygons). Fig.11.15 presents the results. Our method outperforms both competitors. In

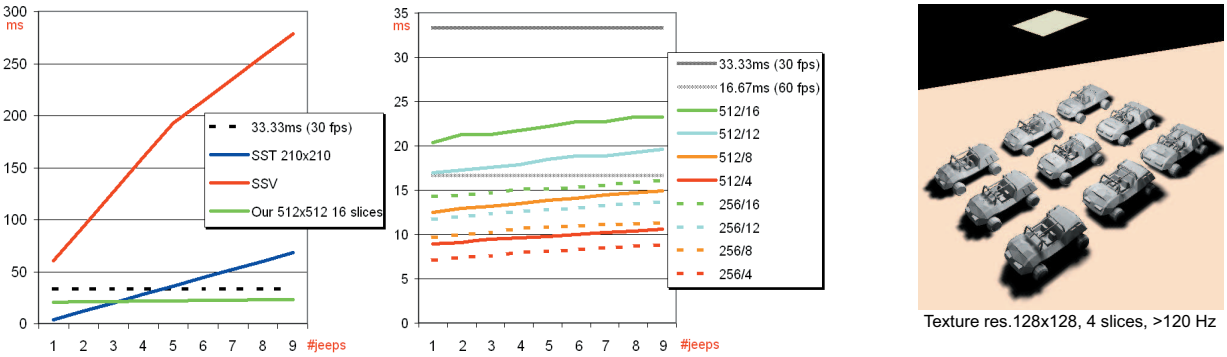


Fig. 11.15 : Performance comparison with other soft shadow methods

Comparison of our method to Soft Shadow Textures (SST) and Soft Shadow Volumes (SSV). We used textures more than 4 times larger than SST and 16 slices, whereas 4 slices of 128×128 *resolution would be enough (right).*

particular, the rendering time is almost independent of the geometry. For SSV, it is clear that adding more geometry increases the amount of work. For SST, the slow-down is caused by the increasing number of shadow-map fragments that need to be reprojected. Thus, although image-based, it is not independent of the scene configuration, as is our method. Note that our tests use 16 slices and high-resolution occlusion textures. Here, only 4 slices and a low resolution still produces nice results and would yield even better comparisons. We largely outperform state-of-the art image-based soft shadow algorithms, and we lift the caster/receivers separation limitation. To emphasize that our method scales well with the number of polygons, we tested a scene with multiple bunnies, each approximately 69k polygons. Figure 11.16 shows that we achieve real time even for challenging scenes of almost 500k polygons, and 16 slices (with 4 slices it runs around 4 times faster). Of course, for a scene with a large depth extent more slices would be necessary thus slowing down our solution, but in these cases many overlapping objects could create problems for other approaches too.

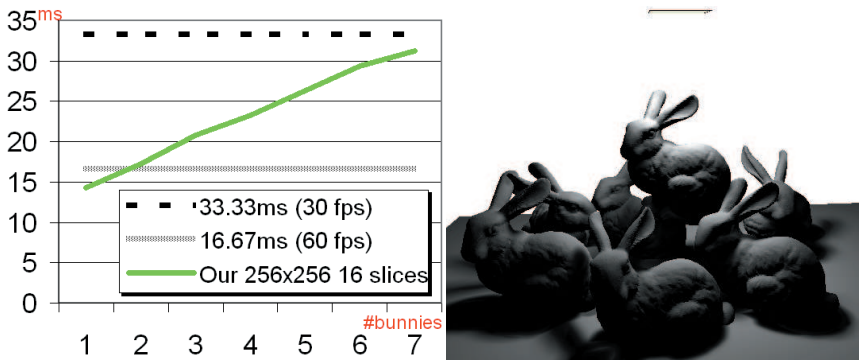


Fig. 11.16 : Performance vs. Complexity

Our method scales well with complexity (each bunny has 69451 triangles, totalling to 486,157 polygons for the seven bunnies)

Summary:

The implementation on older cards is tricky but even on newer ones the presented reformulation leads to a significant speed-up. We further extend the method to treat distant sunlight. We compared the results in quality to reference images and found that in many situations the error remains very acceptable. Furthermore, shadows appear naturally smooth. Nevertheless, the results are only plausible, so the method can fail and the layering assumes a limited depth extent of the scene. On the other hand, the performance of the algorithm is very high and only weakly influenced by the complexity of the geometry.

3 Conclusion and Future Work

We presented a novel image-based soft shadow algorithm that is fast and especially well-adapted to GPU. It requires no precalculation, integrates smoothly with animated scenes, and does not distinguish casters and receivers. Although not physically correct, the resulting shadows are convincing and relatively close to the ground truth: inner and outer penumbrae are handled, and complex inter-object shading is treated through a novel way of combining caster contributions. Although the method is image-based, shadows are smooth, even at low texture resolution. The method is output sensitive, depending only on the amount of shaded points rather than on type or size of the shadows. In particular, very large light sources are naturally treated and it outperforms current state-of-the art algorithms. To our best knowledge, it is the only approach that possesses all these properties. Some artifacts can occur, due to the limited number of slices/resolution and approximated filtering; self-occlusion might fail locally and shadows can flicker for vertical movements as the weighting for the closest slice changes.

As pointed out by Hasenfratz et al. [HLHS03], an unsolved problem is to provide best soft shadows for a given time constraint. Our algorithm gets quite close to this goal. Its run-time is mostly predictable and depends on the number of shadowed pixels in the final output. Quality-based methods [GBP07, SS08] could be directly integrated with our approach.

Currently, we investigate using forward shadow mapping [Zha98]. The idea is to compute shadow values in the light's view and transfer them to the final output. Usually this transfer would be done via splatting, but the slice structure allows us to store the results. Therefore a modified rendering step from the viewpoint would allow each fragment to lookup its shadow value. Such an approach fixes in advance the number of fragments as well as the number of texture lookups, and speed becomes completely controllable via resolution.

Slice placement should be investigated further. Litmaps [Déc05], or CC Shadow volumes [LWGM04] might be used to guide a non-uniform placement. Per-object slicing could address the discontinuities inherent to a scene-based slicing, enforcing "continuous" evolution during animation. As mentioned, more than 16 slices could be generated from the scene (compare Chapter 7). Redistributing the slices into color channels can be done independently of the scene geometry. This would also allow us to do a first scene analysis to better fit slices to the objects. Currently we simply use a uniform distribution. The method recently presented in [LM08] might also be useful.

Reparameterizations [LTYM06, WSP04, SD02] to increase texture resolution locally are a challenging topic, but methods like [GW07c] might integrate more easily.

Finally, hierarchical branching could be interesting, as one lookup gives us information about four slices. On our test hardware, it is currently not advantageous, showing that shader optimization becomes difficult.

"Keep your face to the sunshine and you cannot see the shadow."

Helen Keller

Chapter 12

Visibility Sampling on GPU and Applications

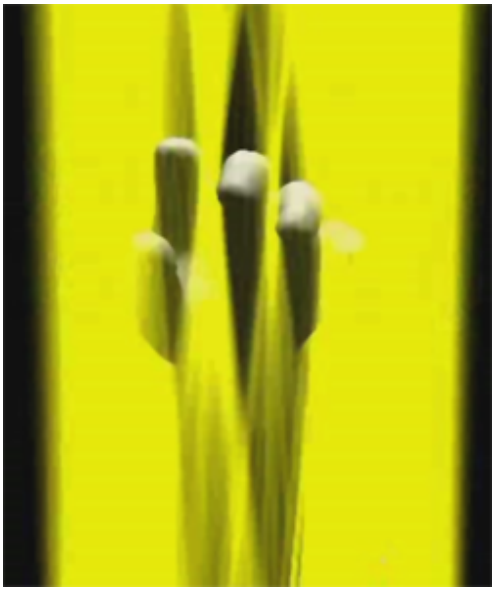


Fig. 12.1 : *Real-Time Visibility Queries*

In the last chapter we aimed at the creation of plausible soft shadows. We presented a highly efficient algorithm, but shadows were based on visibility approximations and were limited to a quantized response not an information of the actual unblocked ray set. This limits the general usage of the approach in other scenarios than soft shadows.

In this chapter we will derive a method that allows us to query the visibility relations of many samples placed on two patches in real time. This allows efficient and accurate soft shadows on a heightfield receiver. Our solution presented here was the first method that achieved this result in real time for more than simple scenes on standard graphics cards.

Information concerning visibility is important far beyond soft shadows. We show that this information can be useful for modeling systems to integrate performance issues already during the creation of scenes.

In the context of our global work this contribution has a twofold status. On the one hand, we show how to efficiently reformulate the visibility tests to allow real-time sampling. On the other hand, we introduce a completely different way of dealing with complexity. Our solution allows us to integrate visibility in the design process of a scene. We believe that this system is only a first step in a direction that allows us to *create* scenes that behave well with respect to the desired computations.

This chapter will also prepare the reader for the extension we will present next. We decided to keep the chapters separate to build a good foundation before going into the details that are necessary to generalize the approach to arbitrary scenes. The extension does not invalidate the solution described here, which is optimized for the described scenario.

Publication notice: *The content of this chapter represents a work with Xavier Décoret and led to a publication at the EUROGRAPHICS conference 2007 [ED07b].*

1 Introduction

We will show how recent GPUs can be used efficiently and conveniently, to sample the visibility between two surfaces, given a set of occluding triangles. We use bitwise arithmetics to evaluate, encode, and combine the samples blocked by each triangle. In particular, the number of operations is almost independent of the number of samples. Our method requires no CPU/GPU transfers; is fully implemented as geometric, vertex and fragment shaders; and thus does not impose modifications on the way the geometry is sent to the graphics card. We then present applications of our method in the context of soft shadows and level design.

2 Previous Work on Visibility Determination

The problem of determining if two regions of space are mutually visible, given some objects in between, appears in many fields [COCSD02]. In computer graphics, it arises for various tasks such as hidden-face removal (finding the first surface visible in a direction), occlusion culling (quickly rejecting parts of a 3D environment that do not contribute to the final image), shadows, and, more generally, illumination computations. The taxonomy of visibility-related algorithms involves different criteria. First, there is the "size" of the regions. Testing the mutual visibility of two points is much easier than testing that of two polygons. What is at stake is the dimensionality of the set of rays joining the two regions. Except for the trivial case of point-point visibility, this set is generally infinite. Typically (e.g., for surface-surface visibility), it has four dimensions which makes it harder, though not impossible [NBG02, HMN05, MAM05], to manipulate. For that reason, many methods reduce the complexity by sampling the two regions, and solving point-point visibility (e.g. computation of the form factor between two polygons in radiosity). This chapter reformulates visibility sampling so that it can benefit from the GPU capabilities.

Of course, sampling yields inaccurate results, fundamentally because visibility cannot be interpolated. You may not see the inside of a room from the ends of a corridor, yet fully see it when you stand in the middle, right in front of the open door. Though not ***exact***, such methods can be made ***conservative***–objects are never classified as not seing each other when they actually do–by various means such as erosion [DDS03] or extended projections [DDTP00]. One key point here is that exactness is not required in all applications of visibility. Typically, soft-shadow computations can afford very crude approximations. In occlusion culling, if the pixel error caused by an approximation is small, it can be very beneficial to perform so-called ***aggressive*** culling [NB04]. Of course, the more samples we have, the lower the error and the trade off will be between efficiency (fewer samples) and accuracy (more samples). The method presented here can treat many samples at once, and can decorrelate the sampling of the two regions, which strongly reduces aliasing.

The classification of visibility algorithms also depends on the type of request. One may either test whether the regions are visible, quantify this value, or even determine where they are visible. For example, occlusion queries not only return whether an object is hidden by an occlusion map, which can be used for occlusion culling [BWPP04], but also indicate how much it is visible, which can be used for LOD selection [ASVNB00]. The problem here is to *represent* which parts of the regions are occluded by a given occluder (i.e., which set of rays it blocks), and then *combine* these

occlusions. It is a difficult task known as *penumbrae fusion* [WWS00]. Explicitly maintained representations such as [MAM05] are very costly because the space of rays is typically a 4D variety in a 5D space. In [HMN05], it is used to drive the selection of good occluders. Leyvand et al. [LSCO03] perform ray-space factorization to alleviate the high dimensionality and benefit from hardware. In guided visibility sampling [WWZ*06], ray mutations are used to sample visibility where it is most relevant. The method is very fast and uses little memory. As we have seen, many soft shadow algorithms combine percentages of occlusion , instead of the occlusion. This can yield visually acceptable shadows, and is much easier to compute, but can also fail and this quantized amount did previously not provide any information about the actual visible ray set other than its approximate size. In this chapter, we show how bitwise arithmetic can be used to encode and combine occlusion correctly, in a GPU-friendly way.

Another concern is a method's simplicity. To our knowledge, very few of the numerous visibility algorithms are implemented in commercial applications, except for frustum culling and cells and portals [AM04]. We believe this is because many methods impose high constraints on data representation (e.g., triangles, adjacency information, additional data per triangle, etc.). Our method works fully on the GPU and needs no knowledge about the scene representation. The same code as for rendering can be used, since everything is implemented as geometry-, vertex-, and fragment shaders. It can handle static or dynamic scenes.

The remainder of the chapter is organized as follows. We present first the principle of our approach, together with considerations on its rationale. We then present two applications, one for soft shadows, and one for level design in games. We finally use the results obtained in these applications to draw lessons about the proposed method.

Summary:

There are many applications for visibility determination, but exact methods are usually very costly. Fortunately, in many situations, sampled visibility is sufficient, but even this is not cheap. Further, many approaches (especially for soft shadows) resort to approximations that lose the actual set of unblocked rays and only provide an occlusion percentage. Visibility methods that provide more information are usually much slower, difficult to implement and/or involve complicated data structures.

3 Our Approach

3.1 Principle

We consider two rectangular patches, a source $\mathcal{S}$ and a receiver $\mathcal{R}$. On each are sample points $S_i, i \in [0,s[$ and $R_j, j \in [0,r[$ respectively. In between are occluding triangles $T_k, k \in [0,t[$. For each R_j, we want to compute the set:

$$\mathcal{B}_j := \left\{ \mathcal{S}_i \text{ such as } \exists k \ [S_i, R_j] \cap T_k \neq \emptyset \right\} \tag{12.1}$$

It is the set of source samples that are not visible from receiver sample R_j. If $\mathcal{S}$ represents a light source, $|\mathcal{B}_j|$ gives the amount of blocked light that is the penumbrae intensity. Computing

$\mathcal{B}_j$ fundamentally requires a triple loop with an inner intersection test. Formally this can be expressed as:

$$\forall i \quad \forall j \quad \forall k \quad [S_i, R_j] \cap T_k \stackrel{?}{\neq} \emptyset \tag{12.2}$$

The commutativity of the "for each" operators allows for different ways (six at most, 3 by eliminating the symmetry) of organizing the computations. For S_i and T_k fixed, finding the R_js that pass the intersection test amounts to a projection from S_i of T_k onto the receiver's plane, and testing which R_j falls inside this projection. This can be done using the projection/rasterization capabilities of graphics cards. In [HPH97], an off-center perspective view (the projection plane is not perpendicular to the optical axis) of the occluders is rendered from each source sample to obtain a black and white occlusion mask. These views are accumulated to obtain an image that represents exactly $|\mathcal{B}_j|$. This approach has two limitations. First, it only computes the cardinal of the sets, not the sets and is thus limited to shadow-like requests. Second, it requires as many renderings as there are source samples. For 1024 samples, this can drastically impact performance as shown in Section 6. Recently, Laine and Aila [LA05] showed that it can be efficient to do all computations involving a particular triangle while it is at hand instead of traversing all triangles again and again. In that spirit, we propose the following approach:

1. traverse all triangles;
2. traverse all receiver samples that are potentially affected by the current triangle, which we call the *triangle's influence region*;
3. find source samples that are hidden by the current triangle from the current receiver sample;
4. combine source samples with those hidden by previous triangles.

This approach requires a *single* rendering pass, using geometry-, vertex-, and fragment shaders. Section 3.2 details how to touch receiving points affected by a triangle. Section 3.3 shows how to backproject the triangle from these points. Section 3.4 explains how to deduce hidden light samples of a point on the receiver, and Section 3.5 shows how to combine the hidden samples for all triangles. For the purposes of clarity, we will use a *fixed* set of 32 light samples. Section 3.6 will show how to improve the sampling.

3.2 Triangle's Influence Region

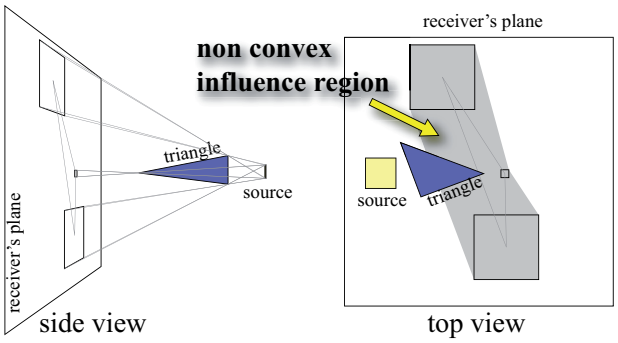


Fig. 12.2 : *From the projected vertices, better constructs than a convex hull are possible*

For a single triangle, the region where occlusion can occur are those points on the receiver from where the projection of the triangle would lie on the source. This is equivalent to the union of all projections of the triangle from the source on the receiver plane. Since the goal is to consider all points potentially influenced, it is sufficient to compute a bounding region, the *influence region*. In the case that the triangle is two-sided, the convex hull of the projections of the triangle from the corners of the source result in the exact region (we provide a short proof in Chapter A). If this is not the case, the region is smaller and if one wants to build the influence region by projecting the source through

the triangle's vertices, a convex hull can be an overestimate. A better use of the vertices would be possible, which is shown in Figure 12.2. The correct approach, if the triangle is one-sided, would be to cut the penumbra region along the triangle's plane. Unfortunately, this modification is expensive on older cards just like computing the convex hull. Instead, we conservatively bound it by an axis-aligned bounding box. On newer cards it is possible to compute a convex hull in an efficient way and we explain this technique in Chapter 13. In this particular case of a planar receiver, there is another reason why the bounding quad is a good idea: It will allow us to efficiently interpolate information from the vertices of this quad and therefore avoid costly computations in the fragment shader. Such a solution would be more costly if many corners were present.

The bounding quad is computed from the triangle using a geometry shader. We pass the four matrices of the projections on the receiving plane from the source's corners as uniform parameters. For plane $ax+by+cz+d=0$ and corner (u,v,w), the matrix is given by:

$$M := [abcd]^T \times [uvw1] - (au+bv+cw)I \tag{12.3}$$

The geometry shader receives the triangle's vertices, projects them using the matrices, computes an axis-aligned bounding box of the obtained 4×3 vertices, and outputs it as a quad.

3.3 Backprojections on Light Source

For each point in the influence region, we find which part of the source is hidden by the triangle by backprojecting it onto the source. This backprojection can be interpolated from the backprojections of the vertices of the triangle's influence region. This is possible because it is a projective application. We compute these in the geometry shader using eq. (12.3) again, and pass them to the fragment shader as three interpolated texture coordinates. Note that this time, the backprojection matrix depends on the triangle, and should therefore be built in the geometry shader, not passed as a uniform. Coordinates of the backprojected triangle are computed in *normalized source frame* where the source is the square $x=\pm 0.5, y=\pm 0.5, z=0$.

At this stage, summarized in Fig. 12.3, we produce fragments that correspond to points inside the triangle's influence region and that have access, through three texture coordinates, to the backprojection of the triangle. The next step is to find which source samples fall in these backprojections.

Summary:

By treating all sample combinations blocked by a triangle while it is at hand, the computations become well adapted to a streaming architecture. So far we have seen how to detect the points on the receiver patch whose visibility is potentially affected by a given triangle. Further, we saw how to efficiently interpolate the backprojection of the triangle on the source inside of this region. The next step is to determine for a receiver point which samples on the source lies in the backprojection of the triangle.

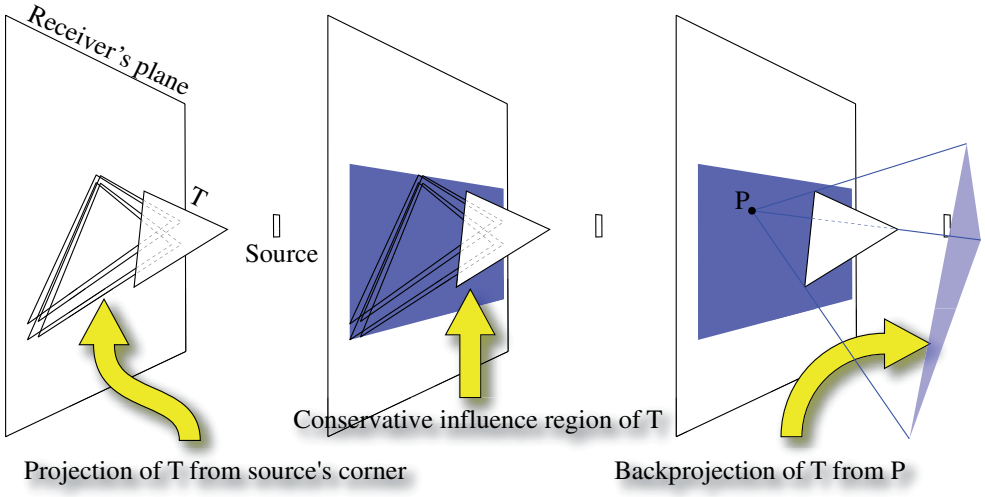


Fig. 12.3 : Overview of our approach for two planar patches

Our approach finds the region blocked by a triangle. A geometry shader computes an overestimate of the true influence region. A fragment shader computes the backprojection and hidden samples.

3.4 Samples inside Backprojection

To encode which samples are inside/outside a backprojection, we use a 32-bit bitmask encoded as RGBA8 color, that is, eight bits per channel. Our fragment shader will then output the sets of blocked samples as the fragment's color.

Computing which samples are inside the backprojection can be done by iterating over the samples, but this would be very inefficient. Since the set of samples inside a 2D triangle is the intersection of the set of samples on the left of the supporting line of each (oriented) edge, we can devise a better method based on precomputed textures, in the spirit of [KLA04].

An oriented line in the 2D plane of the normalized source frame can be represented by its Hough transform [DH72], that is an angle $\theta \in [-\pi, \pi]$ and a distance r to the origin. The distance can be negative because we consider oriented lines. However, we need only those lines that intersect the normalized light square so we can restrict ourselves to $r \in [-\sqrt{2}/2, +\sqrt{2}/2]$. In a preprocess, we sample this Hough space in a 2D texture called the *texture of bitmasks*. For every texel (θ, r), we find the samples located on the left of the corresponding line and encode them as a RGBA8 bitmask (Fig. 12.4, left).

At run-time, our fragment shader evaluates which samples lie within the backprojection as follows. We first ensure the backprojection is counter-clockwise oriented, reversing its vertices if not. Then the lines supporting the three edges of the backprojection are transformed into their Hough coordinates, normalized so that $[-\pi, \pi] \times [-\sqrt{2}/2, +\sqrt{2}/2]$ maps to $[0, 1] \times [0, 1]$. These coordinates are used to look-up three bitmasks (one for each edge). Each bitmask encodes the samples that lie on the left of each edge. The Hough dual is periodic on θ, thus we use the `GL_REPEAT` mode for it and *GL_CLAMP* for r. The latter correctly gives either none or all samples for lines not intersecting the normalized light square. These three bitmasks are AND-ed together to get one bitmask representing the samples inside the backprojection. This result is output as the fragment's color (Fig. 12.4, right).

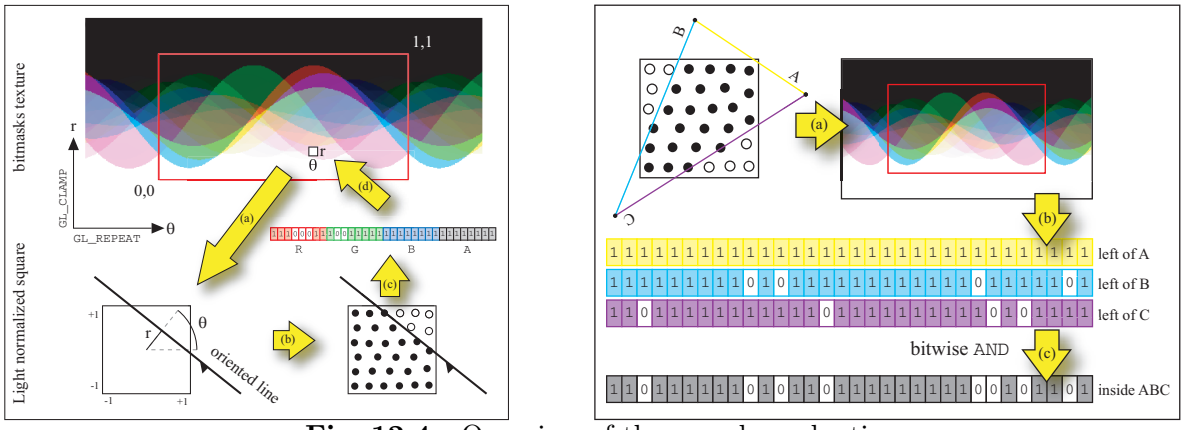


Fig. 12.4 : Overview of the sample evaluation

Left: Pre-processing the bitmask texture for a set of 32 samples. For every texel (θ, r) *we build the line corresponding to this Hough space value (a) and find the samples located on the left of it (b). These samples are encoded as a bitmask (c) that we cast as the RGBA8 color of the texel (d). The sine wave in the resulting textures corresponds to the Hough dual of the sample points. Right: To find the light samples within the backprojection of a triangle (a), we use the bitmask texture. We look up bitmasks representing the samples lying on the left of each edge (b) and combine them using a bitwise AND (c).*

Performing a bitwise logical AND between three values in the fragment shader requires integer arithmetic. This feature is part of DirectX10, and is implemented only on very recent graphics cards. However, if not available, it can be emulated [ED06a] using a precomputed 256^3 texture such that:

$$opMap[i,j,k] = i \text{ AND } j \text{ AND } k \tag{12.4}$$

Then, to perform the AND between three RGBA8 values obtained from the texture of bitmasks, we just do a 3D texture lookup to AND each of the channels together. This approach is general: any logical operation can be evaluated using the appropriate *opMap* texture.

3.5 Combining the Occlusion of All Triangles

The set output by the fragment shader for a receiver sample R_j and a triangle T_k can be formally written as:

$$\mathcal{B}_{j,k} \equiv \left\{ i \text{ such as } [S_i, R_j] \cap T_k \neq \emptyset \right\} \tag{12.5}$$

To compute the $\mathcal{B}_j$, we use the fact that $\mathcal{B}_j = \bigcup_k \mathcal{B}_{j,k}$. With our bitmask representation, the union is straightforward. All we have to do is OR the colors/bitmasks of each fragment incoming at the same location. This is done automatically by the graphics card by enabling the *logical operation* blending mode. The resulting image is called *hidden-samples map*.

3.6 Sampling Considerations

Our algorithm treats the source and the receiver asymmetrically. The source is explicitly sampled. The samples can be arbitrarily positioned using regular or irregular patterns. All that is required is to compute the texture of bitmasks for the chosen pattern. Conversely, the receiver is implicitly

sampled. The rasterization of the influence regions implies that the R_j are at the center of the texels of the hidden-samples map. The samples are regularly placed, and their number is given by the texture resolution. Using a 512×512 yields about 262k samples, which is much higher than the 32 on the source. Finally, the resulting hidden-samples map indicates which part of the source is not visible from receiver points. To get the opposite information – which part of the receiver a source point sees – it only has to be inverted.

Depending on the context, this asymmetry may not be a problem. It can always be alleviated by performing two passes, switching the source and receiver. In subsequent sections, we discuss how we can improve the number of samples, and the quality of the sampling.

3.6.1 Increasing Source Samples

We described the algorithm with 32 samples on the source because it is the number of bits available in a typical OpenGL color buffer. Recent cards support buffers and textures with 4 channels of 32 bits each, leading, in theory, to 128 samples. In practice, when using an integer-texture format, only 31 bits can be used because of reserved special values. Thus, we can treat 124 samples.

Of course, the number of samples can be increased using multiple renderings, but the benefit of the single-pass nature would be lost. We use Multiple Render Targets (MRT), a feature of modern graphics cards that lets fragment shaders output up to eight fragments to separate buffers. Thus, it is possible to evaluate up to 992 samples in a single rendering pass by rendering eight *hidden sample maps*.

3.6.2 Decorrelating Source Samples

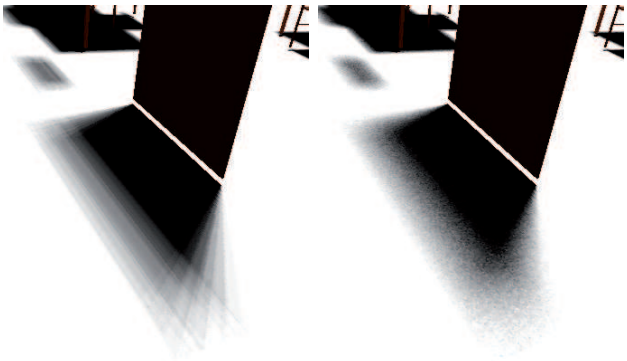


Fig. 12.5 : *Decorrelating source sampling for each receiver point produces noisier but less aliased results (32 light samples* 1024^2 *texture)*^{2}

If there are particular alignments of the scene and the light, or for large sources, a fixed sampling may become visible, even with 128 samples. To alleviate this problem, one can vary the set of samples for each receiving point. We just need to slightly perturb the positions of the backprojections by a random amount based on the position on the receiving plane. We use:
$rand(x,y) = \mathrm{frac}\,(\alpha(\sin(\beta y * x) + \gamma x))$ which produces a reasonably white noise if α, β, γ are correctly chosen depending on the texture resolution. It is interesting that there is a noise function exposed in GLSL, unfortunately the current implemented behavior is to simply return zero. It is nevertheless possible that one day such a function becomes available and will hopefully eliminate the need for this work-around.

3.7 Backface Culling

If the caster is watertight, i.e., it encloses a volume, we can ignore every triangle that is front-facing for all points on the light source. Indeed, a ray blocked by such a triangle is necessarily blocked by a non-front-facing one (it must enter and exit the volume). Culling the front faces is more interesting than culling the back faces, because it yields smaller influence regions. In the geometry shader, we test if the triangle is front-facing by checking if it is front-facing for the four corners of the light source. If yes, it is safe to let the geometry shader discard it. This optimization eliminates roughly half the faces, and thus doubles the framerate.

3.8 Implementation on DX9 Cards

The algorithm can be implemented also on older hardware, without the geometry shader. The solution is to create a *shadow mesh* from the original triangle mesh. For this structure, each triangle is replaced by a quad, storing in its texture coordinates the three vertex positions of the original triangle. The position-information of each vertex indicates what corner of the bounding quad it is supposed to represent (there are four entries in the position vector; setting the corresponding component to one allows to define what corner the vertex will correspond to). With this representation it is possible to determine the bounding quad of the triangle in the vertex shader using the texture coordinates, and to select the according corner. The other obstacle is the sample evaluation. Bitwise operations are not supported and we thus resort to texture-based evaluations to encode the AND operation. Further, 32-bit textures do not support blending and thus impose a strong restriction. We can only compute 128 samples (4 color channels each with 8 bits and 4 MRTs) in a single pass.

Summary:

We presented how to efficiently evaluate blocked samples using the Hough Transform and bitwise operations. The key is that the evaluation of blocked samples has very little cost because it boils down to simple texture lookups. We then presented several accelerations and how to decorrelate the source samples from the receiver samples to avoid aliasing. The implementation is possible on DX9 hardware too, but is more natural for DX10. We next show applications of our technique.

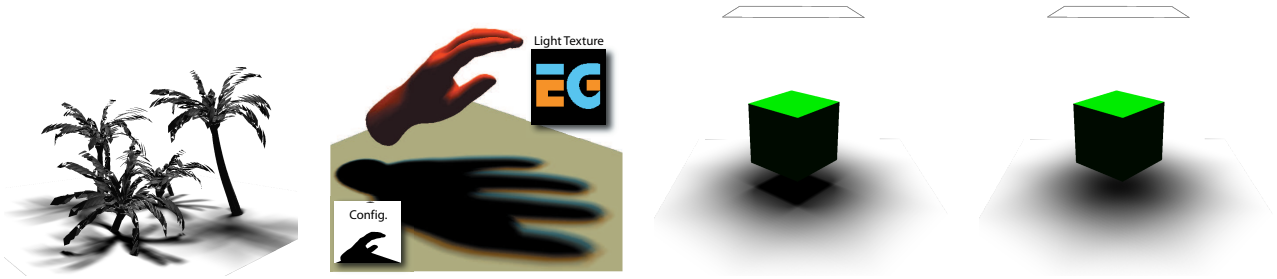
4 Application to Soft Shadows

We can use our method to compute the soft shadows caused by an area light source on a receiving plane. Once we have computed the hidden-sample map, we convert it to a *shadow intensity texture*. This is done in a second pass, rendering a single quad covering the viewport and activating a fragment shader that does the conversion. The shader fetches the hidden sample bitmask for the fragment, and counts how many bits are at 0. This step is similar to the bit counting in Section 1 of Chapter 8.

Finally, we render the receiver texture-mapped with that texture, using linear interpolation and mipmapping. Alternatively, we can do bit counting directly while rendering the receiver. This is

particularly interesting if the influence regions were projected from the receiving plane into the current view, which eliminates any aliasing.

Our approach works correctly for very large sources, where many methods would fail due to the approximation of silhouette edges from the center of the source. An approach like [HPH97] requires many samples, otherwise one can see the sampled nature of the shadows because similar superposed shapes are visible. In our approach, we can decorrelate the sampling (see Section 3.6.2) and trade this artifact for noise. Figure 12.5 shows the improvement.



(a) (b) (c) (d)

Fig. 12.6 : Visibility sampling results

(a) An example of shadows using our method (b)Soft shadows caused by a textured light source, using 2 different colors. Notice the complex shadowing effect. The bottom left inset shows the source/receiver/scene configuration (c) A difficult case handled robustly by our method. Approximating shadows based on silhouettes from the center causes incorrect shadows (d). For moving light sources this approximation also causes strong popping.

Methods such as [AHL*06] are restricted to rectangular light sources. Our approach can handle any planar light source. We find a bounding rectangle, so we can compute backprojections from its corners, and place the samples where there is light. This can be very handy to compute shadows for neon-like sources where there are typically several long and thin tubes inside a square. It is also possible to use color-textured light sources. We pass to the shader the number of different colors, an array of color values, and an array of bitmasks indicating which samples are associated to each color. Then the shader counts bits in each group as before, multiplys by the group color, and sums the result. In our implementation, we organize samples so that the bits inside a texture channel correspond to samples of the same color, so we do not need to pass group bitmasks. Using eight MRTS, we can have up to 24 different colors (which is usually sufficient) and incur a negligible extra cost. Figure 12.6 shows an example of a colored shadow.

Although we described our algorithm with a planar receiver, it can be adapted to handle a bumpy one. The backprojection of the triangle can be done in the fragment shader (instead of computing it in the geometry shader and interpolating it for fragments, as described earlier). Thus, we can offset the fragment's world position, for example looking up a height map (this requires that the receiver is a heightfield as seen from at least one point of the source). As a result, we generate accurate soft shadows with high sampling rates for non-planar receivers, which has not been achieved before at interactive rates. (This computation is key to allow the treatment of general scenes in Chapter 13.) Figure 12.7 shows an example. Some care must be taken, however; to assure conservative influence regions, the projection from the light's corners (Section 3.2) must be performed on a plane lying behind the receiver. A (general) backprojection is a costly operation. Performing it in the fragment – not the vertex – shader can seriously effect the performance, but

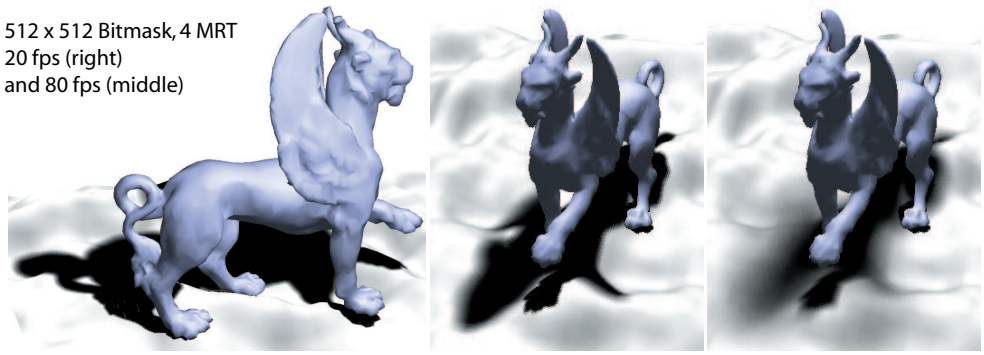


Fig. 12.7 : Soft shadows on a non-planar receiver

Using our visibility sampling we can also compute accurate shadows on bumpy grounds

if the receiver and source planes are parallel - which is a natural choice when using a heightfield - the backprojection has a very simple expression because we can express the receiver samples in the light's frame.

We first compare with the method by Assarson et al. [AAM03, ADMAM03]. Their algorithm has two major shortcomings. First, only edges that are silhouettes for the center of the light are considered. This leads to temporal incoherences and inaccurate soft shadows as seen on Figure 12.6. Furthermore, our result is accurate (up to the sampling), so these artifacts cannot be observed.

This said, our method has several drawbacks. It separates the caster from the receiver. It computes shadows on a planar (possibly bumpy) receiver, like some recent approaches [AHL*06]. Consequently, it is probably not suited to applications like video games, and would be more interesting for fast previsualization during the creation of static lightmaps. Other tasks also benefit from our method, as demonstrated in the next application.

5 Application to Assisted Visibility Edition

In video games, it is very important to guarantee that the framerate never drops below a certain threshold. For example, levels are often designed in a way that level-of-detail switches are not visible, or that many complex objects are not seen together from the current viewpoint. Most of this is anticipated during the design phase. Yet, the scene often needs to be modified afterwards. Typically, beta testers report where the game slowed down, then the artist moves around objects a little, or adds some dummy occluders to reduce complexity. It is interesting that some elements of decor are added just to block the player's view.

Visibility is not only a performance concern. It is often important to enforce that some objects remain invisible while the player is following a certain path in the game. If guards could be killed from a distance a long time before they are actually supposed to be encountered, gameplay would be seriously altered. Here again, tools to interactively visualize visibility could help designers prevent such situations.

With our method, it is possible to place two rectangular regions in a scene, and immediately get information concerning their visibility. In particular, unblocked rays are returned (reading

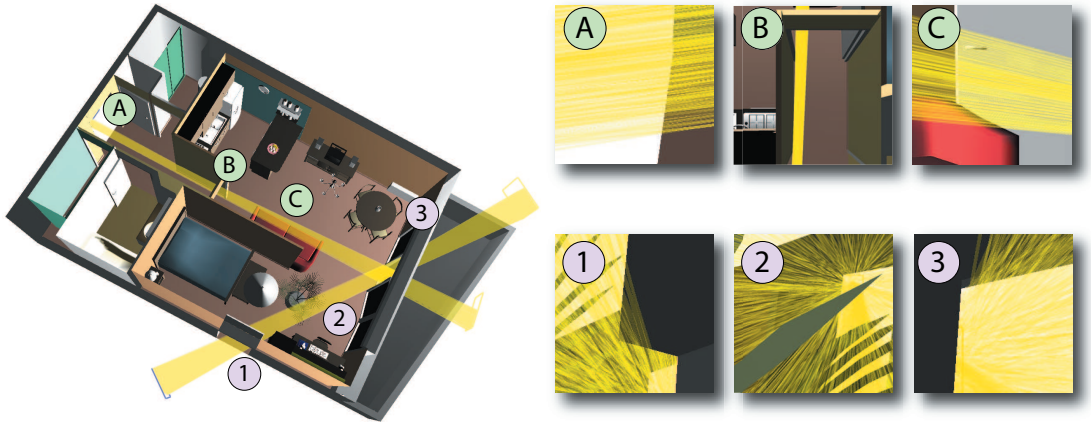


Fig. 12.8 : Interactive visibility visualization

A designer can navigate a scene, place any two patches, and immediately (about 20Hz) see the unblocked rays joining the patches. Here, we show two such shafts (yellow). Closeups emphasize how well our method captures visibility, even for small features like the leaves (2) of the plant. Our webpage offers a video showing a demo session.

back the hidden-samples map). This information can be used to decide where to place objects to obstruct the view between the areas. The algorithm for soft shadows can be used to calculate an importance map indicating *most visible* regions. Accumulation over time is possible and allows us to investigate the impact of moving objects.

We implemented such a system to visualize unblocked rays between two patches (Figure 12.8). We use 32×32 samples on source and receiver, which amounts to 1M rays considered, yet it is very fast, even for complex scenes. The apartment model we used has approximately 80,000 triangles, and a complex instantiation structure. A thousand renderings (Heckbert-Herf's approach) take several seconds. With our method, we achieve about 20 fps, including a naive (inefficient) ray visualisation, which provides interactive feedback to an artist designing a scene. She can edit the scene in real time, and adapt it to control the desired visibility. Note that our method scales extremely well with geometry in this context. In particular, it inherently performs shaft-culling: when a triangle is outside the shaft, the influence region calculated by our geometry shader is outside the NDC range. It is thus clipped and produces no fragments.

The fact that we can consider many rays per second largely compensates for the inherent inexactness of sampling. Interestingly, the visualization of rays takes up more GPU resources than the determination of the rays. We compared the amount of ray per second with state-of-the-art ray-tracing approaches. Wald et al. describe a fast method to perform ray-tracing for animated scenes [WIK*06]. The reported timings (on a dual-core Xeon 3.2GHz processor) for intersection tests are 67Hz for a 5K model, 36Hz for a 16K, and 16Hz for a 80K model, using a viewport of 1024^2 pixels, which amounts to 1M rays.

Our approach does not perform ray tracing (intersections are not retrieved), but in terms of speed, it achieves very high performance (see Section 6), is simpler to implement than a complete ray-tracing system, and runs on an off-the-shelf graphics card. Our approach works seamlessly with GPU-based animation, skinning, or geometric refinement. It is executed completely on the

GPU, leaving the CPU available to perform other tasks. Finally, it would even be possible to combine typical ray-tracing acceleration techniques (e.g., frustum culling) with our approach.

Summary:

We presented two applications of our visibility sampling. Soft shadows are accurate with respect to the entire geometry and we can efficiently extent the solution to bumpy grounds, which previously could not be treated accurately in real-time. The second applications is very novel. We explain how to use the algorithm to support visibility edition in scenes. For level design it is sometimes important to block the view from particular objects or to introduce blockers to allow more efficient geometry culling during the execution of the game.

6 Results and Discussion

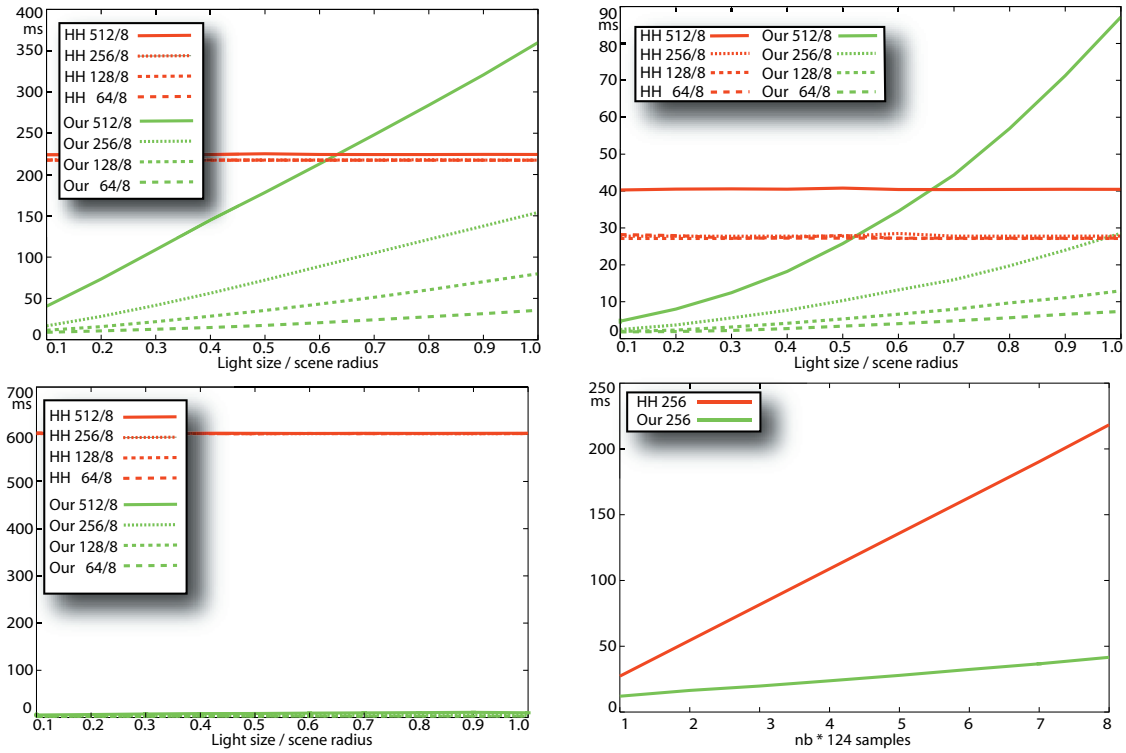


Fig. 12.9 : Performance for the Visibility Sampling algorithm

Compared influence of light and texture size for Heckbert & Herf(HH) and our method and 992 samples (8 MRTs). Top left: palm trees (29 168 polys). Top right: bull (here 4000 tris). Bottom left: office (1700 tris). X-axis indicates light size with respect to scene radius. Y-axis reports rendering time (ms). Bottom right: sampling against computation time for the palm tree scene (top left) with a light/radius ratio of 0.4. For our method, more samples have almost scene independent cost.

We made various measurements comparing our method with a multiple-rendering approach [HPH97] on a P4 3000MHz. To allow for a fair comparison, we implemented a modified version of Heckbert and Herf's algorithm. We rasterize directly into a luminance texture using alpha blending and a stencil test (similar to [Kil99, Bli88]) instead of using the accumulation buffer [HA90]. This requires blending on float textures, which is available on the latest cards. This modified implementation is about ten times faster (for 1000 samples) than the original accumulation buffer version.

Figure 12.9 shows the light's influence. Our method is fillrate limited. The larger the source, the larger the influence region and the slower the method becomes. On the contrary, Heckbert's method is independent of the sample's location, hence the size of the source. For a giant source, it would, however, create aliasing that our method would address (see Figure 12.5). We have really pushed our algorithm, since we considered very large sources (up to the size of the model) very close to the object. Yet, our method behaves better for sources up to half of the scene radius. For the bull model, we activated the watertight optimization (Section 3.7); this is why the performance drops more strongly than for the palm tree, where it was deactivated. When the source is very large, there are fewer and fewer fully front-facing polygons.

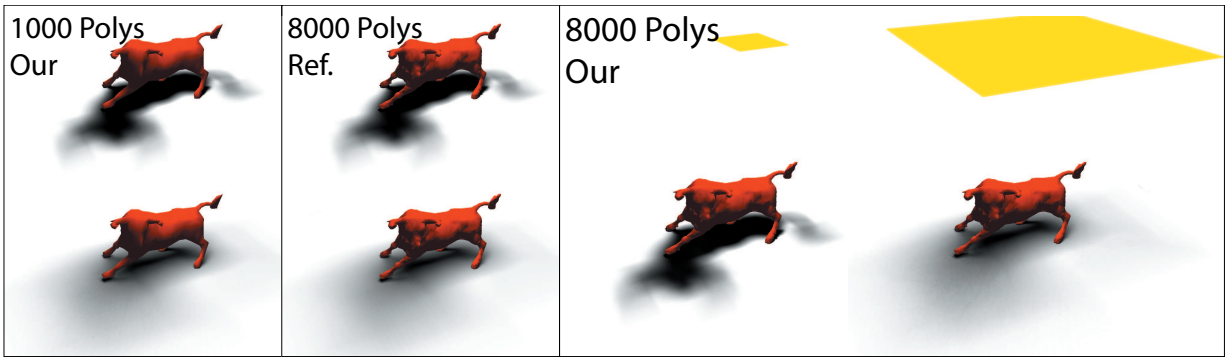


Fig. 12.10 : Quality of our Visibility Sampling method

Our method delivers faithful visibility results. One major cost factor is the fillrate. It becomes expensive in the case of big sources due to overdraw. Interestingly these configurations represent exactly those, where a coarse model (left) would provide almost the same shadow. Therefore fillrate can be reduced drastically via simplification.

The bottom curves of Figure 12.9 are surprising at first: our method performs much better than Heckbert and Herf. Because the result is obtained in a single pass, we apply vertex- and geometry computations (transform, skinning, etc.) only once. Real-life scenes, like the office, often contain objects instantiated several times with different transformations. This causes many CPU-GPU transfers of matrices, context switches, etc. that impede the performance when rendering the scene hundreds of times. The polygon count alone is not a sufficient criterion.

There are other factors that influence the performance of our approach, which therefore depends strongly on the type of scene. A small triangle close to a large source has a very large influence region. Since we treat each triangle independently, highly tesselated models incur a high overdraw and a performance drop. A natural solution is to perform mesh simplification on such models, and to use the result, which has fewer triangles: not only is this natural, it also makes sense. As pointed out in [DHS*05], high frequencies are lost in the presence of large light sources. The

simplified model, even when very coarse, casts shadows that closely match the exact one, as shown in Figure 12.10. This improves rendering time, as shown on Figure 12.11.

Note that it makes sense to compare the polygon count of different versions of the same object. Across models, it is irrelevant: the palm-tree scene has 30K polygons, but since the leaves are small they have relatively small influence regions and cause less fillrate than the 4K polygons of the bull model.

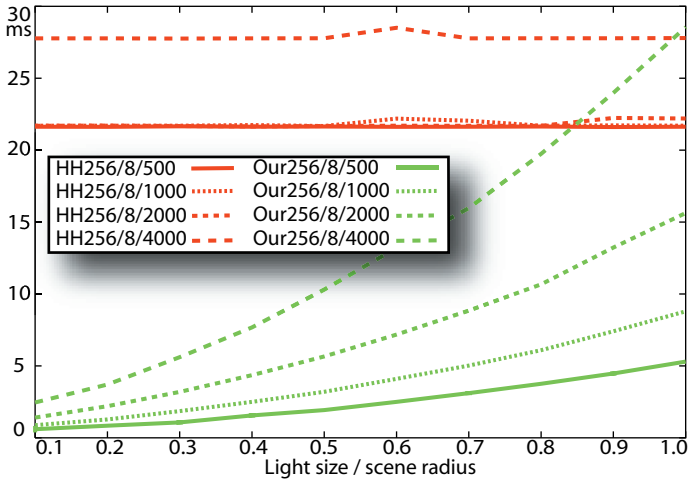


Fig. 12.11 : Rendering time (ms) vs. nb polygons. - *X-axis indicates the size of the light source. The different curves indicate differently simplified versions of the bull model (polygon count is indicated in the legend).*

One feature of our algorithm is that there are no real degeneracies. It always works, up to the precision of the Hough sampling. Special cases, e.g., if the triangles intersect the receiver, can be solved by transporting the backprojection to the fragment shader (as for the bumpy ground). Alternatively, the geometry shader could retriangulate the intersecting triangles. We want to outline that most results in this chapter have been computed with relatively large sources, very close to the caster, which is a worst-case scenario for most other methods. Yet, it works well, even for notably difficult cases such as for the cube in Figure 12.6.

We also implemented the DX9 solution and obtained a weaker - but still remarkable - increase in speed in regard to standard sampling. As for the DX10 solution, speed varies depending on the model and light size. For a 10% (with respect to the bounding radius) light size we obtain for a difficult case (a tree model with many small occluders close to the light and thus casting large penumbrae) an acceleration of 1.25. Less complicated cases, like a person with 2,736 triangles, leads to a increase in speed of 3.6. The office model, which is a more realistic scene, shows as discussed above a strong gain of a factor 22. For even larger light sources, on the other hand, the acceleration quickly vanishes. Nevertheless, for most applications 10% of the bounding radius is already rather large. The complete overview of the timings measured on a Geforce 6800 ULTRA are listed below:

	light size	10%				20%			
	# samples	32	64	96	128	32	64	96	128
Person	our	3.6	5.3	7.2	8.6	7.3	10.3	14.0	16.6
2736 tris	accum	7.8	15.5	23.2	31.0	7.7	15.5	23.2	31.0
Tree	our	50.4	77.7	106.2	127.2	110.3	158.5	219.1	261.1
17804 tris	accum	39.0	76.6	115.4	153.6	39.1	76.5	114.7	153.4
Office	our	2.1	2.4	2.8	3.2	2.4	3.1	3.6	4.1
1700 tris	accum	18.0	36.0	53.3	70.3	18.0	35.8	53.2	70.6

Summary:

Our algorithm performs well in many scenarios, but performance is strongly related to the size of the source patch. If it is too large cost increases rapidly. One thing that is important to notice is that our solution is provided in a single pass over the geometry. If it is costly to perform this traversal the gain with respect to several passes can be tremendous.

7 Conclusion

We presented a fast, GPU-based visibility algorithm without much CPU interaction that samples equivalently to ray-tracing, but is much simpler to implement and faster ($512 \times 512 \times 1000$ samples ≈ 260M rays). Of course, ray-tracers are much more general. We believe it demonstrates the benefits of bitwise arithmetic on modern GPUs, whose interest has already been shown in Chapter 7.

We presented two possible applications illustrating the utility of our method. The soft shadow application – although providing accurate, real-time shadows on non-planar receivers for the first time – should be seen as a proof of concept. It is probably not suited for games where realism is unnecessary. It can be useful, though, for fast computation of lightmaps. The visibility edition is a novel application. Its strength is its speed and the ability to provide unblocked rays. It can be implemented easily and does not interfere with any GPU technique (geometry encoding, skinning, etc.). The information read-back is inexpensive, due to the small texture size.

8 Future Work

There exist various possibilities for future work. We would like to combine our method with algorithms like [NB04] to compute PVS. We are also interested in using it to increase the speed of form-factor estimations in hierarchical radiosity approaches [SP94]. Finally, we are working on improvements of our soft-shadow computations. Inspired by [DHS*05], we can group triangles based on their distance to the source. The closer groups could be rendered in low-resolution textures, and the furthest ones in high-resolution textures, thus reducing fillrate and improving speed.

"I'm a level 5 vegan, I don't eat anything that casts a shadow."

The Simpsons

Chapter 13

Sample-Based Visibility for Soft Shadows Using Alias-Free Shadow Maps

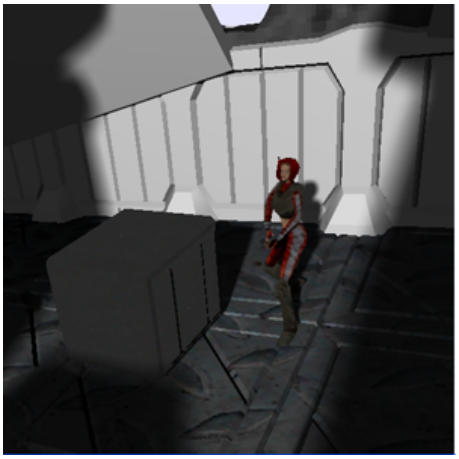


Fig. 13.1 : *Accurate Soft Shadows in a Game Level*

This chapter extends the work on visibility sampling, from the previous chapter, to a more general setting. We treat arbitrary scenes at interactive rates using a standard graphics card. The view-samples are no longer restricted to a planar receiver. Even though we focus on soft shadows in this chapter, this property is interesting in many situations. For artificial intelligence it is often important to take the current field of view of each individual into account to drive its actions. These tests usually represent a major challenge.[1]. We believe that the work in this chapter is a first step in the direction of general accelerated visibility determination. Nevertheless, we will present this work in the context of soft shadow computation. Currently, the algorithm is the only one that produces soft shadows with 1024 samples accuracy at interactive rates for medium-sized scenes.

To achieve this goal, we developed a data representation that allows us to store a chained-list-like structure efficiently on the GPU. We are convinced that this can potentially be useful in many other applications. It is also the key to compute pixel-accurate shadows (hard and soft) in a single pass, by storing view-samples in a shadow map. The lists ensure that all elements are treated. We further address other problems of shadow computations like depth bias and incoherent shading models and provide a GPU friendly visibility test for volumetric sources. This chapter is a nice closure of our work on shadows, by providing a geometry-accurate solution, for arbitrary scenes and sources.

Publication notice: *This chapter represents the result of a collaboration with Erik Sintorn and Dr. Ulf Assarsson from Chalmers University of Technology (Sweden). The work was published in Computer Graphics Forum (Proceedings of EGSR) [SEA08].*

[1] During a conference I had the chance to talk to a software developer for Disney. Its current research aims at the development of tools to test the impact of architecture in theme parks. For this they construct the scenery and use a crowd simulation to examine the behavior of the individuals. For each entity many visibility tests are necessary to obtain the visible percentage of each attraction to control the behavior.

1 Introduction

As mentioned, the goal of this work is to lift the restriction of having receiver samples fixed in a regular pattern on a planar receiver. To simplify explanations we will nevertheless describe our contributions in the context of shadows. This eases comprehension because many structures correspond to representations from the previous chapter.

2 Alias-Free Shadow Maps for Hard Shadows

Overview -Initially, the scene is rendered from the eye into a *view texture* to find points (*view-samples*) for which shadow computations are needed. These are foremost the visible points from the eye. Samples on back-facing geometry w.r.t. the light source can also be excluded since they self occlude (see Figure 13.2). The view-samples are then projected and stored in the shadow map (SM).

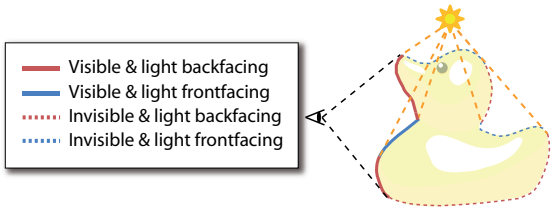


Fig. 13.2 : Extracting View-Samples - *Only visible and light front-facing view-samples (non-dashed blue) are stored in the shadow map. The world space coordinate and shading status of each sample is maintained in a local list at the shadow map pixel. For closed objects, only light back-facing triangles (red dashed and non-dashed) are used as shadow casters.*

Several of the view-samples can fall in the same pixel (see Figure 13.3 - left), which is the source of aliasing in standard shadow mapping. To avoid collisions, we store them in lists, each maintained in a SM pixel (Section 2.1).

Next, we test whether each view-sample is lit or in shadow. For this, we render the scene from the light source as follows. For each triangle t, a fragment shader will be executed on the SM pixels that are partially intersected by t's projection. We achieve this by using conservative rasterization (Section 2.2). For each covered pixel, a fragment shader traverses the list of view-samples stored at this location and determines whether t hides the view-sample from the point light. All triangles are treated separately. After each pass, the shadow information (lit/unlit) is combined with the result for the previously tested geometry (Section 2.3).

Once all triangles are processed, the shadows are applied by rendering a screen-covering quad in the observer's view, and each view-sample recovers its corresponding shadow value from the lists.

2.1 Constructing and Storing the Shadow Map Lists

Filling the shadow map lists with the view-samples uses CUDA's scattering capabilities. The process is memory compact and faster than stencil routing [MB07].

Our goal is to store in each pixel of the SM a *list length* s and a *list offset*. The list offset points into a global array, I_A (see Figure 13.3), from where all consecutive s elements correspond to the entries of the *local list*. With this information, one can iterate over the list elements of each SM pixel by reading successive elements in I_A. Further, each view-sample has an *offset* into I_A. It is

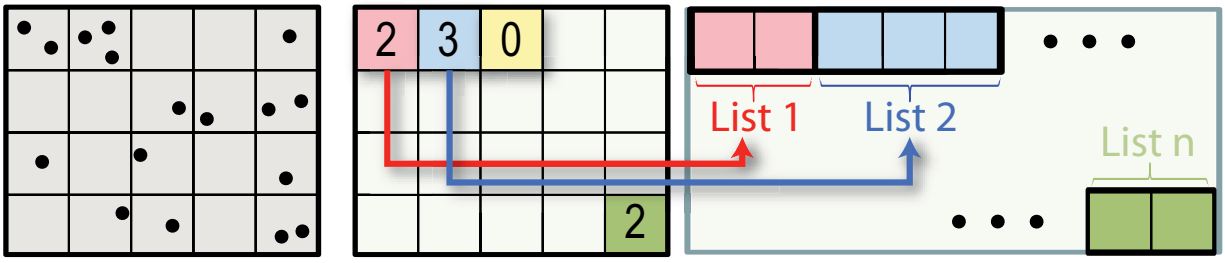


Fig. 13.3 : Memory layout of the SM Lists

View-samples are projected into the SM (left). An offset and length per pixel allow to access the sequentially stored lists (right).

stored in the view-texture and allows us to hedlook a value up associated to a view-sample. In the context of our shadow computation, each entry of I_A holds the view-sample's status information (lit/shadowed) as well as its world position.

2.1.1 Construction

To obtain this structure, we intialize the local lists' lengths in the SM to zero. Then, each view-sample, $\mathbf{p}_{cam}$, is projected into the SM, to a position $\mathbf{p}_{SM}$. The current local list size s at $\mathbf{p}_{SM}$ is read and incremented instantaneously using the atomicInc() instruction in CUDA. The read value is stored at $\mathbf{p}_{cam}$'s position in the view texture. This process associates a unique index i to each view-sample that can be used as an offset into the local list into which it projects. At the same time, on the SM side, this counter ultimately indicates the number of elements in the pixel's local list.

To concatenate the local lists in the global array I_A, each one needs a particular offset. For this, a running sum scan [HSO07] is evaluated on the list-length map. This process interprets the 2D texture as a 1D array (line by line) and stores in each pixel the sum of all proceeding values. These are exactly the offsets that represent the start of each list in I_A.

To directly access the data in I_A for a view-sample, we find its offset by adding the list offset of the list at $\mathbf{p}_{PM}$ and the view-samples index i. This final offset will not change until the end of the frame and we store it in the view-texture.

2.1.2 Storage

In practice, instead of having one array I_A with heterogenous data (world position and lit/shadowed state) we store them in separate arrays using the same offsets we derived for I_A. This allows us to pack the shadow data densely in single bits. This is beneficial because even though the world position remains the same throughout one frame, the shadow state will be updated for each processed triangle and it becomes possible to use the card's bitwise blending operation for this. Today, eight render targets and 32-bit channels allow us to update shadow-state values for 1024 view-sample per list and render pass. If more view-samples are in a list, blocks of 1024 samples are treated, and multi-pass rendering is used until all view-samples have been processed. However, in practice, even 128 samples prove enough in most cases.

Summary:

We have seen how to implement a list in each texel of the shadow map. This allows to project view-samples into the shadow map and add them to the lists. For alias-free hard shadows, the next step we will loop over all triangles, project them in the shadow map and test all view-samples in the covered lists for occlusion.

2.2 Conservative Rasterization

Having lists of view-samples in each pixel of the SM, we next rasterize the scene's triangles one by one into the SM. For each pixel intersected by a triangle t, the local list is traversed and each view-sample is tested for occlusion by t with respect to the point light source. The rasterization needs to be conservative (Figure 13.4). Otherwise, if the center of the list pixel is not covered, no fragment would be produced and thus the computation will not be applied to the list's elements (which in turn, could lie in the partially covered region). We use a version of Hasselgren and Akenine-Möllers's algorithm [HAMO05] executed on the geometry shader, but we always output a fixed number of six vertices (four triangles) instead of up to nine vertices (seven triangles) since that turned out to be significantly faster. Our trade-off is that case (iii), shown in Figure 13.4c), can be overly conservative by one pixel, which has no effect on correctness in our situation.

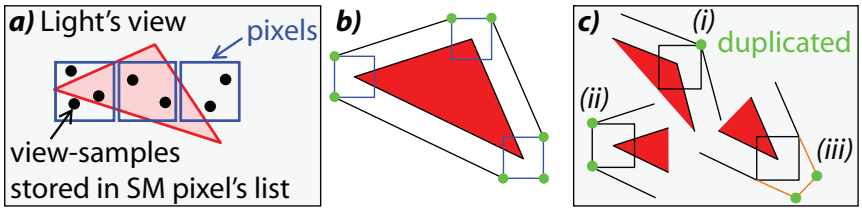


Fig. 13.4 : Conservative Rasterization

a) Without conservative rasterization, only pixels whose center is inside the triangle are rasterized. Since sample points can lie anywhere in the pixel, conservative rasterization is required. b) Conservative rasterization is done in the geometry shader, by computing the convex hull of the triangle's vertex extension to a pixel-sized quad [HAMO05]. c) Three cases can occur at corners of the convex hull. Performance is improved significantly if the area is approximated with only two vertices at each corner. For case iii) edges are extended by the length of a pixel's diagonal.

2.3 Evaluating Visibility

A given view-sample is tested against a triangle t by computing distances to the planes given by t's supporting plane and the pyramid defined by t and the light source. This has the advantage that we can compute the plane equations once in the geometry shader and then perform rapid tests in the fragment shader. Alpha-mapped geometry can be used, we would only need to perform a lookup corresponding to the texture coordinates, here the computation of barycentric coordinates is the most efficient scheme.

Avoiding Bias The algorithm has a nice property: for two-manifold (closed) shadow casters, either the light back-facing surfaces or the light front-facing surfaces suffice to determine the shadows. This fact can be utilized to avoid incorrect self-shadowing and the corresponding problem of surface acne, without using the classic scene-dependent depth bias [Wil78], triangle IDs [HN85], or depth values inside closed objects [Woo92,WE03]. Light back-facing view-samples do not need to be transferred into the SM lists, since they always will be in shadow by the shading. Thus, selecting only the light back-facing triangles as shadow casters eliminates self-shadowing. A triangle will never be both a shadow caster and receiver. This also avoids numerical imprecision problems from floating point transformations between coordinate systems [AL04, Arv07]. Theoretically, problems might remain where front- and back-faces touch, but in practice we did not observe such artifacts. No example in the paper uses any depth bias (e.g., see left part of Figure 13.15 that renders in 66 fps). To avoid even this potential problem, an epsilon value could be introduced to offset the triangle's supporting plane slightly. There are two possible offsets (toward the light and away). Due to our culling strategy, self-shadowing does not occur and both are valid. One favors light -, the other shadow leaks. The latter are less disturbing than light leaks (a bright spot in the dark is more obvious than a slightly extended shadow). Moving the supporting plane towards the light is, thus, the better choice.

A similar trick is actually often used for standard shadow maps to ameliorate the problem of surface acne. For closed shadow casters, only the back-facing triangles with respect to the light source are rendered into the shadow map. However, since standard shadow maps are not alias-free, artifacts will still appear near all silhouettes, and careful adjustment of a bias and resolution is required to suppress the problems.

2.4 Short Discussion of Alias Free Shadow Maps

Our solution for alias-free shadow maps shows many similarities with the work by Johnson et al. [JLBM05]. The main differences are that Johnson et al. proposed hardware extensions to treat more general cases than just hard shadows. Their algorithm makes heavy use of dynamically allocated linked lists. Therefore, besides the adapted memory structures, a supplementary processing unit and waiting queues were proposed to synchronize work. Our solution uses a different way to represent lists. It makes them faster to access in our context (linked list iterators need to follow pointers). They are implementable on current hardware and can be more easily parallelized (only the cheap atomicIncr() operation needs synchronization). No dynamic reallocation is required. Each sample's final position is given by an offset in a global array, which is always of constant length (tightly bound by the view-texture resolution). Further, we compute shadows only where needed, our conservative rasterization improves upon previous work, and we introduced a way to avoid depth fighting and decrease the number of caster triangles.

Summary:

We achieved a pixel-accurate hard shadow algorithm that can be used for arbitrary scenes, including alpha-mapped geometry, for this, we presented an improvement of previous conservative rasterization to mark all lists in the SM that potentially contain shadowed view-samples.

We also explained how to efficiently test for occlusion and provided some acceleration strategies, e.g., all back-facing view-samples don't need shadow computation.

This allows us to circumvent a depth bias when the scene contains watertight objects because no triangle can be caster and receiver at the same time.

3 Soft Shadows

Only two steps of the previous SM algorithm need to be modified for soft shadows by arbitrary volumetric sources. First, the conservative rasterization needs to cover the entire umbra and penumbra region of each shadow casting triangle, the so-called *influence region*. Second, the fragment shader testing for occlusion needs to consider multiple light samples. This is solved using a 3D-texture as explained below in Section 3.2. With respect to the previous chapter, we treat arbitrary receivers with our lists. We also compute tighter influence regions because this time it makes sense because each covered pixel represents a list of view-samples and can thus be potentially very expensive. Further, we will present how to use volumetric light sources and introduce temporal anti-aliasing.

3.1 Computing the Triangle's Influence Region for a Spherical Source

A point $\mathbf{p}$ can only be shaded by a triangle t, if it lies in the union of t's projections from all light source points onto a plane passing through $\mathbf{p}$. The same holds for a plane further away (as seen from the light's center). Therefore, conservative choices for an arbitrary point of the scene include the far plane (see Figure 13.5), a floor plane (if the scene exhibits one), or a plane at a distance of the furthest view-sample. As can be seen in Figure 13.5 (right), when the distance between the light center and far plane grows, the ratio between the size of the influence region and the base of the light frustum is asymptotically bound by the ratio of their respective angles (green and blue). Thus, the size of the influence region is typically more dependent on the triangle's distance to the light than the choice of the far-plane.

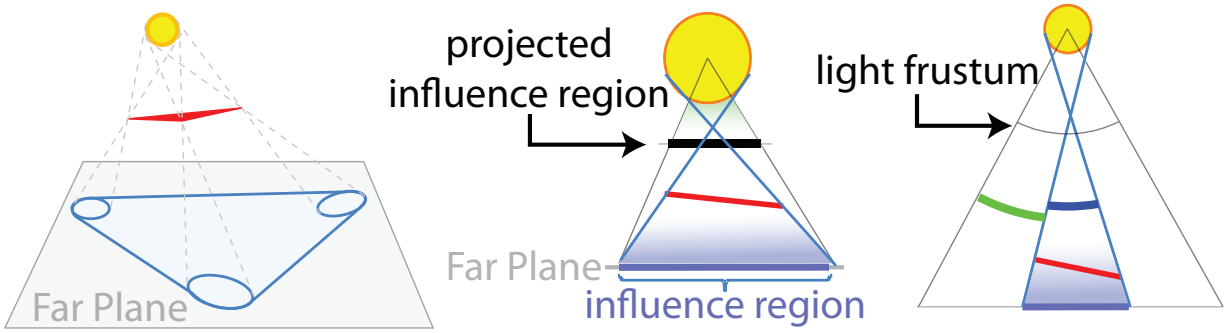


Fig. 13.5 : Influence region for a volumetric light source

Left: a triangle's influence region at a far plane. Middle: affected shadow map pixels when rasterizing the influence region. Right: the influence region size as seen from the light, is asymptotically bound by the ratio between the blue and green angle.

In the case of a planar source, it is relatively simple to calculate the influence region from a shadow-casting triangle. Let us first concentrate on a spherical light source. Here, the influence region on a plane is more complex. It is the convex hull of the source's projection through the vertices which form ellipses in the plane.

We provide a solution that allows arbitrarily tight bounding regions for spherical sources and apply it to arbitrary receivers (see Figure 13.5-left). For each triangle vertex v, we compute sample points of a regular bounding polygon of a desired degree (e.g., a hexagon for six points) for the light's silhouette as seen from v. This silhouette is a circle whose radius and position can be easily inferred and thus the construction of a bounding polygon $\mathcal{P}_v$ is direct. For each vertex w of $\mathcal{P}_v$'s vertices, we compute the intersection of the line through v and w with the desired far plane. These intersection points form bounding polygons of the ellipses and their convex hull would bound the influence region. The convex hull is computed with Graham's scan [Gra72]. The method is $O(n)$ if all points are in clockwise angular order. This is easy to assure for the points of each ellipse, but they do not share a common center. Fortunately, merging them is possible in $O(n)$ time. This makes the full convex hull computation very fast. In practice, we chose hexagons and CUDA was found to be up to twice as fast as geometry shaders.

Conservative rasterization of the influence region is done by adapting the rules of Figure 13.4. We only output one vertex in case (i) and for case (ii) if the angle between the edges is $< 90°$ their intersection point is used to output only one vertex.

To ensure conservative coverage of the influence regions for an arbitrary volumetric light source, we first approximate it by an ellipsoid. A *virtual* scaling of the entire scene allows us to reestablish a spherical light source for which the previous algorithm can be used.

3.2 Sampling Visibility of Volumetric Sources

For hard shadows, we used a single bit to represent the visibility of the point light source. To compute soft shadows, we associate a *bitmask* to each view-sample, where each bit corresponds to the visibility of one light sample. This means that 8 MRTs allow us to output 128 light samples for eight view-samples in a single pass. Multipass rendering is used until all view-samples are computed.

To sample visibility in the influence region, we use a light sample-dependent look-up table LU. LU takes a 3D plane as input and returns a bit pattern corresponding to the light-samples behind the plane (Figure 13.6 top-left). Combining the bitmasks of the planes defined by the view-sample and the edges of the triangle t via an AND operation establishes the correct visibility with respect to t (Figure 13.6 top-right). This is a direct extension of what we presented in the previous chapter to the 3D case and thus allows to handle non-planar light sources and shadow receivers. To accumulate the results, we use, as before for the hard shadows, the blending capacities of graphics hardware (namely the bitwise OR operation).

We can represent LU with a 3D texture by describing the plane with two angles and a distance. To minimize discretization errors, we choose LU's size relative to the number of samples, typically 128^3 per 3D-texture. It leads to good shadow quality with moderate memory cost. If more samples are needed, we can precompute additional k LU-textures for $k \times 128$ different sample locations.

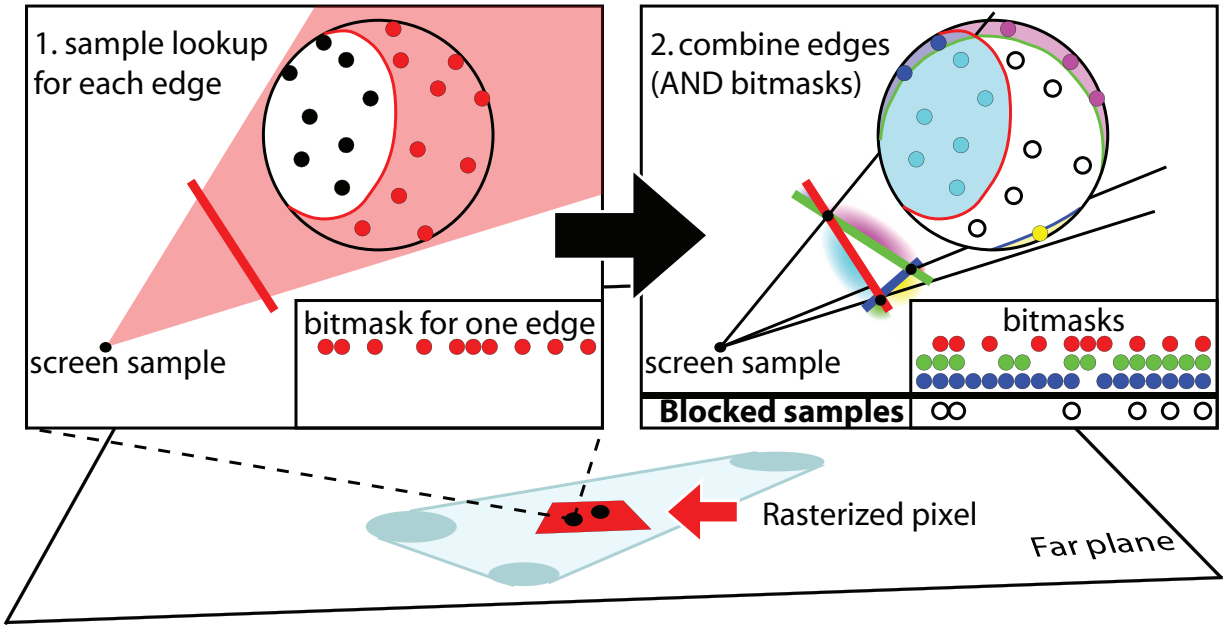


Fig. 13.6 : Computing the visibility for a volumetric source

Visibility is tested for each view-sample in each rasterized fragment (bottom). A texture look-up returns a bitmask indicating the light samples behind the plane defined by an edge and the view-sample (left). The occluded light samples are identified by AND:ing these values (right). The light samples can be arbitrarily located.

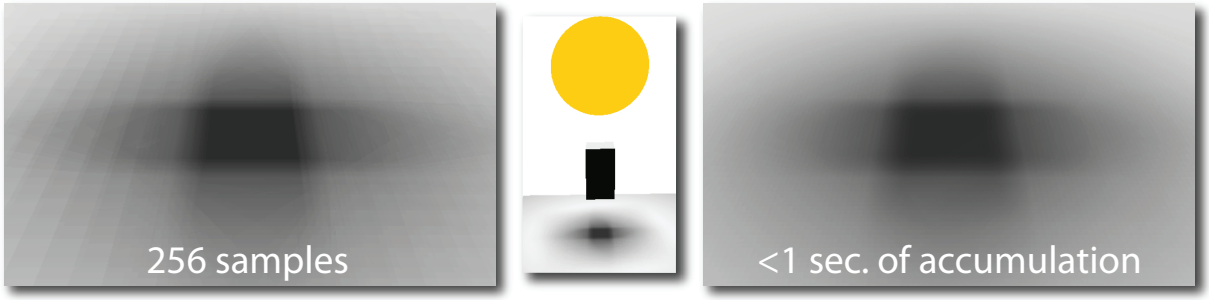


Fig. 13.7 : Temporal jittering for a converging solution

No jittering is applied on the left. The right shows the result of jittering and frame accumulation after one second. Both rely on 128 light samples.

We have seen the possibility of decorrelating source samples before. This can be done on a per-pixel basis to achieve jittering at almost no cost. We can also use temporal jittering where samples change in each time frame and successive frame are accumulated. The solution converges rapidly to a high-quality image, if the scene is static and the camera does not move (see Figure 13.7). As pointed out in [SJW07], lower quality is acceptable for moving cameras and for light designers it is useful to have a fast feedback concerning shadows.

3.3 Optimizations

Soft shadows are more expensive than hard shadows. The following optimization can avoid much of the unnecessary computations e.g., for scenes with large umbra areas cast by many small adjacent triangles.

3.3.1 Parametrization

To keep view-sample lists short, existing techniques to maximize the SM resolution (e.g., [MT04]) can be used to optimize the sample repartition. A good solution we employed is to fit the frustum to a minimal 2D-axis aligned bounding box of the projected view-samples, and we readapt the light frustum between each pass of our algorithm. The parametrization led to increases in speed of around 30% in scenes like the one in Figure 13.8. The same step also computes the maximum view-sample distance to the light source. This allows us to fit an optimal far plane. This keeps influence regions smaller, but its influence on performance is weaker.

3.3.2 Restricted Computations

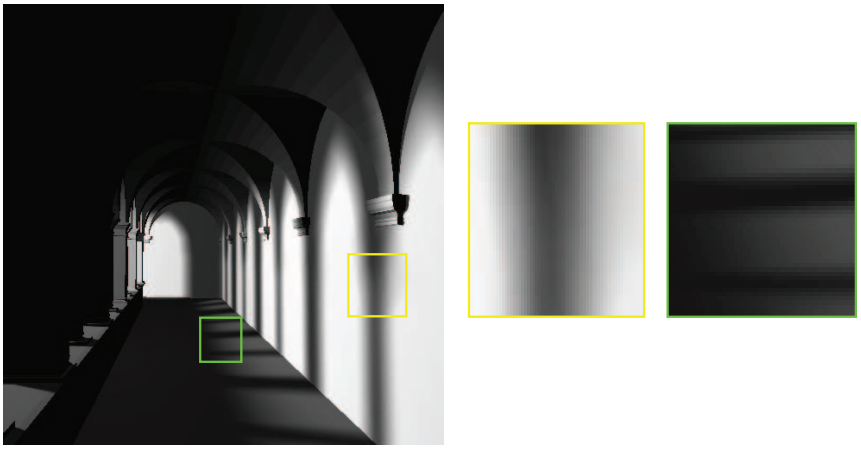


Fig. 13.8 : Sponza Atrium (73k) With Accurate Shadows - *20k shadow casting triangles for this view, 256 light samples per pixel.*

With the stencil buffer we can block all pixels with empty lists. Further, penumbrae can only be cast from silhouette edges. Here, *silhouette* means that there is a point on the light for which the edge is a silhouette. Their detection is done in the geometry shader or using CUDA by computing the relative position of the light with regard to the planes defined by the adjacent triangles [LAA*05]. The edges' influence region is then rendered into the stencil of the SM. This blocks unnecessary computations in the umbra/lit region, where a hard shadow from any sample point is enough. The same reasoning applies for triangles. If models are watertight, triangles can be eliminated based on conservative back-face culling [ED07b]; in addition we can cut off the influence region

based on the triangle's supporting plane [LLA06], which integrates well in our convex-hull computation. Both measures made the soft shadow pass win up to 50%. Another possibility would be to apply a perceptual measure, e.g., based on the frequency of standard lighting and texturing to choose only a view-sample subset. A reconstruction step yields the final image. A first step in the context of shadows was presented in [GBP07]. We did not rely on this for our timings to show the actual cost.

3.3.3 Shading

The standard Gouraud/Phong shading models use per-vertex normals for a smooth interpolant. This might lead to *light bleeding* into back-facing triangles (see figure 13.9). It introduces artifacts at light-silhouette edges because accurate shadows lead to self-shadowing and create a strong discontinuity. This is particularly disturbing for hard shadows. (We encountered a similar problem in [EWHS08]).To avoid this behavior, we bent vertex normals with respect to the adjacent faces. In the context of a real-time application, the efficient evaluation is of importance. Unfortunately, the geometry shader input *triangles with adjacency* does not provide all triangles adjacent to a vertex and thus disqualifies for this task. Nevertheless, a rapid GPU evaluation is possible by passing face normals in form of a texture and adjacent face indices as texture coordinates into the vertex shader. In case the number of neighbors is small, normals can be transferred directly in texture coordinates.

Interpolating the minimum illumination based on adjacent face normals delivers a smooth result that ensures blackness of all light back-facing triangles. Formally, we compute at each vertex: $l_{bend} = min_{F \in adj(v)} dot(l, n_F)$ where n_F is the normal of face F, l the light vector, and $adj(v)$ the adjacent faces to v. Per fragment, we can securely and continuously blend back to Phong illumination l_f, using: $\alpha := min(1, max(c\ l_{bend}, 0))$, $\alpha\ l_f + (1-\alpha)\ l_{bend}$. $c := 3$ works well. Soft shadows are smoother and thus bent normals with respect to the source's center are often sufficient. The definition could be extended to the minimum with respect to the entire source or a bounding volume (e.g., a cube).

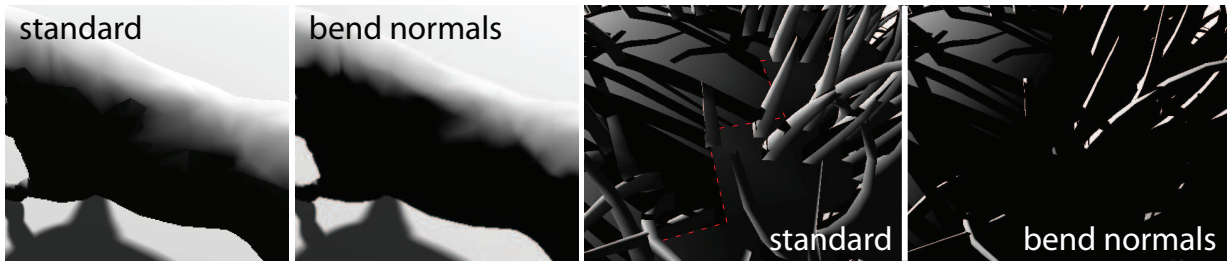


Fig. 13.9 : Normal bending to prevent light leaks

With standard vertex normals, light can leak into back-faces. This can result in a zig-zag appearance (left). Further, shadows are cut at the border of the backfaces (red line, right), this results in visible discontinuities. Bending the normals allows us to obtain a coherent shading.

Summary:

Besides the usage of the view-sample lists to allow soft shadows in general scenes, we present an algorithm that allows volumetric sources. We solved two obstacles: computing a tight influence region and the sample evaluation.

We also mentioned several optimizations that integrate the result over time, allow to block empty lists from further processing and restrict computations tighter around to the penumbra region.

Finally, we explained an efficient implementation of bend normals that remedy artifacts due to inconsistencies between shading and geometry.

4 Results

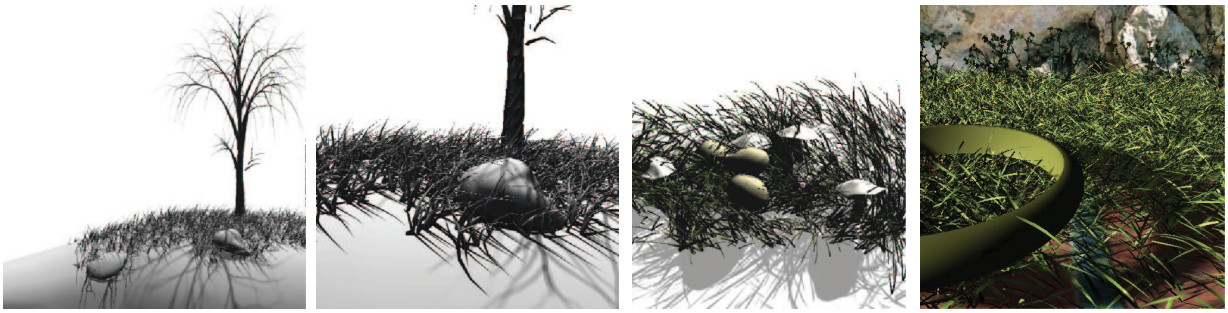


Fig. 13.10 : Complex scenes with accurate shadows

Left to right: 90k triangles (7 fps), zoomed (4.5 fps), 70k triangles (4.8 fps), ring scene: 379k triangles (0.65 fps).

All measurements were performed using 512^2 resolution and a GeForce 8800GTS-512. Five test scenes where used. A *fairy* of 734 triangles, *columns* of 840 triangles, a *hairball* of 44ktris, a *torus-knot* of varying tessellation degrees, the *Sponza Atrium* of 73k triangles, and very high polygon scenes like a ring in grass of 379ktris and others shown in Figure 13.10.

For the case of increasing samples (Figure 13.11) the rendering cost, roughly, only doubles from 128 to 1024. Our method is about an order of magnitude faster than using a corresponding amount of shadow maps, which would be an alternative for producing sample-based soft shadows on arbitrary receivers, but exhibit aliasing artifacts. The other interesting information that is depicted in Figure 13.11 summarizes the behavior of list sizes in a typical scene. The average of the list sizes is relatively short, but there are typically some outliers which becomes visible when looking at the maximum list size. These are usually view-samples that are close to the observer, where the shadow map discretization would be most visible. Better shadow map parametrization could thus be interesting for future work, although the repartition is, as the numbers show, already quite acceptable.

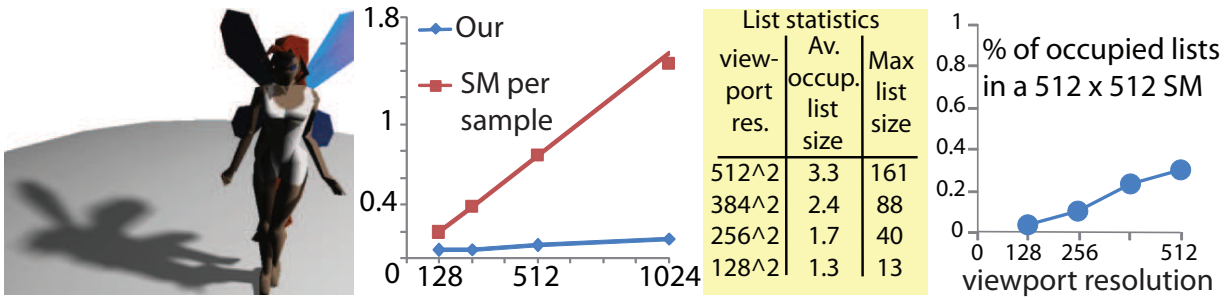


Fig. 13.11 : Performance Statistics

Comparison Our vs. Shadow Maps(left), List Statistics (right).

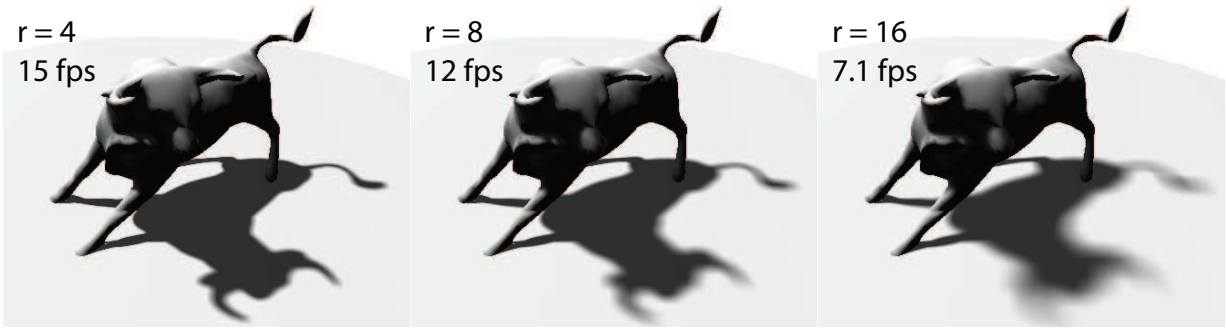


Fig. 13.12 : Performance for a varying-light radius

The larger the source, the more costly the algorithm becomes. As for the previous visibility sampling method, this is due to the increased overdraw of the influence region.

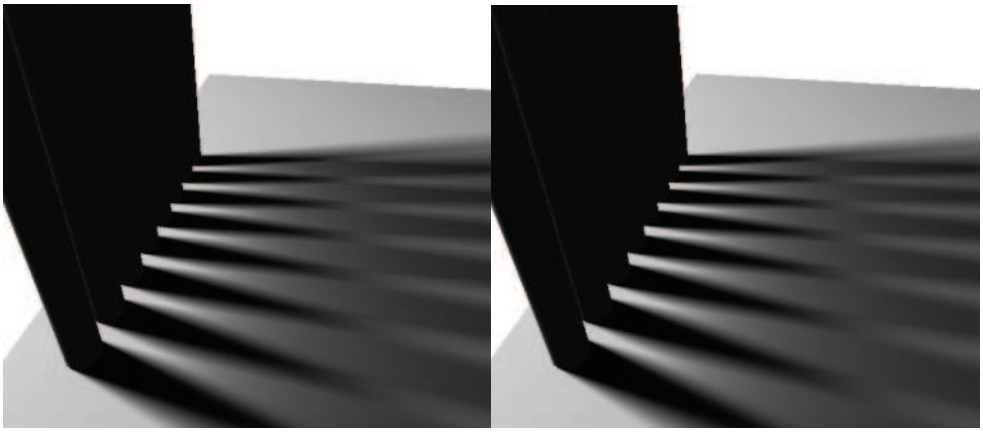


Fig. 13.13 : Shadow Quality

Comparison of 128 samples to 512 samples on a simple scene

Figure 13.12 shows a mostly linear dependence when varying the light radius. The cost mostly results from overdraw of the influence regions.

Figure 13.13 stresses the sampling quality for a large penumbra. The visual difference between 512 and 1024 samples was negligible in this case.

Figure 13.14 uses 128 light samples and varies the tessellation degree. In this case, the algorithm has mostly linear behavior in the number of triangles. Nevertheless, our optimizations (Section 3) to transfer only visible view-samples and eliminate empty lists, as well as the restriction to penumbra regions, lead to important gains.

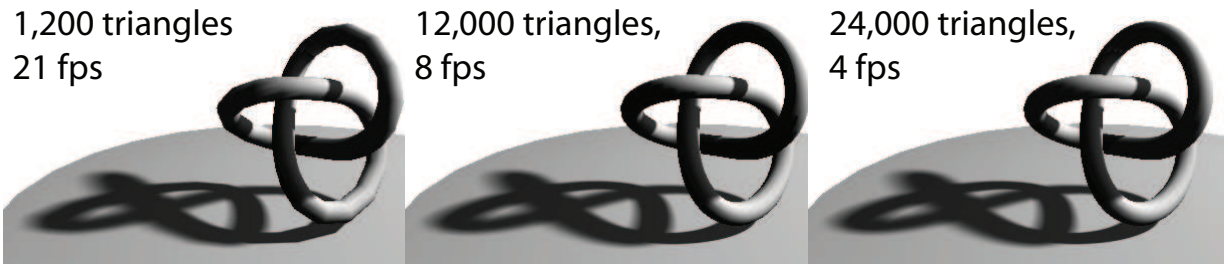


Fig. 13.14 : Varying degrees of tessellation

The torus knot illustrates the mostly linear behavior of our algorithm with respect to the geometric complexity of the scene.

The right-most image in Figure 13.10 shows a scene from [LAA*05]. The computation time using Laine et al.'s algorithm reported 75 seconds for a 512^2 image for this scene. In contrast, we achieve a frame-rate of almost constant 0.65 Hz independently of the viewpoint, which corresponds to a speed-up of 48.75.

Figure 13.8 shows the Sponza Atrium, which runs in 2.6 fps. fps compared to 9 seconds for Overbeck et al. [ORM07] with 256 light samples. We thus provide a 23.4 times speedup. Increasing the number of samples in this scene did not lead to noticeable quality improvements, but in general, Overbeck et al.'s solution is exact, not sampled like ours.

Our approach can deal with complicated scenes like the hairball (Figure 13.15). Our solution works robustly enough to obtain pixel-accurate hard- and soft shadows with high quality.

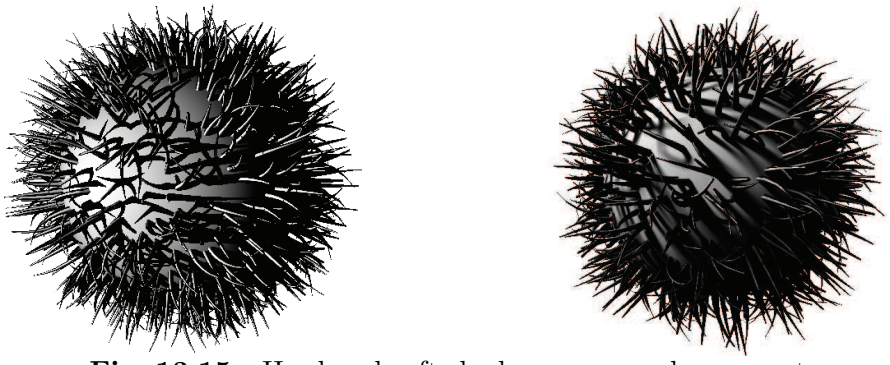


Fig. 13.15 : Hard and soft shadows on complex geometry

The hairy ball has 44k triangles. The many fine details make the computation of shadows with image-based approaches very difficult.

5 Conclusion and Future work

We presented an accurate solution to the general soft shadow sampling problem. Our soft shadow algorithm outperforms soft shadow volumes for ray tracing by 1-2 orders of magnitude, and shadow rays with 2-3 orders and has similar image quality. Our alias-free shadow maps have comparable performance to [LSO07], but the implementation effort of our solution is much lower and has virtually *infinite* resolution.

Approximate transparency is possible by multiplying the transparency values of shadow casting triangles and weighting by the percentage of coverage. For correct transparency, each sample would need to store one RGBA-value per light sample, which requires more bits than currently available.

We currently loose performance because we do the computations (back-projection of occluder, test of visibility) for points that are already found fully shadowed. An estimation of how often this occurs could be very interesting. In our experience, a tree foliage would generate a lot of this. The same for connected occluders. Read/write-techniques would also be possible because our shadow function is monotone. Currently, we can quickly reach a texture bottleneck when doing this on a per fragment basis. One solution that improves performance slightly is a sequential method with front-to-back selection of occluder batches.

Part V

Conclusion

"If we don't change direction soon, we'll end up where we're going."

Irwin Corey

Chapter 14

Conclusions

Our initial thesis statement might not sound very risky: *There is no ultimate solution to render a complex scene in real-time.* After we have seen all the different ways of addressing problems by relying on several mathematical and algorithmic means, you hopefully agree that this statement seems to hold.

Nevertheless, we added a precision that is very specific. We proposed a nomenclature that categorizes algorithms according to five different fields: *Preprocessing, Transformation, Representation, Structuring and Restatement.* Our contributions fall into each of these categories showing that they are all necessary in a mathematical sense, and our work illustrates the advantage of an organization in such a way.

In the introduction, we talked about the major difficulties that come with complexity. We showed that our approaches could address many of them. The major observation was that providing global information about the scene is one of the most challenging problems to deal with. It is hard to capture information from the entirety of an environment that is dynamically changing and to perform computations that relate elements with each together. In this dissertation, we have seen many solutions that address the problem of extracting and providing non-local information.

We concentrated on a variety of areas ranging from realistic rendering (simplification, shadows and other phenomena, ...), over collision detection to non-photorealistic rendering (clip art, stylization, grouping for guided abstraction). In all works, we either took complex problems and provided efficient solutions, or aimed at the reduction of complexity in the final output. In all cases, we delivered practical and working methods.

We thus consider our goal achieved, but much remains to be done.

1 We Are not There yet...

This might sound frustrating after having fought your way through all these pages and basically three and a half years of work.

Much of the particular future research that is related directly to our contributions was already mentioned in the according chapters, so we will not repeat this here. We rather like to focus on some more general points related to our global view of this dissertation that we would like to expose. So far, our contributions can be seen as a measurement in the scene that we use to

derive a more appropriate representation of the data that we then involve in our computations. Consequently, the next step would go from measuring to influencing.

Imagine the situation where a volcano explodes and lava is running down the hill. Obstacles will interact in different ways. Stones will probably redirect the stream and, only if the pressure becomes strong enough, they will be pushed aside. Trees, on the other hand, will capture fire and burn. Also, dynamic objects might be concerned too. It would be of interest to investigate how interactions with the environment can be integrated. For example, to add obstacles and change the flow dynamically. Such a situation is extremely challenging because interaction needs to be detected and the according forces fed back to the scene itself. There is no straightforward solution that would allow to achieve this goal. New adapted data structures and algorithms will be necessary.

Another example in this direction that suffers from dynamism is simplification. Currently, very little work seems to exist for animated scenes, even though taking motion blur, speed, glares and deformations into account would enable a more aggressive simplification. This probably comes from the fact that precomputation is not easily possible anymore because the character animation could include arbitrary poses (e.g., ragdoll animation). One way that might allow us to deal with this situation is to extract coarse amounts of data that is then used to drive a simplification process. A first approach in this direction was presented in [DBD*07], but a strong interaction with the CPU remained and no error bounds could be achieved. There are also steps towards simplification algorithms that are entirely executed on the GPU [DT07]. For the moment, these are not fast enough to be applied on a per frame basis and represent basically a mapping of CPU methods onto the GPU. In the future, it is important to find better structures that we can really adapt to the needed detail level on the GPU. ATI recently added a tessellation unit to their GPUs that could be helpful in this context[1].

An important related topic is quality scalability. This is currently a hot topic for GPU computations (e.g., soft shadows [GBP07, SS08]), nevertheless, it remains to be seen whether we can find such criteria for other fields too. This would lead to an improved control over the result. Especially a more general solution for LODs of alternative representations would be of interest. Approaches so far always relied on the very particular representation.

This is important because alternative representations are really emerging, but they still show several general problems. Aliasing, for example, is already hard to deal with for polygonal models, and can be even more involved if complex shaders produce the output. Deferred shading or early z-Culling are usually not an option for alternative representations because the extraction of this data is usually what governs the rendering times, or they do not represent any geometric data and solely provide a visual approximation. Furthermore, their animation is a big issue because often precomputed acceleration structures are the basis of the performance gain. Because of many shortcomings like these, we are likely to be stuck with shaded triangles for a while. In particular because existing alternative representations often need a lot of memory compared to triangular approximations. This is will especially be an issue for large scenes. It is currently a common trend to expand the universe around a player and this new freedom is reflected in many productions.

[1]This technique is supported by hardware for more than a year now, but ATI did not give access to this functionality, but it is announced for a later DX release

For GTA 4, no less than 100 Million Dollars (!) have been spent solely on content creation (this sounds even more impressive if one considers that a game like E.T. had used a 25 Million Dollar licence and resulted in a game made by a handful of people in six weeks). An interesting avenue for research is to provide the corresponding creation tools. In this context, even though this thesis only scratches the surface, we believe that methods, like those we described, are of real interest in the design process. In the extreme case, the creation of content itself could be automated and only loosely predefined with some rules (a little similar to the grouping behaviors that are created beforehand and then applied to a dynamic scene [BEDT08]). The other direction was to support the creator. Visibility was one example, but it does not have to stop there. Future solutions might directly involve the artist in the optimization process. For example, imagine a complex character that will mostly remain in the dark, thus hiding much of its details. The system could detect this and propose alternatives that provide a visual equivalence or show possible approximations including the expected gain in performance. The same could be used for materials, animations and many other properties of the object itself, or for light placement, where indications are given on how complex shadow casters need to be for a certain light and receiver position. Another major challenge in real-time rendering are glossy or reflecting models. Only few of them can be integrated in a scene and a system might help in deciding which positions should be avoided because they are more costly due to a potentially higher amount of (dynamic) objects that will need to be reflected. All these tedious tasks that usually rely on a trial- and error principle that involves many users and possibilities should be performed by the computer instead.

No matter which road we take to produce and display content, ultimately we aim at the production of images. One question here is whether pixel-images will remain our goal. Vector representations are really becoming popular: on Windows Vista, all icons are usually stored in a very clean vectorial SVG format. We provided algorithms to extract such a representation in this thesis. Vector graphics are adapted to many devices, but what are the devices of the future, and what would be an adapted representation for these? For example, I personally think that 3D cinematography could soon become more popular. The difficulties we are facing in this context are very different from those in standard image synthesis. New representations will become necessary and probably also new rendering techniques. In this situation, complexity for an automated process increases because there is not only a dependency between several scene elements, but also between both views. If the interdependence of the views is neglected, many problems can appear. This was the case for the production team of the 3D Polar Express (Castle Rock Entertainment, 2004). Specularities that are present in the image of one eye and not in the other result in a disturbing flickering. Alternative representations like simple billboards might no longer work because they miss the depth information. Artificial smoke did not work in their tests when simply rendered from two views.

In general, the way we perceive three dimensional objects is extremely fascinating. The devil's triangle is a drawn illusion that is not possible to construct in reality in the way we perceive it. In fact, there is only a way to construct a shape that seen from one particular point of view, looks like the drawn illusion (see figure 14.1). I recently got the chance to use 3D goggles displaying a virtual version of this construct. Interestingly, the brain constantly fights with the perceived depth because of the disparity between the two eyes and the perceived shape. One did not dominate the other and both possibilities alternated.

In general, our brain adds much more 3D shape to a scene than what could be extracted based on pure disparity (otherwise we would probably not even be able to interpret a drawn image as a

3D scene). This shows the freedom that potentially remains when producing stereo-images, but also illustrates how rigid some of our nature dictated interpretations are.

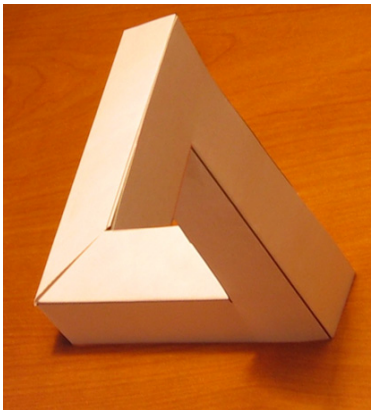


Fig. 14.1 : *A Built Version of the Devil's Triangle (Source unknown)*

Another remark in this context concerns materials. It seems that specularities and reflections help us significantly in evaluating an object's shape. On the other hand, and this is somewhat contradictory, the reflected elements themselves are less important (compare [AMH02] and generally the coarse approximations made with environment mapping). Finding these cues, evaluating their importance and steering them could ultimately give us the possibility to produce more accessible, convincing and powerful images. For example, you probably did not notice the shadow in figure 14.1. You noticed there is one, so you had just enough information to understand that the object is on the table, but if you had looked more closely, you would have realized that the shadow reveals the construction and contains the data to solve this perspective mystery. Extracting important information from an image and reducing its complexity to deliver a cleaner and more focused representation is a very interesting goal.

These are only some directions that need further exploration and that fall into the context of this document. Many more lie ahead of us and it seems the more insight we gain, the more new challenges appear.

2 Last Words

After all this effort, I decided to add this last section mostly for me to *add my two cents* and really conclude this thesis by taking a look back over the years.

Early on in my PhD, we started working on visibility and simplification. Our first contribution was a mathematically sound blocked ray definition, and an out-of-core visibility method. Later, a voxel based visibility culling, and alternative visibility parameterizations. Finally, I decided to not include any of these projects in this dissertation because I wanted to keep it at least as short as possible (even though Xavier is probably still disappointed, but I know that whoever read up to here already went through a lot...). One thing that occurred to me in this context was that visibility is really hard. It has many hard cases and our code was quite involved. In contrast, our result on visibility sampling delivered "only" an approximation, but a very good, simple and stable one.

This holds for our voxelization too: One billion voxels is a really large amount. Obtaining such a resolution at 100Hz is almost incredible, even more voxels than people on earth can be achieved at more than real-time rates. I know, I might be one of a small group of people that are *really* excited about our follow-up paper (also Chapter 7), but for me, it finally concludes binary voxelization and will be hard to beat in any near future. It addresses all issues that might have remained in our first paper. In my eyes, it is not even important that it outperforms previous methods, it is important that one can feel that it is the right way of addressing this problem.

We always wanted to finalize the combination of voxelization and visibility, which would lead to an even stronger fusion of all our contributions. Conversely, we would not have had the time to spend on soft shadows that attracted me particularly. Soft shadows are naturally entwined with visibility. The definition of the problem is so simple and solving it so hard. The very high performance of our first method came almost as a surprise. When we published the first paper, we did not optimize it a lot and it already ran at around 30-40 Hz. Although this rate was very acceptable for a soft shadow algorithm, one of the reviewers pointed out that, in real applications, 40 Hz are not enough and 80 Hz would be better. He was right and this barrier was really bugging me because I felt like our algorithm should get close to it. So, we spent some time on optimizations and, when we finally presented the paper at the conference, we reached 90 Hz on the very same hardware. However, the solution was approximate, and finding a more accurate approach was thus a motivating factor. We finally achieved this goal with our algorithms on visibility sampling and alias-free soft shadows. It makes me particularly happy that it lead to acceptable performance despite the accuracy. Further, it allows us to somewhat wrap up the work on this topic.

The effort we spent on optimizations for the shadow paper was not comparable with the clip art work. Coding the system took quite long and the reviews we received varied from "How often does it cause a segmentation fault" to "I hope I will never have to code it myself." Even though it was a huge amount of work, which is sad because most of the initial implementations were well-known concepts, I think we can be more than happy with the result. I really fell in love with the project when we finally produced the first stylized clip art. The program is yet far from shipping code, but I can see why people would use a commercial solution. Further, it motivated us (Xavier and me) to explore other NPR techniques and, in collaboration with Hedlena and Joëlle, we proposed a new way of stylization, what I found quite exciting.

When writing these lines, I realize that after all, I really like what I am doing (despite the fact that writing a dissertation is stressful and makes you forget about it). It is surprising that it was almost luck that made me end up in graphics.

When I just arrived at ENS, all students were already asked to leave... no joke... right in the beginning we were informed that during the summer we were to do internships at a university or in the private sector to gain experience and get in contact with research. When the different topics were proposed, I was directly attracted to the one called *images and virtual reality*. I had done a little bit of graphics before, after all I had a computer, but usually it had been for fun or in a game context. In my mind, it seemed tempting to combine mathematics with visual creations. Therefore, when asked about what my research topic is going to be, I said: *Images AND virtual reality* (I had the feeling that virtual reality sounded somewhat more serious...) Now, years later, I know how difficult it is to "create images". The fact that we can relatively easily describe our problems to a large public makes the entire topic even more attracting. Visuals are just everywhere around us, not unlike mathematics, but much more explicit and easily observable for everybody. Even though we might see things a little different from others: who else would wonder about indirect lighting when looking at at tree, observe clouds carefully to see how much detail is perceivable and how *wrong* they look, or try to produce quadric-shaped umbrae with fingers and a desk lamp?

The first quotation in this dissertation stated "It's a small world, but I wouldn't want to paint it." Well, we still keep trying, although chances are high that we will never get there... but we make one step forward everyday. Let's see how small this world is!

Part VI

Appendix

"God separated the light from the darkness."

Genesis 1:4

Appendix A

Penumbra Regions

The shape of a penumbral region can be complex to compute. Stark et al. [SCLR99] showed that shadows can be described with special spline functions that are derived from a multidimensional combination of a polygonal blocker and emitter in the presence of a receiver plane. Nevertheless, the source is assumed planar and the occluder too.

In this chapter we will investigate the shape of the penumbra region in the presence of a spherical source. This is interesting for our shadow algorithms described in chapters 12 and 13. We will derive an analytical description of the shape of the penumbra region in the case of a spherical light source. Even though, we finally resorted to a simpler and less accurate solution in Chapter 13, it could be of interest to have this solution for reference and it might prove more efficient in the future.

The proof is relatively simple and uses very basic calculus. It is not necessary for the understanding of our contribution. It was included for completeness reasons.

1 Penumbra Region Determination

We start by showing the convexity of the penumbra region. This allows us to define the penumbra region as the convex hull of the light source projections through the corners of the triangular occluder. For each corner the light source appears as a circle. The projections of these circles form ellipses in the plane. We first derive a way to compute this equivalent circle light for a corner of the occluder. Then we show how to compute the ellipse's axis of the projection and finally the size of the ellipse. In a last step we explain how to find the tangent points at these ellipses defined by the sides of the convex hull. This is sufficient information to construct the final accurate shape of the penumbra region.

Convexity

We will start with a first simple observation.

The penumbra region of a convex occluder in the presence of a volumetric and convex source onto a planar receiver is convex.

Proof: Without loss of generality we can assume that the light is not behind the plane defined by the triangle. If this is the case, we will cut the light along the triangle's plane $\boldsymbol{T}$ and pursue

our proof for both halves and the two triangle orientations separately. This is valid due to the common borderline which is the intersection of the triangle plane with the ground. It is easy to show that the penumbra is locally a two-manifold with border, due to the definition of being the union of all projected triangles. Let's suppose the penumbra region $\mathcal{P}$ is not convex. This means, that we can assume that there are $P_1, P_2 \in \mathcal{P}^O$ (where $\mathcal{P}^O$ denotes the interior of $\mathcal{P}$) with $P_1 \neq P_2$, but $\exists P_3 \in [P_1, P_2] : P_3 \notin \mathcal{P}$. Due to our choice to cut the light source along the triangle's plane T, there cannot be any intersection between (P_1, P_2) and the supporting plane of T. $P_3 \notin \mathcal{P}$ implies the existence of a separating plane P between the source and the occluder passing through P_3. If P contains $[P_1, P_2]$, then the problem is 2D and trivial. Otherwise we translate P along $[P_1, P_2]$ in direction of the source. The moment that P becomes tangent, implies that all the points of the segment $[P_1, P_2]$ that lie on the same side of the source with respect to P will be completely lit, contradicting the fact that both extremities P_1, P_2 are in penumbra.

Convex Hull of Projections

The penumbra region is given by the convex hull of the source's projection through the corners of the triangle.

Proof: Let P be a point on the boundary of the penumbra region. Thus there is a separating plane that is tangent to the occluder (triangle) and the source. This is easy to show, but also a known fact [AAM03]. The tangent plane at the triangle can pass either through one of its vertices or a side. In the first case, it means that P also lies in the projection through the corner, which is part of the convex hull of all projections through the corners. Thus, let's assume the plane is tangent to a side E of the triangle. Let S be the tangent point on the source. Now the projection of E from S on the plane contains P and is linear. Because the extremities of E correspond to the corners of the triangle, P needs to lie in the convex hull of the corner projections.

Deriving the Relative Light Size

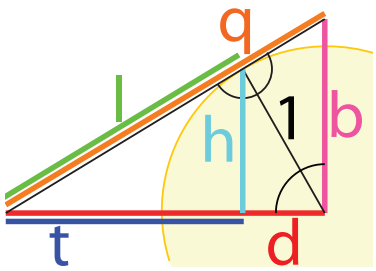


Fig. A.1 : *Notations*

Instead of considering a spherical source, it is easier to consider discs of appropriate radius for each corner of the triangle. The first step is to derive this radius for a given point.

Please follow the notations in figure A.1.

$$l = \sqrt{d^2 - 1}$$

$$t^2 + h^2 = l^2 = d^2 - 1 \tag{A.1}$$

$$(d - t)^2 + h^2 = 1 \tag{A.2}$$

Combining A.1 and A.2:

$$t = d - 1/d$$

Consequently:

$$h = \sqrt{1 - 1/d^2}$$

To ease the following we calculate b, the apparent disc radius. This is done by starting with the linear function $g(x) = x * h/t + b$ and the condition that $g(-t) = 0$, we obtain $b = d/\sqrt{d^2 - 1}$. Therefore instead of considering a sphere, we can think of it as a disc of radius b.

Deriving the Projected Ellipses

Now that we have verified that the penumbra region is the convex hull of the projections of the source through the corners, and we have seen how to replace, for a given corner the spherical with an equivalent disc source, we will now concentrate on computing the projection of the source through a vertex. This time one should follow the notations of figure A.2.

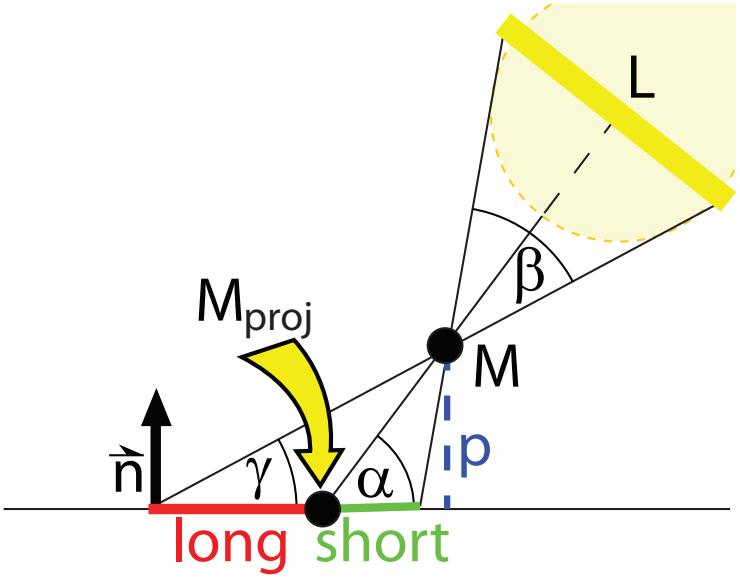


Fig. A.2 : *Notation for the extent (red) that we solve for.*

It is easy to show that the axes are given by $\vec{a} = \vec{n} \times (L - M)$ and $\vec{b} = \vec{a} \times \vec{n}$, where *times* denotes the cross-product, $\vec{n}$ is the normal of the receiver plane, L the light position and M the vertex we consider. We now derive the extent of these axes. The extent in direction $\vec{a}$ can be recovered from a simple relationship, once we establish the extent along $\vec{b}$.

Following the figure A.2, we see, that the extent is given by $p * (cot(\alpha) - cot(\gamma))$ where $\gamma = \alpha - \beta$. Interestingly this equation can be rewritten to an expression in α and $\cot\beta$. The latter being simple $t/h = \sqrt{d^2 - 1}$ following figure A.1. We obtain:

$$long := d(cot(\alpha) - cot(\alpha - \beta))$$

for the other segment next to it, we have

$$short := d(cot(\alpha + \beta) - cot(\alpha))$$

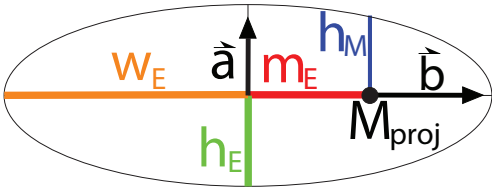


Fig. A.3 : *Deriving the correct extent along $\vec{a}$*

Once we have the length of these two segments, the width of the ellipse is thus given by $w_E = (long + short)/2$. It is also simple to find the center of the ellipse.

All that is now missing to complete the ellipse definition is the extent along the vector $\vec{a}$. Only at the projection point of M (M_{proj}) the distance along $\vec{a}$ is a simple ratio of distances and light size, but this point does not coincide with the center of the ellipse as illustrated in figure A.3.

At the projection point, with notations of figureA.1, we have the relation:

$$\frac{b}{d} = \frac{h_M}{p_{proj}},$$

where h_M is the height of the ellipse along $\vec{a}$ at M_{proj} the projection of M on the plane and $p_{proj} := ||M - M_{proj}||$ is the euclidian distance from M to the projection of M on the plane.

As mentioned before, h_M is *not* the extent of the ellipse, because in general M_{proj} is *not* the center of the ellipse. We will now derive the real height height of the ellipse h_E. The relative position of M_{proj} with respect to the center of the ellipse is $m_E := long - (long + short)/2$. Now this point has to be on the ellipse $((\frac{x}{w_E})^2 + (\frac{y}{h_E})^2 = 1)$, thus the following equation needs to hold:

$$(\frac{m_E}{w_E})^2 + (\frac{h_M}{h_E})^2 = 1$$

Solving this equation gives us the real height:

$$h_E = \sqrt{1 - (\frac{m_E}{w_E})^2} h_M$$

This solves the entire problem of finding the ellipse.

Covering your Penumbra Region

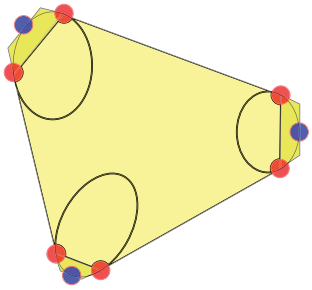


Fig. A.4 : *Covered Penumbra Region*

The proof for convexity actually implies something more: The linear sides of the convex hull are given by the tangent planes through the edges of the triangle. These penumbra wedges [AAM03] are defined by a source tangential point S and a triangle edge $E = (M_1, M_2)$. The intersection of this plane with the receiver yields the tangent line of the convex hull. The tangential points are even easier to find, they are nothing else but the projection of E's extremities from S.

An efficient way to compute the ellipse's tangent points is to consider a coordinate system where $M_1 - M_2$ is one axis. Then the problem becomes two dimensional. The same calculation as in figure A.1 applies (realize that $M1$ and $M2$ would coincide in this figure). For this we need a plane passing through M_1, M_2 and being orthogonal to $M - L$. One solution is to compute $(M_1 - M_2) \times ((M_1 - M_2) \times (M_1 - L))$. It gives us a normal for the plane, that allows us to establish the needed value d. Once all the values are computed, the tangent point can be found using the vectors $\vec{n}$ and $\vec{n} \times (M_1 - M_2)$.

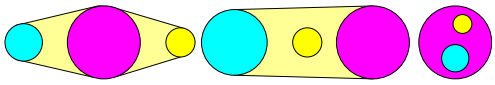


Fig. A.5 : *Other Non-Standard Cases*

With this information it is finally possible to define the entire shape of the influence region as shown in the image A.4. Parts of the shape are elliptical (which can be tested against the discovered equation in the fragment shader), whereas the body is simply polygonal. In practice, when putting all the formulae together, there are many possible improvements that we integrated in the implementation, but these are out of the scope of this chapter and mostly technical in nature. One more important cache is that not always all three vertices have an impact on the shape. There are three more cases, depicted in Figure A.5, that can occur. The test for these is relatively simple, when sorting the three vertices by height.

"The supreme accomplishment is to blur the line [between work and play]."

Arnold Toynbee

Appendix B

Undersampled Image-based Anti-aliasing

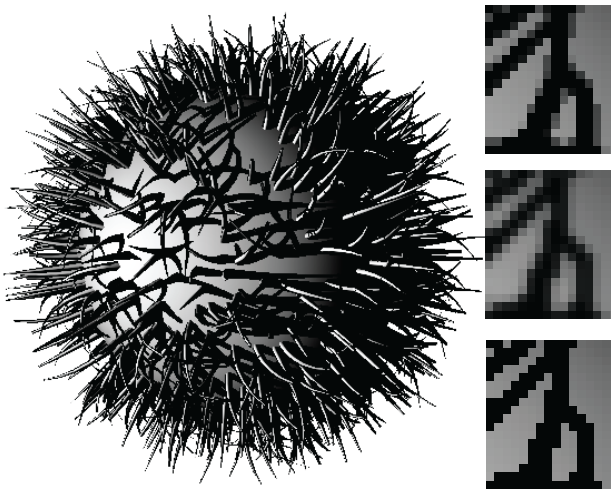


Fig. B.1 : Anti-Aliasing Comparison - *Supersampling (top), our (middle), no (bottom)*

In this chapter we present an image-based approach to perform fast anti-aliasing. The algorithm runs entirely on graphics hardware.

Aliasing is a result of sampling. The sub-pixel information, that is the percentage of a pixel that is occupied is usually not taken into account.

A simple solution consists in creating a much larger image and then shrink it to screen resolution, which is to some extent what basically happens during super-sampling in graphics hardware. Obviously this is a very expensive technique concerning fill rate and memory. In particular a lot of work is spent on textures although this problem is addressed by automatic filtering. Nevertheless, it is true that it is still an important and yet not entirely solved problem but leading to less obvious artifacts.

Newer graphics cards are capable of supersampling small regions around discontinuity edges. This still results in more memory costs and also increased fill-rate. The main reason what makes it problematic is the almost incompatibility with deferred shading.

Our approach exploits the idea, to correct only those regions close to discontinuity edges, but we will not perform supersampling. Instead, we consider the information at the current resolution. This is surprisingly often giving an acceptable result.

Implementation: *The image B.1 has been obtained offline. The approach has not yet been integrated into the original software. It is currently running in a separate application.*

Our alias-free shadows in Chapter 13, benefit from anti-aliasing although this may sound contradictive. This is because there are two kinds of aliasing in shadow mapping. One is due to the limited texture resolution and leads to stair-stepping artifacts, which is the one we addressed in our work. The second is due to the fact that each view-sample actually represents a volume in space, not a ray. Thus a view-sample can lie partially in shadow. The shadow boundary does therefore not directly vary from black to white, even though the center sample-based evaluation suggests so.

1 Aliasing

Computer graphics tackled the problem of aliasing for a long time and following Hoppe et al. [SHSG01] there are four different types.

- **Aliasing in the interior of triangles** come from shading or textures. The texture problem can be addressed, but the high frequencies in shading are still a current research topic. (First steps in this direction can be found [HSRG07].)
- **Aliasing at discontinuities**, typically triangle edges. These can occur in form of different materials, different normals or silhouette edges.
- **Aliasing due to sub-pixel sized triangles.** Which usually are dealt with by supersampling, level-of-detail control or alternative representations. For example, mip-mapped Billboard Clouds [DDSD03] may result in a higher image quality than drawing the actual geometry that contains too many details and thus produces high frequency output images.
- **Aliasing due to triangle intersections** are rather rare and dealt with again using supersampling.

We will only focus on aliasing at discontinuities. In the meantime [Shi06], a similar method was presented that relies on a single framebuffer image to address the aliasing. They also apply a filtering pass to help with this problem. We will therefore only focus on the novel contributions. Namely, we will investigate what filter should be used to achieve a good approximation. Interestingly, the intuition of the authors in [Shi06] was very good, as the Gaussian filter we derive coincides with their choice. The second question is on how to evaluate this gaussian efficiently. Even though we thought the latter is a well-known technique, we could not find a reference and discussions with several researchers revealed that it might be less known than initially thought, which motivated us to mention it here.

2 Filter Derivation

Obviously applying a filter to the resulting image where we encounter discontinuities, will not remedy the fact that subpixel information misses, but a simple theoretical consideration shows that the use of a Gaussian filter is reasonable.

We will consider the 1D case and take a look at three consecutive pixels on the screen P_1, P_2, P_3 with colors $C(P_i)$. The segment occupied by these pixels correspond to a set of rays emitted from the viewpoint and passing through the corresponding positions on the near plane. Let S be a sample function of all these rays that associates a color which, for a given ray, is the one defined by the impact position in the scene. Let's assume S is linearly parameterized and represents $S(-1) = C(P_1)$, $S(0) = C(P_2)$,$S(1) = C(P_2)$. In other words, the color of each pixel only represents a point sample of the function S, which corresponds to a ray passing through the center of the pixel.

The correct anti-aliased value for pixel P_2 would therefore be:

$$C_{AA}(P_2) := \int_{-0.5,0.5} S(t)dt$$

We do not want to use more information than given by the three pixel values. Therefore, we will impose certain conditions on S. We will assume that W is a step function of values $C(P_1), C(P_2), C(P_3)$. The unknown is when value $C(P_1)$ changes to $C(P_2)$ and when $C(P_2)$ changes to $C(P_3)$.

Let's assume we have $p < q \in [-0.5, 0.5]$ such that $Im(S_{|[-0.5,p)}) = \{C(P_1)\}$, $Im(S_{|(p,q)}) = \{C(P_2)\}$ and $Im(S_{|(q,0.5)}) = \{C(P_3)\}$, thus:

$$C_{AA} := \int_{-0.5,0.5} S(t)dt = (0.5+p)*C(P_1) + (q-p)C(P_2) + (0.5+q)C(P_3)$$

To simplify we can perform a reparametrization and assume an interval $[0,1]$, with a slight abuse of notation, we continue calling the function S. We still do not know where the steps p, q are placed in S. Let us further denote $S_{k,l}$ the function that has the steps at k, l. A good candidate for C_{AA}, would minimize the difference of all possible step functions. Therefore we are interested in

$$\begin{aligned} &argmin_{C_{AA}} ||S - C_{AA}|| \\ &\qquad := \quad argmin_{C_{AA}} \int_{l\in[0,1]} \int_{k\in(0,l)} (S_{k,k+l} - C_{AA})^2 \\ &\qquad = \quad argmin_{C_{AA}} \int_{l\in[0,1]} \int_{k\in(0,l)} (k*C_{P_1} + t*C_{P_2} + (1-k)*C_{P_3} - C_{AA})^2 dkdl \end{aligned}$$

This L_2-norm is known to be minimized by the mean, but this can also be shown easily: The minimum is the root of

$$dC_{AA}||S - C_{AA}|| = -2*\int (k*C_{P_1} + t*C_{P_2} + (1-k)*C_{P_3} - C_{AA})dkdt$$

Solving this we obtain:

$$\begin{aligned} & \textstyle\int_{l\in[0,1]} \int_{k\in(0,l)} k*C_{P_1} + t*C_{P_2} + (1-k)*C_{P_3} - C_{AA} dkdt = 0 \\ \Leftrightarrow \quad & \textstyle\int_{l\in[0,1]} (1-t)^2/2*C_{P_1} + (1-t)*t*C_{P_2} + (1-k)*C_{P_3} - C_{AA} dt = 0 \\ \Leftrightarrow \quad & \textstyle\int_{l\in[0,1]} (1-t)^2/2*C_{P_1} + (1-t)*t*C_{P_2} + (1-k)*C_{P_3} - C_{AA} dt = 0 \\ \Leftrightarrow \quad & 1/4*C(P_1) + 1/2*C(P_2) + 1/4C(P_3) = C_{AA} \end{aligned}$$

This justifies the use a Gaussian filter across the discontinuity. The advantage of a Gaussian is that it can be implemented effectively on graphics hardware. Only two texture lookups are enough to obtain a 1D Gaussian of size three and a full $3x3$-matrix needs only four. The trick is to sample the texture at the corners of the center pixel and to activate linear filtering. It can be easily verified, that the average of the resulting samples corresponds to the evaluation of a Gaussian filter. It allows for filter kernel separation and because successive Gaussian filters lead to a Gaussian of larger scale, this also works for larger supports.

"The ships hung in the sky in much the same way that bricks don't."

Douglas Adams

Appendix C

Equivalence of the Plantinga/Vegter and the Cippola/Giblin System

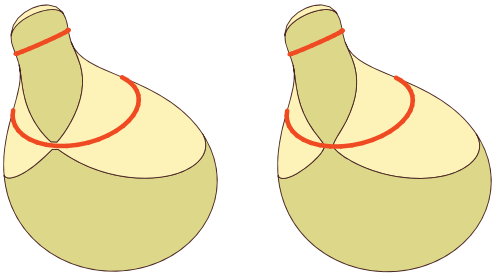


Fig. C.1 : Shadow contours on the squash passing through a critical point - *The critical points of the shadow contours are also parabolic points because they correspond to silhouettes as seen from the light.*

In this short chapter we will investigate the equivalence between the contour tracking systems defined in [PV06] by Plantinga and Vegter and the system of equations given by Cippola and Giblin [CG00]. This was mentioned in our work [SEH08], but a proof was not provided due to space limitations. Both track critical points where the silhouette changes topology.

The equations by Platinga/Vegter are formulated to find critical points for an orthogonal projection along the z-axis:

$$F(x) = 0 \tag{C.1}$$
$$F_z(x) = 0 \tag{C.2}$$
$$f_{zz} = 0 \tag{C.3}$$
$$f_x f_{yz} - f_y f_{xz} = 0 \tag{C.4}$$

The (slightly modified) equations from Cippola and Giblin are:

$$F(x) = 0 \tag{C.5}$$
$$F_v(x) = 0 \tag{C.6}$$
$$\kappa_r(v) = 0 \tag{C.7}$$
$$\kappa_g = 0 \tag{C.8}$$

We will shortly discuss the properties of both equation systems and provide two proofs that shows the equivalence in the context of an orthographic projection.

Notice: *The work in this chapter was finished with and initiated by Dr. Matei Stroila and Prof. John C. Hart from the University of Illinois.*

1 Introduction

Tracking topology of the contour generator can be interesting for several purposes. On the one hand it enables our particle curve adhesion to be chained only once and then tracked over time while the camera is moving. Although, in practice, this currently proves less efficient than chaining in each time step. The results could also be interesting in the context of silhouette stylization. Here, changes in topology can directly affect the parametrization of silhouette contours and strongly influence their lengths. Having a possibility of predicting these changes would allow an adaptation of the style to achieve a continuous evolution.

Before talking about the equivalence, let's illustrate the notations a little more: $F_v(x) =< (\nabla F(x))|v >$ which is the derivative in direction v. The normal at an iso-surface is given by the gradient of the implicit function. Thus this term states that x lies on the silhouette of the iso-surface F. We also denote $\nabla F(x) = (f_x, f_y, f_z)$, where we make use of the common notation simplification to leave out the dependence on x (compare [CG00]). Similarly, the components of the Hessian of F are written as f_ij, with $i, j \in x, y, z$. κ_r denotes the radial curvature, which is view-dependent and κ_g the Gaussian curvature, which is view-independent.

There are two major differences between these two systems of equations. Cipolla and Giblin support general projections, whereas Plantinga and Vegter rely on an orthographic camera. Furthermore, the Cipolla and Giblin system contains only two instead of three view-dependent equations. This allows, in the case of a static model and a moving camera to precompute many values and simply interrogate the representation.

We already mentioned in [SEH08] that the critical points can only occur at parabolic points. These parabolic points are exactly those with a zero Gaussian curvature. They can thus be connected to extract corresponding curves on which all topology events have to occur. This is illustrated in figure C.1.

2 Equivalence Proof

In a first step, let us explain our modification to Equation C.7. The original formulation stated that the view ray should be asymptotic (which means that the sectional curvature equals 0). Because radial curvature equals sectional curvature for tangent vectors (silhouette), the modification is valid.

In fact, we will provide two proofs, one with an operator and one with standard notations. Operator notations allow a condensed proof, but might be less familiar to the reader.

In the following we will assume an orthographic camera, thus we will impose $v = (0,0,1)^t$. We leave the *demanding* proof of the equivalence of Equations C.1, C.5 and C.2, C.6 to the motivated reader...

2.1 Proof I

We will observe a surface locally at a point of the silhouette (thus fulfilling Equations C.1,C.2). Translation to the origin and rotation around the z-axis do not influence Equation C.3. Because

the vector $(0,0,1)^t$ lies in the tangent plane (eq. C.2), it is possible to turn the surface in such a way that the tangent plane coincides with the y,z-Plane.

Under these conditions we can represent the surface locally in the special Monge form:

$$m(y,z) = (\frac{1}{2}(ay^2 + 2byz + cz^2) + h.o.t, y, z), \tag{C.9}$$

where h.o.t. denotes higher order terms.

Proof: First we can parameterize the surface with y,z because the gradient and thus the normal points along direction z. Thus the implicit function theorem gives us a mapping that only depends on y and z. Realize that we can always represent a manifold locally by its Taylor terms up to degree two ($Taylor2$) and the rest of higher order. Then the application $\hat{m} : \mathbb{R}^2 \mapsto R^3, (y,z) \rightarrow (m(y,z) = Taylor2(y,z) + h.o.t., y, z)$ is a map of the surface. Because of the translation to the origin the zero degree terms vanish. The derivatives of this map give the spanning vectors of the tangent plane, thus we have necessarily: $\frac{\partial Taylor2}{\partial x}(0,0) = \frac{\partial Taylor2}{\partial y}(0,0) = 0$. This gives us the special form of Equation C.9.

The equations C.3 and C.7 are equivalent.

Proof: It is easy to see that $f_{zz} = c$. Further, in this special form the sectional curvature $\kappa_s((0,0,1))$ is simply given by c, because the second fundamental form is given by the matrix of this quadric part itself:

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

For vectors of the tangent plane the sectional and radial curvature are the same: $c = \kappa_s((0,0,1)^t) = \kappa_r((0,0,1)^t) = (\prod((0,0,1)^t, (0,0,1)^t)$

This implies that Equation C.3 is nothing else but $k_r(v) = 0$.

One might be tempted to think that Equations C.4, C.8 are also directly equivalent, but this is not the case. Equation C.4 is view-dependent whereas the Gaussian curvature is not. Nevertheless, both depend on second order information. Intuitively one can guess that Equation C.3 contains the needed information and we will see that this is the case.

Let's first consider a *simpler* situation where the surface is actually given by C.9. This is not yet resulting in a generally valid proof because the derivatives in Equation C.4 do depend on the axis x and y. Therefore a rotation around the z-axis influences the value.

One realizes that the implicit equation corresponding to the map $\hat{m}$ is simply:

$$f(x,y,z) = \frac{1}{2}(ay^2 + 2byz + cz^2) + h.o.t - x$$

Now $f_x f_{yz} - f_y f_{xz} = -1 * b - f_y * 0 = -b$. Thus Equation C.4 implies $b = 0$. The Gaussian curvature on the other hand is given by the determinant of the second fundamental form, thus the

quadric matrix, leading to: $ac - b^2$ which equals ac because $b = 0$. We know $c = 0$ because the radial curvature is zero showing that the point is parabolic. The other implication is straight forward: the radial curvature c is zero thus Gaussian curvature equals to zero if $-b^2 = 0$ implying Equation C.4.

Actually at this point the proof could almost be considered complete. All that remains to be shown is that Equation C.4 equals zero iff for a rotation transformation around the z-axis Equation C.4 equals zero for the new axis $\hat{x},\hat{y},\hat{z} = z$. One might intuitively guess (because of the form of Equation C.4 which looks like a determinant) that a linear application modifying only the subspace x,y might just be linked by its determinant and this is what we are going to show next, but to ease understanding, one might still just think of a rotation around the z-axis:

$$f_x f_{yz} - f_y f_{xz} = det(R)(f_{\hat{x}} f_{\hat{y}z} - f_{\hat{y}} f_{\hat{x}z}),$$

where R transforms $\hat{x} = Rx = ax + by, \hat{y} = Ry = cx + dy$ and $hatz = Rz = z$. (realize the equivalence for an invertible R because $det(R^{-1}) = 1.0/det(R)$)

Proof: To show this we consider the following matrix: $T_2 :=$

$$\begin{pmatrix} f_x & f_y \\ f_{xz} & f_{yz} \end{pmatrix}$$

realize that $det(T_2) = 0$ is Equation C.4.

Now we create a matrix: $T_3 :=$

$$\begin{pmatrix} f_x & f_y & 0 \\ f_{xz} & f_{yz} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$=$

$$\begin{pmatrix} \bigtriangledown f \\ \bigtriangledown f_z = z^t Hess(f) \\ z^t \end{pmatrix}$$

Realize that $det(T_3) = det(T_2)$ and thus $det(T_3) = 0$ is still Equation C.4.

Before proving this invariance, one could realize that this defines a frame of the situation. The rotation of the surface around the z-axis corresponds to a rotation of the parameter space and vice versa. The silhouette is not changing in this case because the view direction is constant.

To prove this formally, let us suppose $\hat{x} = ax + by$ and $\hat{y} = cx + dy$ and $\hat{z} = z$.

$$\bigtriangledown f = \hat{\bigtriangledown} \hat{f} R^{-1}$$

thus:

$$\bigtriangledown f R = \hat{\bigtriangledown} \hat{f}$$

$$Hess(f) = \bigtriangledown^t \bigtriangledown f = R^{-t} \hat{\bigtriangledown}^t \hat{\bigtriangledown} f R^{-1}$$

thus:

$$\hat{Hess}(\hat{f}) = R^t \bigtriangledown^t \bigtriangledown f R$$

We have $z = Rz = \hat{z}$ and $z = R^t z$.

Therefore we get $\hat{T_3} :=$

$$\begin{pmatrix} \hat{\bigtriangledown}\hat{f} \\ \hat{z}^t \hat{Hess}(\hat{f}) \\ \hat{z}^t \end{pmatrix}$$

$=$

$$\begin{pmatrix} \bigtriangledown f R \\ z^t R^t Hess(f) R \\ z^t R \end{pmatrix}$$

$=$

$$\begin{pmatrix} \bigtriangledown f R \\ z^t Hess(f) R \\ z^t R \end{pmatrix}$$

Using $det(A(\vec{x}),A(\vec{y}),A(\vec{z})) = det(A)det(\vec{x},\vec{y},\vec{z})$, $det(AB) = det(A)det(B)$ we conclude:

$$det(\hat{T_3}) = det(R)det(T_3)$$

q.e.d.

2.2 Proof II

If we want to avoid the operator way of proving this result, it is also possible to derive it by using only the fact that the derivative of a function in a certain direction is given by the dot product of the gradient with this direction.

So let us see how directional derivatives are modified: $\bigtriangledown f = \hat{\bigtriangledown} f R$

$$f_{\hat{x}} = \bigtriangledown f(R(1,0,0)^t) = (R^t(\bigtriangledown f)^t)^t(1,0,0)^t$$

Equivalently we obtain almost the same equations for $(0,1,0)$ and $(0,0,1)$.

Thus the vector $\bigtriangledown(f)^t$ is transformed into $A^t \bigtriangledown (f)^t$

In the same way we can observe the second derivatives. Here we get:

$$\begin{aligned} & f_{\hat{x},\hat{z}} \\ = \; & (R(1,0,0)^t)^t Hess(f)(R(0,0,1)^t) \\ = \; & (1,0,0)R^t Hess(f)(0,0,1)^t \end{aligned}$$

And equivalently:

$$\begin{aligned} & f_{\hat{y},\hat{z}} \\ = \; & (0,1,0)R^t Hess(f)(0,0,1)^t \end{aligned}$$

Thus the vector $Hess(f)(0,0,1)^t$ is transformed into $R^t(Hess(f)(0,0,1)^t)$.

Finally we have $R^t(0,0,1)^t = (0,0,1)^t$

Now we know that for any linear application A:

$$det(R^t) = det(R)$$

...and although this is just an appendix... let's finish this dissertation with:

q.e.d.

"Real programmers can write assembly code in any language."

Larry Wall

Appendix D

Introduction française

Thèse:*Il n'y a pas de solution ultime pour ce qui est de la représentation de données. Il y a beaucoup de différents types de complexité (taille des données, complexité de l'algorithme ...) ainsi qu'énormément de tâches différentes (rendu, illumination, animation ...). Néanmoins, il est possible de classifier les solutions et on propose la catégorisation suivante : Préprocessus, Transformation à la volée, Représentation, Structuration et Reformulation.*

Fig. D.1 : Personnages de films

De gauche à droite: Toy Story (1996, Pixar/Disney) , Shrek (2001, Dreamworks) , Ice Age (2002, Blue Sky/20th Century Fox) , Finding Nemo (2003, Pixar/Disney) , Wall-E (2008, Pixar/Disney)

L'*informatique graphique* désigne la recherche liée à la création et à la manipulation de contenus visuels. Ceci concerne non seulement les images finales, mais aussi la géométrie utilisée pendant le processus d'affichage (aussi appelé rendu). Apparu pendant les années 60 avec les travaux de Sutherland [Sut63], ce jeune domaine a depuis grandi et a rapidement donné des résultats incroyables. Les images créées par ordinateur, ainsi que la pléthore d'outils utilisés pour la création et la manipulation de ce contenu digital, ont atteint un niveau de qualité impressionnant.

Même si la portée de l'informatique graphique est souvent considérée comme limitée, son impact est néanmoins énorme et influence plusieurs domaines :

- **Jeux** - avec 9,5 milliards de dollars (2007), l'industrie vidéo-ludique constitue l'un des marchés les plus grands dans l'industrie de divertissement.
- **Films** - les effets spéciaux deviennent de plus en plus importants et les images créées par l'ordinateur sont maintenant très répandues (la figure D.1 montre quelques exemples de production de films). Le contenu de Star Wars - Episode 3 est à 90% virtuel et la fusion avec les éléments véritablement filmés a été un défi important.
- **Architecture** - la prévisualisation devient de plus en plus importante. Des projets comme la création d'un nouveau stade peut coûter des centaines de millions de dollars. Une esti-

mation visuelle et réaliste (possiblement animée et interactive) assure que l'aperçu correspondra à la réalisation finale. Les choix d'ordre esthétique, mais aussi économique peuvent bénéficier de telles techniques.

- **Contrôle à distance** - la vue augmentée d'une simulation précise peut aider pour guider un robot à distance. Les défis restant à surmonter incluent l'interaction, la réponse haptique et la qualité des rendus, et ce souvent avec des données mesurées et digitalisées à la volée.
- **Visualisation de données médicales** - des volumes gigantesques de données existent et à l'heure actuelle une navigation continue, mais surtout l'interaction temps-réel sont impossibles. Il est en outre intéressant de constater l'impact sur d'autres domaines, comme le travail de Liu et al. [LTF*05] sur l'exagération de mouvements. Ceci a notamment aidé pour la découverte du mécanisme d'écoute dans notre oreille [GAF07].
- **Entraînement** - les simulations ont des avantages importants par rapport à une exécution réelle d'un point de vue économique et sécuritaire. Beaucoup de facteurs ont besoin d'être considérés pour obtenir une image et une interaction réalistes.
- **Biologie** - les animaux s'adaptent à leur environnement et leurs comportements changent en captivité. Récemment, l'utilisation de la réalité virtuelle a permis de convaincre des animaux captifs de leur liberté. Ainsi, il était possible d'examiner leur comportement naturel dans un environnement contrôlé. Le Max-Planck-Institute pour l'ornithologie a démontré avec cette technique que les oiseaux n'utilisent que la moitié de leur cerveau pendant des vols distants. Ceci leur permet de se reposer et dormir par intermittence pendant les trajets. Le résultat était possible en utilisant des projecteurs et une simulation d'environnement (les changements jour/nuit et les étoiles dont les oiseaux ont besoin pour s'orienter). Les machines à vent ont gardé les oiseaux dans la même position pendant l'expérience `http://www.orn.mpg.de/`. Aussi les athlètes utilisent l'informatique graphique pour visualiser et optimiser leurs mouvements lors de leur entraînement.
- **Outils** - la création de contenu est de plus en plus importante. La conception de niveaux de jeux, la création d'images et le développement de vidéos sur internet, ainsi que les communautés virtuelles en règle générale, augmentent le besoin d'outils adaptés. La création simple de contenu complexe est une tâche de première importance.

Fig. D.2 : *(StevenMillerPhotography.com)*

Le problème général et récurrent dans tous ces scénarii d'application est la complexité. Partout le réalisme, la précision et la facilité d'interaction sont bridés par des limitations d'ordre calculatoire et algorithmique. Par exemple, il suffit de considérer la différence évidente entre les scènes de la vie réelle, comme celles capturées par le photographe Miller (Figure D.2) et les scènes que l'on est capable de simuler approximativement dans un monde virtuel. La nature est belle et la beauté est complexe. Les échanges lumineux entre les feuilles d'un arbre, les ombres projetées sur le sol, la semi-transparence du feuillage, les caustiques et réfractions provoquées par des gout-

telettes d'eau sont des phénomènes observés très souvent dans la réalité. Néanmoins, les modèles physiques associés sont compliqués et leur transfert dans le monde digital difficile. La création d'images vraiment photo-réalistes est encore un problème ouvert pour des scènes compliquées, même avec des capacités de calculs très importantes. Probablement, ceci va rester un problème pendant longtemps, si jamais on trouve une solution véritablement finale.

Les tâches d'affichage et d'interaction deviennent encore plus compliquées quand les images ont besoin d'être créées en temps réel (environ 30-60 fois par seconde). Des scènes de centaines de milliers de polygones doivent être affichées sur un écran avec plusieurs millions de points (*pixel*) en quelques millisecondes. De plus, il y a aujourd'hui une tendance à privilégier des résolutions toujours plus élevées pour les projecteurs ainsi qu'à l'utilisation accrue du suréchantillonnage pour assurer une bonne qualité visuelle. Compliquant encore davantage cette situation, les véritables applications combinent souvent plusieurs calculs simultanément ; en conséquence, chaque tâche doit être résolue à une vitesse en vérité supérieure à 60 fps. Ceci est souvent oublié et on devrait être plus prudent quand on affirme qu'une technique est prête à être directement utilisée en pratique uniquement sur le fait qu'elle dépasse la limite des 60 fps. Par exemple, les jeux actuels n'utilisent que 5 ms pour les calculs d'ombres, ce qui correspond à 200 fps en réalité. Beaucoup d'algorithmes actuels sur ce sujet sont donc disqualifiés. Néanmoins ce sont des indicateurs pour l'avenir et ils pourraient devenir une solution acceptable dans un avenir proche.

Mais les avancées technologiques des ordinateurs sont toujours accompagnées par une volonté d'augmenter le réalisme et la qualité des images. La complexité ne reste pas donc sur un niveau fixé *a priori* : le nombre de personnages augmente dans les jeux (en particulier pour les jeux en ligne), le niveau de détails se raffine, les effets deviennent plus réalistes, les données en medicine/biologie/geologie représentent des terra-octets d'informations, l'interaction devrait être évaluée entre des scènes de plus en plus volumineuses, les images sont également de très haute résolution... Le Saint Graal ici serait donc une représentation qui traite tous ces problèmes liés à la complexité simultanément. Néanmoins selon nous, ceci est impossible. En conséquence, notre première thèse dans cette dissertation est qu'*une solution ultime pour traiter l'ensemble des problèmes de complexité n'existe pas.*.

1 Aux origines de la complexité

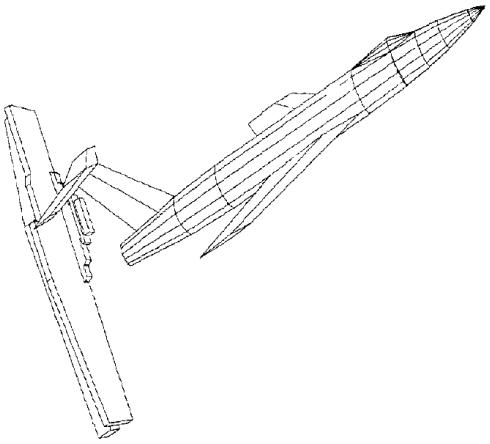


Fig. D.3 : *Une scène complexe en 1967*

Une définition exacte de la *complexité* sans entrer dans les détails est une tâche ardue. Par exemple, Seth Lloyd [Llo06] en donne trente-deux définitions différentes. La définition qui vient de l'informatique, est souvent liée à la complexité du *temps de calcul* ou de *coût de mémoire*. Typiquement, ceci est décrit avec la notation de Bachmann-Landau, qui définit le comportement limite de l'algorithme par rapport à ces arguments. C'est donc une mesure de comment la taille de l'entrée fait varier le temps de calcul et l'utilisation de mémoire. De plus, souvent l'entrée est choisie d'un certain type pour évaluer les bornes inférieure, supérieure ou moyenne, et il n'est pas évident de tirer des conclusions pour les entrées utilisées en pratique. En revanche, comme le résul-

tat est valable seulement pour la limite, le temps précis que l'algorithme prend pour terminer ses calculs peut différer grandement. En particulier, les constantes employées n'influencent pas la classe de complexité à laquelle appartient l'algorithme.

En informatique graphique, mais aussi ailleurs, les constantes et la performance effective sur de véritables ensembles de données utilisés en pratique sont cruciaux. Pour ajouter à la confusion, la définition d'un *ensemble de données pratiques* peut changer d'une génération d'ordinateurs à l'autre. En 1967 [App67], une scène composée de 195 polygones était jugée complexe et avait besoin de 84.6 secondes(!) pour être affichée à l'écran (cf Figure D.3). Aujourd'hui des scènes qualifiées de complexes excèdent le million de polygones. Des jeux actuels, tel que celui présenté en figure D.4, utilisent des centaines de milliers de polygones et impliquent également des calculs plus coûteux comme, par exemple, l'ombrage, les rendus de matériaux (textures), les ombres, et même une amélioration d'image *a posteriori*, le tout en assurant des fréquences comprises entre 30 et 60Hz. Pour des productions de films, un million polygones n'est pas beaucoup et des personnages composées de plusieurs millions de polygones sont maintenant communs. La notion devient donc ambiguë.

Considérations algorithmiques mises à part, il y a d'autres situations que l'on pourrait associer avec la notion de complexité. En général, une tâche peut être complexe car il y a beaucoup d'étapes à suivre (par exemple construire un bâtiment); où il y a beaucoup de cas particuliers à traiter, certains imprévisibles. Il peut y avoir beaucoup de choix successifs à faire qui mèneraient à une explosion combinatoire si l'on s'attelait à traiter chacun d'eux; ou bien, la sortie peut être simplement beaucoup plus large que l'entrée. Enfin la complexité peut aussi trouver son origine, soit du fait que la représentation courante ne permet pas soit d'inférer aisément un résultat, soit qu'une solution ne peut être trouvée qu'après plusieurs itérations sur le jeu entier des données.

Fig. D.4 : *Assassin's Creed (2008, UbiSoft)*

Il convient donc de définir la complexité d'une manière à prendre en compte la tâche considérée. Une tâche est dite complexe s'il n'y a pas une solution directe pour la résoudre, voire si aucune solution n'apparaît au terme d'un délai acceptable. Comme nous défendons ici l'assertion qu'il n'existe pas de solution ultime, on va donc fournir un ensemble de stratégies pour s'attaquer à ce problème. Autrement dit, même si la complexité est strictement liée à la tâche et au contexte, nous pouvons néanmoins en tracer des origines communes.

On distingue ici les sources suivantes :

- **Entrée/Stockage** - L'entrée ou la représentation des données peut avoir un impact significatif sur la complexité d'une tâche. Si des composants multiples et/ou inadaptés sont impliqués, un calcul peut vite devenir coûteux et compliqué. Par exemple, la description d'un modèle de ville gigantesque avec beaucoup de détails via une liste de polygones peut mener à un coût de stockage prohibitif. Ceci peut résulter en des problèmes de mémoire et de bande passante. *A contrario*, un modèle procédural est plus léger car les données sont créées à la volée pour le point de vue courant.

- **Manipulation** - Des tâches peuvent parfois s'avérer redondantes et théoriquement être transférées d'une entité de la scène à une autre. Ceci est impossible si la scène ne fournit pas une mesure de similarité de ses composants. Dans le cas d'une ville, si représentée par une soupe de polygones, des changements globaux tels que "subdivise chaque cylindre représentant une cheminée en plus de polygones" sont impossibles. La création et la manipulation de contenu est donc complexe en soit. Avec un système d'instanciation, on pourrait éditer un seul objet de référence et propager la modification à ses instances au sein de la scène. Un autre exemple pourrait être la construction d'une telle ville. Si la vue finale est seulement confinée à une rue, la majeure partie de la ville sera au final invisible. Il serait donc possible de se concentrer sur les parties non-cachées, l'accès à ce genre d'information serait donc ici très intéressant. Comme nous allons le voir, la détermination de la visibilité implique donc des calculs complexes et est souvent déterminée en amont via un préprocessus. Un tel calcul est néanmoins compromis si la scène évolue lors de sa visualisation, ce qui est évidemment le cas lors de l'édition de géométrie. Un tel problème est donc particulièrement difficile à résoudre.

- **Interaction** - Dans beaucoup de scénarii, nombreux sont les éléments qui entrent en interaction avec la scène elle-même (collisions, *light transfer* ...) et plus leur nombre augmente, plus les processus en jeu deviennent coûteux. Des personnages au comportement indépendant explorant une ville et guidés par leur champ de vision impliquent beaucoup de tests pour déterminer ce qu'ils sont supposés voir. Si on tient compte des réflexions dans les fenêtres, c'est encore plus dur. De même que pour un miroir, le champ de vision sera potentiellement élargi. Et l'interaction de chaque individu avec les autres génère également d'importants calculs. Un autre exemple est la pluie : des millions de goûtes ont besoin d'être testées contre les bâtiments de la scène et sont susceptibles de s'accumuler entre elles et créer bassins ou rigoles.

- **Evaluation** - Les données sont souvent traitées selon des tâches variées qui, dans le cadre de l'informatique graphique, réfèrent par exemple à l'affichage à l'écran, l'extraction de lignes significatives (dessins au trait) ou des déformations perspectives selon un modèle de caméra spécifique. Par exemple, le rendu d'un bâtiment implique une opération de projection, rasterisation et calcul d'ombrage pour chaque polygone. Tâche relativement complexe, mais parfois pas nécessaire. En fait, pour les bâtiments lointains, il serait beaucoup moins coûteux et visuellement équivalent d'afficher une fausse façade représentée par une simple image. En particulier, l'utilisation de *shaders* (voir chapitre 2) est un moyen d'enrichir l'apparence sans ajout de géométrie.

- **Sortie** Il y a beaucoup de situations où la sortie est le véritable problème du processus. Un examen exhaustif (comme la visibilité dans une ville) peut rapidement créer un coût important. Il faut donc se poser la question de la représentation d'une telle sortie, une sortie appropriée pouvant s'avérer extrêmement avantageuse. Par exemple, la vue de la façade pourrait être générée avec une image de résolution 1024×1024. Cette solution peut être coûteuse à transférer à travers un réseau et même à déchiffrer, par exemple, dans le contexte d'un guide de ville. À la place, un *clip-art* ou un rendu vectoriel peut s'avérer plus adapté de par sa complexité réduite susceptible de rendre sa compréhension plus aisée.

2 Contributions de la thèse

Mathématiquement, la complexité ne compromet pas l'existence d'une solution. Mais elle représente clairement un obstacle, les problèmes apparaissant au moment où on veut utiliser une telle solution *en pratique*. Pour trouver une solution acceptable, tout en demeurant suffisamment efficace, il faut attaquer le problème d'une manière adaptée. Il faut donc trouver des solutions spécifiques pour un type spécifique de complexité dans le contexte d'une tâche spécifique. La dissertation contient une variété de contributions dans des domaines multiples de l'informatique graphique.

Chaque idée est ici concrétisée par un programme qui représente une solution pratique améliorant les solutions précédemment connues. Typiquement, chaque programme offre une interface utilisateur qui permet d'explorer une variété de paramètres. Comme mentionné précédemment, il convient de nos jours à privilégier des solutions dont le contrôle, le plus intuitif possible, peut être compris facilement et permet de focaliser la puissance de calcul envers des tâches hautement répétitives. Ceci se reflette au sein de cette dissertation sous deux formes. Premièrement, tous les algorithmes ne s'appuient que sur un ensemble réduit de paramètres et deuxièmement, des solutions nécessitant l'intervention de l'utilisateur sont privilégiées à des techniques entièrement automatisées. Ceci convient pour des tâches faciles pour un être humain, mais encore difficiles du point de vue de l'ordinateur tels le caractère sémantique du *grouping* d'éléments d'une scène [BEDT08] ou le sens artistique nécessaire à la stylisation de ses éléments [EWHS08].

La puissance de calculs peut de même être utilisée pour aider l'utilisateur à résoudre des problèmes répétitifs ou fastidieux. Le style ou la sémantique des données sont décrites à un haut niveau et l'ordinateur l'applique dans des cas précis. C'était aussi une de nos motivations dans le chapitre 12, où on montre qu'une réponse instantanée de visibilité peut aider pendant la création d'une scène. Avec cette information, un artiste peut directement intégrer des considérations de performance dans le processus de création. Notre travail sur la simplification d'objets 5 partage cet aspect.

Même si le but de diminuer la complexité peut sembler très vague et vaste, on a trouvé que toutes les approches qui partagent ce but semblent suivre cinq principes présentés dans la suite. Ceci permet de structurer les approches et cette catégorisation est utile pour le développement de nouvelles méthodes. On a suivi ces principes dans nos travaux et nous avons contribué dans chacune des classes que nous avons définies. Il est clair néanmoins que ces classes sont pas parfaitement séparées et souvent la combinaison de plusieurs de leurs aspects reste une possibilité. Ce document fournit plusieurs états de l'art qui soulignent que notre nomenclature est suffisamment riche pour classer de nombreux algorithmes et minimale dans le sens où chaque catégorie contient des représentants.

Préprocessus:

Une manière de gérer la contrainte de temps est d'effectuer un maximum de calculs en avance. Dans ce contexte, les temps de calcul sont moins significatifs car le temps est perdu avant les évaluations critiques. Il est donc une bonne idée de viser des algorithmes plutôt précis car on a plus de temps à leur accorder.

Par exemple, trouver l'aire d'une surface d'un modèle polygonal peut prendre quelque temps (parcours des faces, produit vectoriel, somme...). Il est donc plus efficace de stocker le résultat

une fois pour chaque face. De plus, si l'on connaît l'utilisation de cette valeur et peut donc déduire la précision nécessaire à l'application, on peut intégrer ce savoir dans le préprocessus pour diminuer, par exemple, la mémoire nécessaire pour enregistrer les valeurs en jeu. Dans ce document, on va voir des contributions beaucoup plus sophistiquées que l'exemple précédent.

Des scènes extrêmement complexes ne peuvent être affichées en temps-réel. Pour y arriver, il est nécessaire de simplifier leurs éléments. Même si la variété d'algorithmes partageant cet objectif est énorme (un état de l'art est donné dans chapitre 4), il semble que peu de travaux assurent une simplification précise, qui permettrait de borner l'erreur commise pendant le processus. La question que l'on s'est posée dans notre travail est : étant donné un modèle initial, de combien peut on le déformer en une forme simplifiée telle que son apparence reste proche de l'original quand l'objet est observé depuis une région prédéfinie ? Même si ***les bornes exactes d'erreur*** sont d'un intérêt avant tout théorique, leur connaissance donne une vision approfondie du processus général de simplification et mène à une nouvelle interprétation d'erreurs géométriques. De plus, le résultat peut être utilisé dans le contexte d'un système de validation pour trouver la déviation majeure de l'objet simplifié par rapport à l'objet initial, ainsi que la position de vue qui maximise cette erreur. On se concentre sur la situation 2D, mais nous montrons aussi comment étendre le résultat à une validité de points en 3D. Ainsi a-t-on développé un nouvel ***algorithme de simplification extrême*** qui devrait être considéré comme l'illustration applicative de nos résultats théoriques. Dans ce contexte, nous considérons l'apparence globale de la scène et traitons tous les objets comme une seule entité. On évite l'utilisation d'information d'adjacence et on produit une représentation qui agglomère tous les éléments susceptibles d'être fusionnés. Notre approche semble être la première à discuter de l'erreur de simplification dans un cadre aussi général et est présentée dans le chapitre 5.

Notre travail a été publié dans Computer Graphics Forum:
E. Eisemann and X. Décoret:
On Exact Error-bounds for View-dependent Simplification [ED07a].

Représentation:

La manière dont sont représentées les données est cruciale. Une représentation peut en effet être plus naturellement adaptée pour certaines formes de données que d'autres. Une sphère peut ainsi être décrite facilement par une équation aussi simple que $||x|| = 1$, et qui néanmoins capture parfaitement sa forme, *a contrario* d'une construction géométrique via des triangles qui entraînera nécessairement une erreur. Une application classique de cette formulation implicite par une équation est la détection approximative de collisions, rendue bien plus aisée qu'en utilisant sa version polygonalisée. De plus, si l'on veut calculer des propriétés différentielles (courbure, espace tangentiel ...), il est plus difficile de le faire sur un maillage polygonal (géométrie différentielle discrète), tandis que la formulation implicite permet une définition facile et cohérente de ces valeurs.

Dans notre travail, nous avons développé des algorithmes qui bénéficient particulièrement d'une interprétation appropriée de leurs données. Ainsi plusieurs de nos contributions produisent des sorties qui dépassent le cadre d'ordinaires images, comme par exemple l'ensemble des courbes d'intérêt extraites d'un modèle. Le résultat est alors une illustration simple de l'entrée initiale,

à base de régions polygonales superposées. Ces polygones capturent la forme globale de l'objet initial, alors que les travaux antérieurs avaient besoin de garder une collection de triangles projetés. Cette nouvelle représentation est la clé pour une édition simplifiée et une représentation compacte.

Pour des modèles polygonaux, même s'il est important d'assurer l'extraction de régions fermées, on peut s'appuyer sur des travaux précédents. Pour les surfaces implicites, il convenait d'introduire un nouveau ***schéma d'advection*** permettant de diriger un système de particules vers les courbes d'intérêt le long de la surface. Le principe est basé sur les multiplicateurs de Lagrange et permet plus généralement la détermination de l'intersection entre deux variétés.

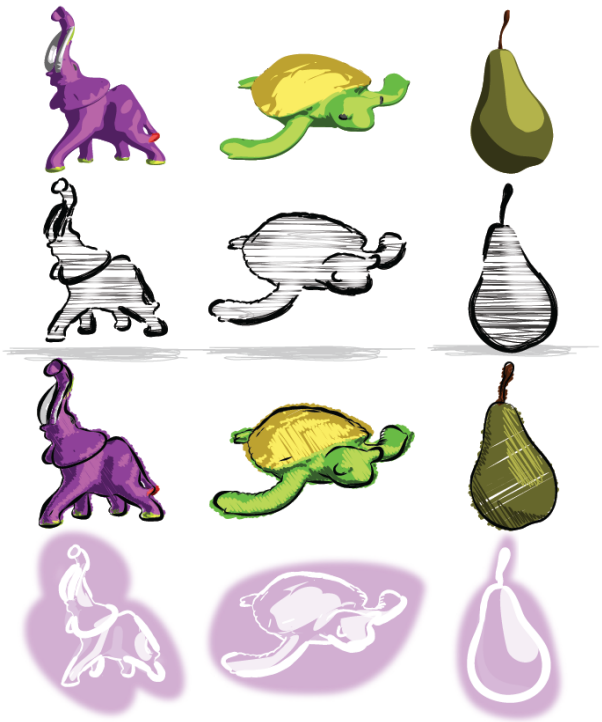


Fig. D.5 : *Rendus vectoriels stylisés*

La sortie vectorielle brise la barrière souvent imposée par la grille de pixels. Le résultat est adapté pour une variété d'écrans et les déformations éventuelles n'introduisent pas d'artéfacts qui normalement apparaissent pour des versions pixélisées. Egalement on a présenté un système générique pour ***styliser*** et enrichir l'apparence de telles représentations vectorielles. Cette question intervient au moment d'illustrer un document. Pose, lumière et style sont normalement combinés de manière conjointe et interdépendante par un artiste-expert. Il est donc difficile de trouver une image qui corresponde immédiatement à toutes les attentes. La possibilité de créer facilement des images vectorielles dans un style cohérent peut se résumer comme une nouvelle manière de navigation dans l'espace gigantesque des illustrations et correspond aux réels besoins des utilisateurs.

A présent, notre système vise la création d'images statiques seulement. Néanmoins, la théorie sous-jacente au suivi des caractéristiques topologiques des lignes de silhouette a été explorée et permettrait d'assurer la cohérence temporelle de rendus vectoriels animés. Seul ce résultat a été mentionné dans [SEH08], la preuve formelle manquant à l'appel. On explique ici en détail comment les équations de Plantinga et Vegter [PV06] peuvent accélérer les méthodes existantes et on donne une preuve d'équivalence dans l'annexe C.

La création d'une sortie vectorielle pour les surfaces implicites est apparue dans TVCG:
M. Stroila, E. Eisemann and J. C. Hart:
Clip Art Rendering of Smooth Isosurfaces [SEH08].

Leur construction à partir d'un maillage et le système de stylisation ont été présentés à EGSR:
E. Eisemann and H. Winnemöller and J. C. Hart and D. Salesin:
Stylized Vector Art from 3D Models with Region Support [EWHS08].

Transformation:

Nous avons mentionné que pour beaucoup de situations, une tâche peut être plus simple avec une représentation adaptée. Pour une scène dynamique, il peut donc être intéressant de transformer les données à la volée sous une forme plus intéressante quant à la tâche à remplir, forme probablement très approximative et *a priori* valide uniquement à un instant précis de la simulation. Cette opération a besoin d'être rapide car sinon le gain de l'accélération obtenue par le changement de représentation risque d'être perdu.

Par exemple, pour le calcul d'ombres, on a besoin de déterminer pour chaque point d'une scène (ou sur l'écran) si son pendant 3D est dans l'ombre ou non. Il est donc important d'avoir une manière de répondre à cette requête rapidement. Malheureusement, on ne peut pas précalculer et enregistrer cette information à une précision suffisante (et l'on ne peut pas considérer toutes les possibles positions de lumière en amont). Il est donc intéressant d'avoir une manière rapide de transformer la scène dans une structure de données qui permet des requêtes rapides. Dans cet exemple, ceci pourrait être réalisé via un tampon d'ombre (*shadow map*) [Wil78], qui encode la première surface vue à partir de la lumière (donc celles qui sont illuminées). Ceci peut être obtenu rapidement avec un simple rendu de la scène à partir de la source. Cependant comme précisé plus haut, cette représentation ne serait valide que pour cet instant dans le temps. Au delà, à la fois la lumière et/ou les objets peuvent avoir changé d'emplacement. Un autre exemple est de choisir le niveau de détails géométriques selon le point de vue courant : beaucoup de détails sont perdus si l'objet se projette sur une trop petite zone de l'écran.

Dans ce document, on se pose la question de comment peut-on transformer les données triangulées en une représentation qui, sur le GPU (Graphics Processing Unit), permettrait une réponse rapide pour des requêtes relatives à la présence de matière dans la scène ou de ses attributs comme les normales. La partie III traite de la voxélisation d'une scène effectuée sur le GPU qui est un exemple d'extraction d'informations à la volée. Plusieurs tâches bénéficient d'une telle information globale accessible rapidement et aléatoirement, des requêtes de visibilité n'ayant par exemple besoin que de l'information de présence de matière. Ce constat a en effet été la motivation principale de notre recherche dans ce domaine. Dans le chapitre 7, on présente un algorithme efficace pour ***voxéliser*** une scène. Il est exécuté entièrement sur le GPU et peut éviter tout transfert ou interaction avec le processeur principal de la machine. En conséquence, la représentation peut être mise à jour à chaque image, ce qui permet d'étendre notre nouvelle représentation à des scènes dynamiques. De plus, on y avons ajouté une modification qui permet l'obtention d'un ***intérieur solide*** qui constitue l'ensemble des parties de la scène renfermant de la matière. La solution est étonnamment simple et le plus impressionnant est que les performances obtenues dépassent celles d'autres algorithmes récents (par exemple [CLT07]) de plusieurs ordres de grandeurs. Une extension intéressante permet de dériver la densité locale de cet intérieur ainsi que ses normales par la suite. On montre une large variété d'applications dans le chapitre 8 qui bénéficient de cette extraction haute résolution ($> 1024^3$) pour des maillages de plus de 300.000 polygones à des fréquences supérieures à 90Hz. Enfin toutes les applications mentionnées surpassent les solutions obtenues jusqu'alors de plusieurs aspects.

La voxélisation sur GPU a été publiée à I3D:
E. Eisemann and X. Décoret:
Fast scene voxelization and Applications [ED06a].

Le papier a été repris comme *sketch* à SIGGRAPH:
E. Eisemann and X. Décoret:
Fast scene voxelization and Applications [ED06b].

La voxélisation solide a été publiée à GI:
E. Eisemann and X. Décoret:
Single-Pass GPU Solid Voxelization and Applications [ED08b].

Structuration:

Une manière classique pour traiter un problème complexe est d'en proposer une hiérarchisation. Pour une scène tridimensionnelle, cette tâche est relativement aisée pour peu que la structure soit donnée, elle est sinon bien moins évidente. Ceci peut dépendre de considérations techniques comme pour la création de groupes de taille uniforme. Par exemple, la technique de *lancer de rayons* utilise typiquement des hiérarchies de volumes englobants. Au delà, les groupes pourraient dépendre d'une définition sémantique comme la recherche d'instances d'un même modèle ou du regroupement des éléments partageant un même matériau. Ce dernier point est très intéressant soit pour un rendu efficace, soit dans le cadre de manipulations de la scène. D'ailleurs, nombreuses sont les situations pouvant tirer profit d'une structuration multi-échelle.

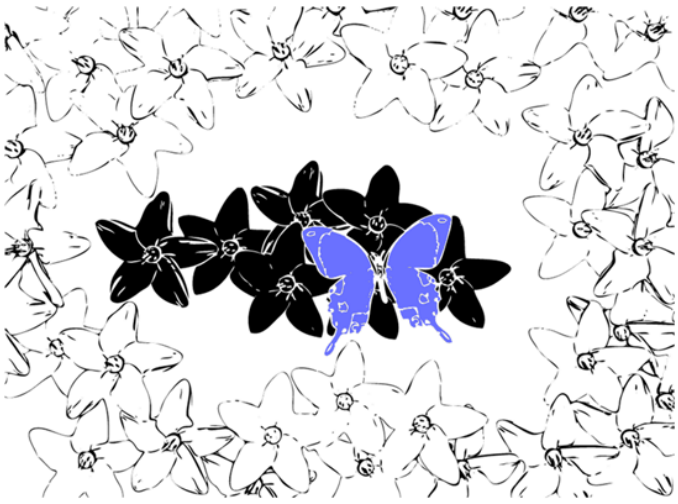


Fig. D.6 : L'attention est attirée ici par le papillon - *son importance est partagée par son groupe.*

Dans cette dissertation, au lieu de se limiter à une méthode de structuration dont le seul but serait l'accélération, on propose un système qui guide un processus de stylisation de scènes qui en exploite l'information structurelle. On explore également comment l'abstraction de dessins peut bénéficier de cet ajout. La ***stylisation de groupes*** est un moyen en art pour communiquer des connections et relations entre les objets et ainsi guider l'interprétation d'une image. On introduit une nouvelle manière d'établir des groupes dans des scènes dynamiques et on exploite cette information pour créer des rendus non photo-réalistes et en contrôler l'animation.

La technique donne à chaque entité une information globale concernant les éléments groupés ainsi que leurs attributs. Cette information est extraite d'une manière novatrice et efficace. Les critères pour ce processus peuvent être adaptés par l'utilisateur et permettent de décrire des comportements complexes avec des interactions simples. Une possibilité de stylisation est d'appliquer une abstraction similaire à tous les membres d'un même groupe, ce qui peut donner une apparence plus compréhensive et unifiée.

Les résultats sur la stylisation ont été publiés à NPAR:
H. Bezerra, E. Eisemann, X. Décoret and J. Thollot:
3D Dynamic Grouping For Guided Stylization [BEDT08].

Reformulation:

Par ce mot, nous entendons la volonté de reconsidérer le problème initial à résoudre et ainsi aboutir à une solution équivalente mais via des calculs plus simples ou une modification d'objectif. Autrement dit, au lieu de modifier le modèle, on modifie la tâche.

Dans le cadre du transport de lumière dans une scène par exemple, le calcul est utile tant que l'intensité reste visible à l'écran. De même si le transfert est trop coûteux, se restreindre à un certain type de matériel seulement (par exemple diffus) peut énormément simplifier la tâche. Si l'on veut calculer la lumière directe uniquement, les faces qui ne sont pas orientées vers la lumière peuvent être exclues du calcul car ne pouvant pas recevoir d'énergie. La question centrale dans ce contexte est alors combien de calculs peuvent être évités tout en continuant à obtenir une solution satisfaisante.

Dans la partie IV, on applique ce principe à plusieurs algorithmes de visibilité. En particulier, on a développé des approches pour la production d'ombres douces, problème restant encore majeur pour son extension aux rendus temps-réel. Pour le démontrer et nous comparer aux travaux précédents, nous présentons dans le chapitre 10 un état de l'art avant de présenter nos contributions. Celles-ci varient entre des solutions précises mais relativement lentes, à des résultats approximatifs mais extrêmement rapides.

Dans nos travaux sur les ***ombres douces plausibles*** dans le chapitre 11, on obtient une solution approximative et très efficace. À ce moment de l'écriture, la technique ici présentée est (à notre connaissance) l'algorithme d'ombres douces le plus rapide pour créer les zones de pénombre d'une source locale et les performances obtenues sont indépendantes de la taille de la lumière. Ceci est une propriété à la fois remarquable et rare, d'autres méthodes étant très souvent fortement pénalisées, quand la taille de la lumière augmente. D'autres ne peuvent simplement pas considérer des sources locales ou seulement des sources tellement gigantesques que les détails des ombres tendent à disparaître. Nous obtenons ce résultat par une réinterprétation du calcul de visibilité comme un processus de filtrage, basé sur le travail de Soler et Sillion [SS98].

La deuxième contribution de cette partie concerne l'***échantillonnage de visibilité*** et est présentée dans le chapitre 12. Ici, la visibilité est échantillonnée précisément dans des scènes triangulées et ce calcul est effectué efficacement sur la carte graphique. La méthode trouve ses applications dans le contexte du calcul précis des textures de lumière (*lightmaps*) et dans la modélisation assistée par la visibilité. La situation initialement simple ne considérant qu'une source et un récepteur uniques permet l'optimisation des requêtes. Globalement, on accumule consécutivement la contribution totale d'occlusion de chaque triangle au moment celui-ci est traité par le GPU. On reste dès lors compatible avec le processus de *streaming* qui est à la base des machines parallèles. Notre application à la modélisation assistée par visibilité est particulièrement intéressante car l'on s'attaque alors aux racines du problème de la complexité, au moment même de la création des modèles. A l'image de notre travail sur la simplification, l'information fournie peut aider

un utilisateur à faire les bons choix en termes d'efficacité au moment de la modélisation et ainsi appréhender les considérations de performance en amont du processus de production.

La dernière contribution du chapitre 13 étend l'algorithme précédent à des ***ombres douces exactes sans aliasing***. La distinction entre source, occulteur et récepteur est alors supprimée, l'entrée étant simplement constituée d'une source et de la scène triangulée. L'approche crée une image équivalente au lancer de rayons pour des sources volumétriques, mais à des coûts bien inférieurs en exploitant les dernières extensions des cartes graphiques accessibles depuis DX10. Il n'y a plus aucun besoin de structures hiérarchiques, le résultat restant exact à l'échelle des pixels pour le point de vue donné, et ce en présence d'ombres dures. Une difficulté de l'algorithme est la détermination de la ***région de pénombre***, soit la région qui nécessite la détermination de la visibilité de la source lumineuse. Dans la pratique, on a trouvé qu'une surestimation est suffisante, mais nous avons développé une dérivation analytique de sa forme pour les sources lumineuses ellipsoïdales. On la présente en annexe A. Pour conclure, on s'interroge en annexe B sur comment améliorer la qualité des ombres calculées par une approche en espace image.

Le travail sur les ombres douces plausibles a été publié tout d'abord à SIBGRAPI:
E. Eisemann and X. Décoret:
Plausible Image-based Soft Shadows using Occlusion Textures [ED06c]

Une version étendue est ensuite apparue dans Computer Graphics Forum:
E. Eisemann and X. Décoret:
Occlusion Textures for Plausible Soft Shadows [ED08a]

Notre approche pour l'échantillonnage de visibilité a été publiée à Eurographics:
E. Eisemann and X. Décoret:
Visibility Sampling on GPU and Applications [ED07b]

Finalement, le travail sur les ombres douces dépourvues d'aliasing a été publié à EGSR:
E. Sintorn, E. Eisemann and U. Assarsson:
Sample Based Visibility for Soft Shadows using Alias-free Shadow Maps [ED07b]

Bibliography

[AAM02] Assarson U., Akenine-Möller T.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2002), Springer Computer Science, Eurographics, Eurographics Association, pp. 297–306.

[AAM03] Assarsson U., Akenine-Möller T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 22*, 3 (2003), 511–520.

[AAM04a] Aila T., Akenine-Möller T.: A hierarchical shadow volume algorithm. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2004), Eurographics Association, pp. 15–23.

[AAM04b] Assarsson U., Akenine-Möller T.: Occlusion culling and z-fail for soft shadow volume algorithms. *The Visual Computer 20*, 8-9 (2004).

[AAM05] Aila T., Akenine-Möller T.: Conservative and tiled rasterization. *Journal of Graphics Tools 10*, 3 (2005).

[ABB*07] Andujar C., Boo J., Brunet P., Fairen M., Navazo I., Vazquez P., Vinacua A.: Omni-directional relief impostors. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3 (2007), 553–560.

[ACSD*03] Alliez P., Cohen-Steiner D., Devillers O., Levy B., Desbrun M.: Anisotropic polygonal remeshing. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2003), pp. 485–493.

[ADM*08] Annen T., Dong Z., Mertens T., Bekaert P., Seidel H.-P., Kautz J.: Real-time all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 27*, 3 (2008), to appear.

[ADMAM03] Assarson U., Dougherty M., Mounier M., Akenine-Möller T.: An optimized soft shadow volume algorithm with real-time performance. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2003), ACM Press.

[AHL*06] Atty L., Holzschuch N., Lapierre M., Hasenfratz J.-M., Hansen C., Sillion F.: Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum 25*, 4 (2006).

[AHT04] Arvo J., Hirvikorpi M., Tyystjärvi J.: Approximate soft shadows with an image-space flood-fill algorithm. *Computer Graphics Forum (Proceedings of Eurographics) 23*, 3 (2004), 271–280.

[AL99] Aliaga D. G., Lastra A.: Automatic image placement to provide a guaranteed frame rate. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1999), Addison Wesley Longman, pp. 307–316.

[AL04] Aila T., Laine S.: Alias-free shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Springer Computer Science, Eurographics, Eurographics Association, pp. 161–166.

[AM04] Aila T., Miettinen V.: dpvs: An occlusion culling system for massive dynamic environments. *IEEE Computer Graphics and Applications 24*, 2 (2004), 86–97.

[AM05] Aveneau L., Mora F.: Fast and exact direct illumination. In *Proceedings of CGI (Computer Graphics International)* (2005), IEEE.

[AMB*07] Annen T., Mertens T., Bekaert P., Seidel H.-P., Kautz J.: Convolution shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), vol. 18 of *Eurographics / ACM SIGGRAPH Symposium Proceedings*, Eurographics, pp. 51–60.

[AMH02] Akenine-Möller T., Haines E.: *Real-Time Rendering (2nd Edition)*. AK Peters, Ltd., 2002.

[AMS*08] Annen T., Mertens T., Seidel H.-P., Flerackers E., Kautz J.: Exponential shadow maps. In *Proceedings of GI (Graphics Interface)* (2008).

[And] Anderson S. E.: Bit twiddling hacks. graphics.stanford.edu/~seander/bithacks.html.

[App67] Appel A.: The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 22nd National Conference* (1967), pp. 387–393.

[ARHM00] Agrawala M., Ramamoorthi R., Heirich A., Moll L.: Efficient image-based methods for rendering soft shadows. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), Annual Conference Series, ACM SIGGRAPH, pp. 375–384.

[Arv07] Arvo J.: Alias-free shadow maps using graphics hardware. *Journal of Graphics Tools 12*, 1 (2007), 47–59.

[ASK06] Aszodi B., Szirmay-Kalos L.: Real-time soft shadows with shadow accumulation. In *Short Paper Eurographics* (2006).

[ASVNB00] Andújar C., Saona-Vázquez C., Navazo I., Brunet P.: Integrating occlusion culling with levels of detail through hardly-visible sets. *Computer Graphics Forum (Proceedings of Eurographics) 19*, 3 (2000), 499–506.

[BAS02] Brabec S., Annen T., Seidel H.-P.: Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering (Proceedings Computer Graphics International 2002)* (2002), Springer, pp. 397–408.

[BAS05] Brabec S., Annen T., Seidel H.-P.: Practical shadow mapping. In *Graphics Tools: The jgt Editors' Choice*. A.K. Peters, 2005, p. 343.

[BCS08] Bavoil L., Callahan S. P., Silva C. T.: Robust soft shadow mapping with backprojection and depth peeling. *Journal of Graphics Tools 13*, 1 (2008).

[BD06a] Baboud L., Décoret X.: Realistic water volumes in real-time. In *Eurographics Workshop on Natural Phenomena* (2006), Eurographics.

[BD06b] Baboud L., Décoret X.: Rendering geometry with relief textures. In *Proceedings of GI (Graphics Interface)* (2006), Canadian Information Processing Society, pp. 195–201.

[BEDT08] Bezerra H., Eisemann E., Décoret X., Thollot J.: 3d dynamic grouping for guided stylization. In *Proceedings of NPAR (Symposium on Non-Photorealistic Animation and Rendering)* (2008).

[Ber86] Bergeron P.: A general version of crow's shadow volumes. *Computer Graphics and Applications 6*, 9 (1986), 17–28.

[Bli] Blinn J.: Planar shadows. http://www.shadowstechniques.com/blinn.html .

[Bli88] Blinn J. F.: Me and my (fake) shadow. *IEEE Computer Graphics and Applications 8*, 1 (1988), 82–86.

[BMC05] Bertails F., Ménier C., Cani M.-P.: A practical self-shadowing algorithm for interactive hair animation. In *Proceedings of GI (Graphics Interface)* (2005).

[BP96] Baldazzi C., Paoluzzi A.: From polyline to polygon via xor tree, 1996. Tech. Rep. INF-04-96, Dip. Disc. Scient., Università Roma Tre, Rome, Italy, May 1996.

[BP04] Buck I., Purcell T.: *GPU Gems*. Addison-Wesley, 2004, ch. 37: A toolkit for Computations on GPUs.

[BPD06] Bae S., Paris S., Durand F.: Two-scale tone management for photographic look. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 25*, 3 (2006), 637–645.

[Bre65] Bresenham J. E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965).

[BS01] Brabec S., Seidel H.-P.: Hardware-accelerated rendering of antialiased shadows with shadow maps. In *Proceedings of CGI (Computer Graphics International)* (2001), IEEE, pp. 209–214.

[BS02] Brabec S., Seidel H.-P.: Single sample soft shadows using depth maps. In *Proceedings of GI (Graphics Interface)* (2002).

[BS03] Brabec S., Seidel H.-P.: Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Proceedings of Eurographics) 25*, 3 (2003).

[BS05] Boubekeur T., Schlick C.: Generic mesh refinement on gpu. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2005), ACM, pp. 99–104.

[BS06] Bavoil L., Silva C. T.: Real-time soft shadows with cone culling. In *Technical Sketch at SIGGRAPH* (2006).

[Bun06] Bunnell M.: *GPU Gems 2*. Addison-Wesley, 2006, ch. Dynamic Ambient Occlusion and Indirect Lighting.

[BWPP04] Bittner J., Wimmer M., Piringer H., Purgathofer W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum (Proceedings of Eurographics) 23*, 3 (2004), 615–624.

[BYM05] Bell N., Yu Y., Mucha P. J.: Particle-based simulation of granular materials. In *Proceedings of SCA (ACM SIGGRAPH/Eurographics Symposium on Computer Animation)* (2005), ACM Press, pp. 77–86.

[Car] Carmack J.: Z-fail shadow volumes. Internet Forum.

[Cav05] Cavanagh P.: The artist as neuroscientist. *Nature 434*, 434 (2005).

[CCMS97] Ciampalini A., Cignoni P., Montani C., Scopigno R.: Multiresolution decimation based on global error. *The Visual Computer 13*, 5 (1997), 228–246.

[CD03] Chan E., Durand F.: Rendering fake soft shadows with smoothies. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2003), Springer Computer Science, Eurographics, Eurographics Association.

[CD04] Chan E., Durand F.: An efficient hybrid shadow rendering algorithm. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Eurographics Association, pp. 185–195.

[CF90] Chin N., Feiner S.: Near real-time shadow generation using BSP trees. *Computer Graphics (Proceedings of SIGGRAPH) 24*, 4 (1990), 99–106.

[CF92] Chin N., Feiner S.: Fast object-precision shadow generation for area light sources using bsp trees. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (1992), ACM, pp. 21–30.

[CF99] Chen H., Fang S.: Fast voxelization of 3d synthetic objects. *Journal of Graphics Tools 3*, 4 (1999), 33–45.

[CG00] Cipolla R., Giblin P.: *Visual motion of curves and surfaces*. Cambridge Univ. Press, 2000.

[CG04] Chong H., Gortler S. J.: A lixel for every pixel. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Springer Computer Science, Eurographics, Eurographics Association.

[Cho03] Chong H.: *Real-Time Perspective Optimal Shadow Maps*. Master's thesis, Harvard University, 2003.

[CHPR07] Cook R. L., Halstead J., Planck M., Ryu D.: Stochastic simplification of aggregate detail. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 26*, 3 (2007), 79.

[CJW*06] Cai X.-H., Jia Y.-T., Wang X., Hu S.-M., Martin2 R. R.: Rendering soft shadows using multilayered shadow fins. *Computer Graphics Forum 25*, 1 (2006).

[CK01] Chhugani J., Kumar S.: View-dependent adaptive tessellation of spline surfaces. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (2001), pp. 59 – 62.

[Cla76] Clark J.: Hierarchical geometric models for visible surface algorithms. In *Communications of the ACM 19(10)* (1976).

[CLT07] Crane K., Llamas I., Tariq S.: *GPU Gems 3*. Addison-Wesley, 2007, ch. 30: Real-Time Simulation and Rendering of 3D Fluids.

[CMO97] Cohen J., Manocha D., Olano M.: Simplifying polygonal models using successive mappings. In *Proceedings of VIS (IEEE Conference on Visualization)* (1997), IEEE Computer Society Press, pp. 395–ff.

[COCSD02] Cohen-Or D., Chrysanthou Y., Silva C., Durand F.: A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics* (2002).

[COM98] Cohen J., Olano M., Manocha D.: Appearance-preserving simplification. In *Proc of SIGGRAPH* (1998), ACM Press.

[CRL01] Cornish D., Rowan A., Luebke D.: View-dependent particles for interactive non-photorealistic rendering. In *Proceedings of GI (Graphics Interface)* (2001), pp. 151–158.

[Cro77] Crow F. C.: Shadow algorithms for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH) 11*, 3 (1977), 242–248.

[Cro84] Crow F. C.: Summed-area tables for texture mapping. In *Computer Graphics (Proceedings of SIGGRAPH)* (1984).

[CRS98] Cignoni P., Rocchini C., Scopigno R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[CS95] Chrysanthou Y., Slater M.: Shadow volume BSP trees for computation of shadows in dynamic scenes. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (1995), pp. 45–50.

[CS08] Coming D., Staadt O.: Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (2008).

[CSAD04] Cohen-Steiner D., Alliez P., Desbrun M.: Variational shape approximation. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2004), ACM, pp. 905–914.

[CTCS00] Chai J.-X., Tong X., Chan S.-C., Shum H.-Y.: Plenoptic sampling. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 307–318.

[CVM*96] Cohen J., Varshney A., Manocha D., Turk G., Weber H., Agarwal P., Brooks F., Wright W.: Simplification envelopes. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1996), ACM Press.

[D02] Décoret X.: *Pré-traitement de grosses bases de données pour la visualisation interactive*. PhD thesis, Université Joseph Fourrier, 2002.

[dB06] de Boer W. H.: Smooth penumbra transitions with shadow maps. *Journal of Graphics Tools 11*, 2 (2006), 59–71.

[DBD*07] Drettakis G., Bonneel N., Dachsbacher C., Lefebvre S., Schwarz M., Viaud-Delmon I.: A perceptual rendering pipeline using contrast and spatial masking. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), Springer Computer Science, Eurographics, Eurographics Association.

[DCB*04] Dong Z., Chen W., Bao H., Zhang H., Peng Q.: Real-time voxelization for complex polygonal models. In *Proceedings of the Pacific Conference on Computer Graphics and Applications* (2004).

[DCFR07] DeCoro C., Cole F., Finkelstein A., Rusinkiewicz S.: Stylized shadows. In *Proceedings of NPAR (Symposium on Non-Photorealistic Animation and Rendering)* (2007), ACM, pp. 77–83.

[DD02] Durand F., Dorsey J.: Fast bilateral filtering for the display of high-dynamic-range images. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2002), ACM, pp. 257–266.

[DDP96] Durand F., Drettakis G., Puech C.: The 3d visibility complex, a new approach to the problems of accurate visibility. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (1996), Springer Verlag, pp. 245–257.

[DDS03] Décoret X., Debunne G., Sillion F.: Erosion based visibility preprocessing. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2003), Eurographics.

[DDSD03] Décoret X., Durand F., Sillion F., Dorsey J.: Billboard clouds for extreme model simplification. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2003), ACM Press.

[DDTP00] Durand F., Drettakis G., Thollot J., Puech C.: Conservative visibility preprocessing using extended projections. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), pp. 239–248.

[Déc05] Décoret X.: N-buffers for efficient depth map query. *Computer Graphics Forum (Proceedings of Eurographics) 24*, 3 (2005).

[DFG99] Du Q., Faber V., Gunzburger M.: Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev. 41*, 4 (1999), 637–676.

[DH72] Duda R. O., Hart P. E.: Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM 15*, 1 (1972), 11–15.

[DHL*98] Deussen O., Hanrahan P., Lintermann B., Měch R., Pharr M., Prusinkiewicz P.: Realistic modeling and rendering of plant ecosystems. *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques 32*, Annual Conference Series (1998), 275–286.

[DHS*05] Durand F., Holzschuch N., Soler C., Chan E., Sillion F.: A frequency analysis of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 24*, 3 (2005).

[DL06] Donnelly W., Lauritzen A.: Variance shadow maps. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2006).

[Dro07] Drone S.: Advanced real-time rendering in 3d graphics and games. ACM SIGGRAPH Course Notes, 2007.

[DS03] Dachsbacher C., Stamminger M.: Translucent shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2003).

[DS06] Dachsbacher C., Stamminger M.: Splatting indirect illumination. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2006), ACM, pp. 93–100.

[DSDD07] Dachsbacher C., Stamminger M., Drettakis G., Durand F.: Implicit visibility and antiradiance for interactive global illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 26*, 3 (2007), 61.

[DSSD99] Décoret X., Sillion F., Schaufler G., Dorsey J.: Multi-layered impostors for accelerated rendering. *Computer Graphics Forum (Proceedings of Eurographics) 18*, 3 (1999).

[DT07] DeCoro C., Tatarchuk N.: Real-time mesh simplification using the gpu. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2007), ACM, pp. 161–166.

[DVS03] Dachsbacher C., Vogelgsang C., Stamminger M.: Sequential point trees. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 22*, 3 (2003), 657–662.

[ED04] Eisemann E., Durand F.: Flash photography enhancement via intrinsic relighting. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2004), vol. 23, ACM Press, pp. 673–678.

[ED06a] Eisemann E., Décoret X.: Fast scene voxelization and applications. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2006), ACM SIGGRAPH, pp. 71–78.

[ED06b] Eisemann E., Décoret X.: Fast scene voxelization revisited. In *Technical Sketch at SIGGRAPH* (2006), ACM SIGGRAPH.

[ED06c] Eisemann E., Décoret X.: Plausible image based soft shadows using occlusion textures. In *Proceedings of SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing)* (2006), Conference Series, IEEE, IEEE Computer Society, pp. 155–162.

[ED07a] Eisemann E., Décoret X.: On exact error bounds for view-dependent simplification. *Computer Graphics Forum 26*, 2 (2007), 202–213.

[ED07b] Eisemann E., Décoret X.: Visibility sampling on gpu and applications. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3 (2007).

[ED08a] Eisemann E., Décoret X.: Occlusion textures for plausible soft shadows. *Computer Graphics Forum 27*, 1 (2008), 13–23.

[ED08b] Eisemann E., Décoret X.: Single-pass gpu solid voxelization for real-time applications. In *Proceedings of GI (Graphics Interface)* (2008), Canadian Information Processing Society, pp. 73–80.

[EM99] Erikson C., Manocha D.: Gaps: general and automatic polygonal simplification. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (1999), ACM, pp. 79–88.

[Eri96] Erikson C.: *Polygonal Simplification: An Overview.* Tech. Rep. TR96-016, The University of North Carolina at Chapel Hill, 1996.

[Eri00] Erikson C. M.: *Hierarchical levels of detail to accelerate the rendering of large static and dynamic polygonal environments.* PhD thesis, The University of North Carolina at Chapel Hill, 2000. Advisor-Dinesh Manocha.

[ESSS01a] El-Sana J., Sokolovsky N., Silva C. T.: Integrating occlusion culling with view-dependent rendering. In *Proceedings of VIS (IEEE Conference on Visualization)* (2001), IEEE Computer Society.

[ESSS01b] El-Sana J., Sokolovsky N., Silva C. T.: Integrating occlusion culling with view-dependent rendering. In *Proceedings of VIS (IEEE Conference on Visualization)* (2001), IEEE Computer Society, pp. 371–378.

[ESV98] El-Sana J., Varshney A.: Topology simplification for polygonal virtual environments. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (1998), 133–144.

[ESV99] El-Sana J., Varshney A.: Generalized view-dependent simplification. *Computer Graphics Forum (Proceedings of Eurographics) 18*, 3 (1999), 83 – 94.

[Eve01] Everitt C.: Interactive order-independent transparency. webpage sur NVIDIA developpers http://developer.nvidia.com/, 2001.

[EWHS08] Eisemann E., Winnemöller H., Hart J. C., Salesin D.: Stylized vector art from 3d models with region support. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering) 27*, 4 (2008).

[FBP06] Forest V., Barthe L., Paulin M.: Realistic soft shadows by penumbra-wedges blending. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2006), ACM, pp. 39–46.

[FBP08] Forest V., Barthe L., Paulin M.: Accurate Shadows by Depth Complexity Sampling. *Computer Graphics Forum (Proceedings of Eurographics) 27*, 2 (2008), 663–674.

[Fer] Fernando R.: Percentage-closer soft shadows. Technical Sketch at SIGGRAPH / White Paper NVidia.

[FFBG01] Fernando R., Fernandez S., Bala K., Greenberg D. P.: Adaptive shadow maps. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graph-*

ics and Interactive Techniques (2001), ACM Press / ACM SIGGRAPH, pp. 387–390.

[FL00] Fang S., Liao D.: Fast csg voxelization by frame buffer pixel mapping. In *Proceedings of VVS (Symposium on Volume Visualization)* (2000), ACM Press, pp. 43–48.

[Fre00] Frey P. J.: About surface remeshing. In *Proceedings of 9th International Meshing Roundtable* (2000).

[FS93] Funkhouser T. A., Séquin C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques 27*, Annual Conference Series (1993), 247–254.

[FWLG08] Fischer J., Whittaker D., Lefohn A., Gooch B.: Semiautomatic shader code generation for rendering voxelized polygonal models. Poster at SIGGRAPH, 2008.

[GAF07] Ghaffari R., Aranyosi A. J., Freeman D. M.: Longitudinally propagating traveling waves of the mammalian tectorial membrane. In *Proceedings of the National Academy of Sciences of the United States of America* (2007), vol. 104 (42), pp. 16510–5.

[GBP06] Guennebaud G., Barthe L., Paulin M.: Real-time soft shadow mapping by backprojection. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2006), Eurographics, pp. 227–234.

[GBP07] Guennebaud G., Barthe L., Paulin M.: High-Quality Adaptive Soft Shadow Mapping. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3 (2007), 525–534.

[GGSC96] Gortler S. J., Grzeszczuk R., Szeliski R., Cohen M. F.: The lumigraph. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1996), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 43–54.

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1997), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 209–216.

[GH98] Garland M., Heckbert P. S.: Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings of VIS (IEEE Conference on Visualization)* (1998), IEEE, pp. 263–270.

[GHFP08] Gascuel J.-D., Holzschuch N., Fournier G., Peroche B.: Fast non-linear projections using graphics hardware. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2008).

[Gib95] Gibson S. F. F.: Beyond volume rendering: Visualization, haptic exploration, and physical modeling of voxel-based objects. In *Visualization in Scientific Computing*. Springer-Verlag Wien, 1995, pp. 9–24.

[GKM93] Greene N., Kass M., Miller G.: Hierarchical z-buffer visibility. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1993), ACM, pp. 231–238.

[GLL*03] Govindaraju N. K., Lloyd B., Lloyd B., Yoon S.-E., Sud A., Manocha D.: Interactive shadow generation in complex environments. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2003).

[GM05] Gobbetti E., Marton F.: Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2005), ACM, pp. 878–885.

[Gra72] Graham R. L.: An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1* (1972), 132–133.

[Gué99] Guéziec A.: Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 168–189.

[GW07a] Giegl M., Wimmer M.: Fitted virtual shadow maps. In *Proceedings of GI (Graphics Interface)* (2007), Canadian Human-Computer Communications Society, pp. 159–168.

[GW07b] Giegl M., Wimmer M.: Queried virtual shadow maps. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2007), ACM Press, pp. 65–72.

[GW07c] Giegl M., Wimmer M.: Queried virtual shadow maps. In *ShaderX 5 - Advanced Rendering Techniques*. Charles River Media, Inc., 2007.

[GW07d] Giegl M., Wimmer M.: Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. *Computer Graphics Forum 26*, 1 (2007), 46–49.

[GWH01] Garland M., Willmott A., Heckbert P. S.: Hierarchical face clustering on polygonal surfaces. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (2001), ACM, pp. 49–58.

[GwHC97] Gortler S., wei He L., Cohen M. F.: Rendering layered depth images. In *Tech Report MSTR-TR-97-09* (1997).

[HA90] Haeberli P., Akeley K.: The accumulation buffer: Hardware support for high-quality rendering. *Computer Graphics (Proceedings of SIGGRAPH) 24*, 4 (1990), 309–318.

[Hai01] Haines E.: Soft planar shadows using plateaus. *Journal of Graphics Tools 6*, 1 (2001), 19–27.

[Hal02] Hall-Holt, Olaf: Kinetic visibility. *Ph. D. thesis, Department of Computer Science, Stanford University* (2002).

[HAMO05] Hasselgren J., Akenine-Möller T., Ohlsson L.: *GPU Gems 2*. Adison-Wesley, 2005, ch. Conservative Rasterization on the GPU, pp. 677–690.

[HBS00] Heidrich W., Brabec S., Seidel H.: Soft shadow maps for linear lights. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2000), Eurographics Association, pp. 269–280.

[HDD*93] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.: Mesh optimization. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1993), ACM, pp. 19–26.

[HDG99] Hart D., Dutré P., Greenberg D. P.: Direct illumination with lazy visibility evaluation. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 147–154.

[HDKS00] Heidrich W., Daubert K., Kautz J., Seidel H.-P.: Illuminating micro geometry based on precomputed visibility. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 455–464.

[He96] He T.: Volumetric virtual environments, 1996. PhD thesis, State University of New York at Stony Brook.

[Hei91] Heidmann T.: Real shadows, real time. *Iris Universe 18* (1991), 28–31. Silicon Graphics, Inc.

[Hei98] Heidrich W.: View-independent environment maps. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (1998).

[Hei99] Heidrich W.: *High-quality shading and lighting for hardware-accelerated rendering*. PhD thesis, Universität Erlangen, 1999.

[HG97] Heckbert P. S., Garland M.: Survey of polygonal surface simplification algorithms. ACM SIGGRAPH Course Notes, 1997.

[HG99] Heckbert P. S., Garland M.: Optimal triangulation and quadric-based surface simplification. *Computational Geometry 14*, 1-3 (1999), 49–65.

[HH97] Heckbert P. S., Herf M.: *Simulating Soft Shadows with Graphics Hardware*. Tech. Rep. CMU-CS-97-104, Carnegie Mellon University, 1997.

[HHK*95] He T., Hong L., Kaufman A., Varshney A., Wang S.: Voxel based object simplification. In *Proceedings of VIS (IEEE Conference on Visualization)* (1995), pp. 296–303.

[HHLH05] Hornus S., Hoberock J., Lefebvre S., Hart J. C.: Zp+: correct z-pass stencil shadows. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2005), ACM, ACM Press.

[HHVW96] He T., Hong L., Varshney A., Wang S.: Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (1996), 171–184.

[HJ08] Hoberock J., Jia Y.: *GPU Gems 3*. Addison-Wesley, 2008, ch. 12: High-Quality Ambient Occlusion.

[HK06] Harada T., Koshizuka S.: Real-time cloth simulation interacting with deforming high-resolution models. Poster at SIGGRAPH, 2006.

[HLHS03] Hasenfratz J.-M., Lapierre M., Holzschuch N., Sillion F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (2003), 753–774. State-of-the-Art Reviews.

[HLTC05] Hsieh H.-H., Lai Y.-Y., Tai W.-K., Chang S.-Y.: A flexible 3d slicer for voxelization using graphics hardware. In *Proceedings of GRAPHITE (International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia)* (2005), ACM Press, pp. 285–288.

[HMN05] Haumont D., Makinen O., Nirenstein S.: A low dimensional framework for exact polygon-to-polygon occlusion queries. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2005), Eurographics Association, pp. 211–222.

[HN85] Hourcade J.-C., Nicolas A.: Algorithms for antialiased cast shadows. *Computer & Graphics* (1985), 259–265.

[Hop96] Hoppe H.: Progressive meshes. *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques 30*, Annual Conference Series (1996).

[Hop97] Hoppe H.: View-dependent refinement of progressive meshes. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1997).

[Hou62] Hough P.: Method and means for recognizing complex patterns. US Patent 3 069 654, 1962.

[HPH97] Heckbert S., Paul, Herf M.: *Simulating Soft Shadows with Graphics Hardware.* Tech. Rep. CMU-CS-97-104, Carnegie Mellon University, 1997.

[HR05] Hable J., Rossignac J.: Blister: Gpu-based rendering of boolean combinations of free-form triangulated shapes. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2005), ACM Press, pp. 1024–1031.

[Hrb] Hrbek S.: Fast correct soft shadows for nvidia gpus. `http://dee.cz/fcss/`.

[HS99] Heidrich W., Seidel H.-P.: Realistic, hardware-accelerated shading and lighting. *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques 33*, Annual Conference Series (1999), 171–178.

[HSC*05] Hensley J., Scheuermann T., Coombe G., Singh M., Lastra A.: Fast summed-area table generation and its applications. *Computer Graphics Forum (Proceedings of Eurographics) 24*, 3 (2005).

[HSO07] Harris M., Sengupta S., Owens J. D.: Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*. Addison-Wesley, 2007, ch. 39, pp. 851–876.

[HSRG07] Han C., Sun B., Ramamoorthi R., Grinspun E.: Frequency domain normal map filtering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 26*, 3 (2007), 28.

[HW02] Haumont D., Warzee N.: Complete polygonal scene voxelization. *Journal of Graphics Tools 7*, 3 (2002).

[IG03] Isenburg M., Gumhold S.: Out-of-core compression for gigantic polygon meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 22*, 3 (2003), 935–942.

[IKLH04] Ikits M., Kniss J., Lefohn A., Hansen C.: *GPU Gems.* Addison-Wesley, 2004, ch. 39: Volume Rendering Techniques.

[ILGS03] Isenburg M., Lindstrom P., Gumhold S., Snoeyink J.: Large mesh simplification using processing sequences. In *Proceedings of VIS (IEEE Conference on Visualization)* (2003), pp. 465–472.

[IMG00] Isaksen A., McMillan L., Gortler S. J.: Dynamically reparameterized light fields. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 297–306.

[JCLP04] Jakobsen B., Christensen N. J., Larsen B. D., Petersen K. S.: Boundary correct real-time soft shadows. In *Proceedings of CGI (Computer Graphics International)* (2004).

[Jen01] Jensen H. W.: *Realistic Image Synthesis Using Photon Mapping.* AK Peters, Ltd., 2001.

[JLBM05] Johnson G. S., Lee J., Burns C. A., Mark W. R.: The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics 24*, 4 (2005), 1462–1482.

[JW02a] Jeschke S., Wimmer M.: *An Error Metric for Layered Environment Map Impostors.* Tech. Rep. TR-186-2-02-04, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2002.

[JW02b] Jeschke S., Wimmer M.: Textured depth meshes for real-time rendering of arbitrary scenes. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2002), Eurographics Association.

[JWP05] Jeschke S., Wimmer M., Purgathofer W.: Image-based representations for accelerated rendering of complex scenes. In *EUROGRAPHICS 2005 State of the Art Reports* (2005), EUROGRAPHICS, The Eurographics Association and The Image Synthesis Group, pp. 1–20.

[JWS02] Jeschke S., Wimmer M., Schuman H.: Layered Environment-Map Impostors for Arbitrary Scenes. In *Proceedings of GI (Graphics Interface)* (2002).

[Kaj86] Kajiya J. T.: The rendering equation. In *Computer Graphics (Proceedings of SIGGRAPH)* (1986), ACM, pp. 143–150.

[KAMJ05] Kristensen A. W., Akenine-Möller T., Jensen H. W.: Precomputed local radiance transfer for real-time lighting design. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 24*, 3 (2005), 1208–1215.

[KCS08] Kharlamov A., Cantlay I., Stepanenko Y.: *GPU Gems 3.* Addison-Wesley, 2008, ch. 4: Next-Generation SpeedTree Rendering.

[KD03] Kirsch F., Doellner J.: Real-time soft shadows using a single light sample. In *Proceedings of Winter School on Computer Graphics* (2003).

[KH01] Keller A., Heidrich W.: Interleaved sampling. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2001), pp. 269–276.

[KHS04] KOSTER M., HABER J., SEIDEL H.: Real-time rendering of human hair using programmable graphics hardware. In *Proceedings of Computer Graphics International (CGI)* (2004).

[Kil99] Kilgard M. J.: Improving shadows and reflections via the stencil buffer. NVidia white paper `http://developer.nvidia.com/attach/6641` , 1999.

[KJ01] Kolb A., John L.: Volumetric model repair for virtual reality applications, 2001. Short paper Eurographics.

[KKMB96] Kersten D., Knill D. C., Mamassian P., Bülthoff I.: Illusory motion from shadows. *Nature 379*, 31 (1996).

[KL05] Kontkanen J., Laine S.: Ambient occlusion fields. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2005), ACM Press, pp. 41–48.

[KLA04] Kautz J., Lehtinen J., Aila T.: Hemispherical rasterization for self-shadowing of dynamic objects. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Eurographics Association, pp. 179–184.

[KM99] Keating B., Max N.: Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (1999), Springer Computer Science, Eurographics, Eurographics Association, pp. 205–220.

[KN01] Kim T.-Y., Neumann U.: Opacity shadow maps. *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2001).

[Koz04] Kozlov S.: *GPU Gems.* Addison-Wesley, 2004, ch. Perspective Shadow Maps: Care and Feeding.

[KPT99] Karabassi E.-A., Papaioannou G., Theoharis T.: A fast depth-buffer-based voxelization algorithm. *J. Graph. Tools 4*, 4 (1999), 5–10.

[Krü07] Krüger J.: *GI-Edition Lecture Notes in Informatics (LNI).* PhD thesis, Technical University of Munich, 2007.

[KT96] Kalvin A. D., Taylor R. H.: Superfaces: Polygonal mesh simplification with bounded error. *IEEE Comput. Graph. Appl. 16*, 3 (1996), 64–77.

[KTHS06] Kontkanen J., Turquin E., Holzschuch N., Sillion F.: Wavelet radiance transport for interactive indirect lighting. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2006), Eurographics.

[LA05] Laine S., Aila T.: Hierarchical penumbra casting. *Computer Graphics Forum (Proceedings of Eurographics) 24*, 3 (2005), 313–322.

[LAA*05] Laine S., Aila T., Assarson U., Lehtinen J., Akenine-Möller T.: Soft shadow volumes for ray tracing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2005), 1156–1165.

[Lai05] Laine S.: Split-plane shadow volumes. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2005), ACM Press, pp. 23–32.

[Lan02] Landis H.: Production-ready global illumination. In *ACM SIGGRAPH Course Notes* (2002).

[Lat04] Latta L.: Building a million particle system. Lecture at the GDC, 2004.

[Lau08] Lauritzen A.: *GPU Gems 3*. Addison-Wesley, 2008, ch. 8: Summed-Area Variance Shadow Maps.

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH)* (1987), ACM Press, pp. 163–169.

[LD04] Luft T., Deussen O.: Watercolor illustrations of plants using a blurred depth test. In *Proceedings of NPAR (Symposium on Non-Photorealistic Animation and Rendering)* (2004), pp. 11–20.

[LE97] Luebke D., Erikson C.: View-dependent simplification of arbitrary polygonal environments. *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques 31*, Annual Conference Series (1997), 199–208.

[Len] Lengyel E.: Advanced stencil shadow and penumbral wedge rendering. Presentation at Game Developers Conference 2005 http://www.terathon.com/gdc_lengyel.ppt.

[LFWK04] Li W., Fan Z., Wei X., Kaufman A.: *GPU Gems 2*. Addison-Wesley, 2004, ch. 47: Simulation with Complex Boundaries.

[LGMM07] Lloyd D. B., Govindaraju N. K., Molnar S. E., Manocha D.: Practical logarithmic rasterization for low-error shadow maps. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2007), Eurographics Association, pp. 17–24.

[LH96] Levoy M., Hanrahan P.: Light field rendering. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1996), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 31–42.

[LH08] Liao D., Hahn J. K.: Real-time solid voxelization by slice functions. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2008), ACM, pp. 1–1.

[LHN05a] Lefebvre S., Hornus S., Neyret F.: Texture sprites: texture elements splatted on surfaces. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2005), ACM, pp. 163–170.

[LHN05b] Lefebvre S., Hornus S., Neyret F.: Texture sprites: Texture elements splatted on surfaces. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2005), ACM SIGGRAPH, ACM Press.

[LHSW03] Losasso F., Hoppe H., Schaefer S., Warren J.: Smooth geometry images. In *Proceedings of SGP (Eurographics/ACM SIGGRAPH Symposium on Geometry Processing)* (2003), Eurographics Association, pp. 138–145.

[Lin00] Lindstrom P.: Out-of-core simplification of large polygonal models. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 259–262.

[Lis98] Lischinski D.: Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (1998), pp. 301–314.

[LKR*96] Lindstrom P., Koller D., Ribarsky W., Hodges L., Faust N., Turner G.: Real-time continuous level of detail rendering of height fields. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1996).

[LKS*06] Lefohn A., Kniss J. M., Strzodka R., Sengupta S., Owens J. D.: Glift: Generic, efficient, random-access gpu data structures. *ACM Transactions on Graphics 25*, 1 (2006), 60–99.

[LLA06] Lehtinen J., Laine S., Aila T.: An improved physically-based soft shadow volume algorithm. *Computer Graphics Forum (Proceedings of Eurographics) 25*, 3 (2006), 303–312.

[Lla07] Llamas I.: Real-time voxelization of triangle meshes on the gpu. In *Technical Sketch at SIGGRAPH* (2007), ACM Press.

[Llo06] Lloyd S.: *Programming the Universe: A Quantum Computer Scientist Takes On the Cosmos*. Knopf, 2006.

[LM08] Lauritzen A., McCool M.: Layered variance shadow maps. In *Proceedings of GI (Graphics Interface)* (2008).

[LPRM02] Lévy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 21*, 3 (2002), 362–371.

[LR82] Lee Y., Requicha A. A. G.: Algorithms for computing the volume and other integral properties of solids. In *Communications of the ACM, 25(9):635 650* (1982).

[LRC*03] Luebke D., Reddy M., Cohen J., Varshney A., Watson B., Huebner R.: *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers Inc., 2003.

[LS01] Lindstrom P., Silva C. T.: A memory insensitive technique for large model simplification. In *Proceedings of VIS (IEEE Conference on Visualization)* (2001), IEEE Computer Society, pp. 121–126.

[LS04] Lin Z., Shum H.-Y.: A geometric analysis of light field rendering. *International Journal of Computer Vision 58*, 2 (2004), 121–138.

[LSCO03] Leyvand T., Sorkine O., Cohen-Or D.: Ray space factorization for from-region visibility. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 22*, 3 (2003), 595–604.

[LSK*05] Lefohn A., Sengupta S., Kniss J. M., Strzodka R., Owens J. D.: Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH Conference Abstracts and Applications* (2005).

[LSK*07] Laine S., Saransaari H., Kontkanen J., Lehtinen J., Aila T.: Incremental instant radiosity for real-time indirect illumination. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), Eurographics Association.

[LSO07] Lefohn A. E., Sengupta S., Owens J. D.: Resolution matched shadow maps. *ACM Transactions on Graphics 26*, 4 (2007), 20:1–20:17.

[LT97] Low K.-L., Tan T.-S.: Model simplification using vertex-clustering. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (1997), ACM, pp. 75–ff.

[LT98] Lindstrom P., Turk G.: Fast and memory efficient polygonal simplification. In *Proceedings of VIS (IEEE Conference on Visualization)* (1998), IEEE Computer Society Press, pp. 279–286.

[LT99] Lindstrom P., Turk G.: Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 98–115.

[LT00] Lindstrom P., Turk G.: Image-driven simplification. *ACM Transactions on Graphics 19*, 3 (2000).

[LTF*05] Liu C., Torralba A., Freeman W. T., Durand F., Adelson E. H.: Motion magnification. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2005), ACM, pp. 519–526.

[LTYM06] Lloyd B., Tuft D., Yoon S., Manocha D.: Warping and partitioning for low error shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2006), Eurographics Association.

[LV00] Lokovic T., Veach E.: Deep shadow maps. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co.

[LWGM04] Lloyd B., Wendt J., Govindaraju N. K., Manocha D.: Cc shadow volumes. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Springer Computer Science, Eurographics, Eurographics Association.

[LWX06] Liu B., Wei L.-Y., Xu Y.-Q.: *Multi-Layer Depth Peeling via Fragment Sort.* Tech. Rep. MSR-TR-2006-81, Microsoft Research, 2006.

[LZK*07] Lehtinen J., Zwicker M., Kontkanen J., Turquin E., Sillion F., Aila T.: *Meshless Finite Elements for Hierarchical Global Illumination.* Tech. Rep. TML-B7, Helsinki University of Technology, 2007.

[MAM05] Mora F., Aveneau L., Mériaux M.: Coherent and exact polygon-to-polygon visibility. In *Proceedings of WSCG (International Conference on Computer Graphics, Visualization and Computer Vision)* (2005).

[MB07] Myers K., Bavoil L.: Stencil routed a-buffer. In *Technical Sketch at SIGGRAPH* (2007), ACM, p. 21.

[McC00] McCool M. D.: Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics 19*, 1 (2000), 1–26.

[McM97] McMillan L.: *An Image-based Approach to Three-Dimensional Computer Graphics.* PhD thesis, University of North Carolina at Chapel Hill, 1997.

[Mil94] Miller G.: Efficient algorithms for local and global accessibility shading. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1994), ACM, pp. 319–326.

[MJW07] Mantler S., Jeschke S., Wimmer M.: *Displacement Mapped Billboard Clouds.* Tech. Rep. TR-186-2-07-01, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2007.

[ML94] Marschner S. R., Lobb R. J.: An evaluation of reconstruction filters for volume rendering. In *Proceedings of VIS (IEEE Conference on Visualization)* (1994), pp. 100–107.

[MMAH07] Malmer M., Malmer F., Assarsson U., Holzschuch N.: Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools 12*, 2 (2007), 59–71.

[MMMY97] Möller T., Machiraju R., Mueller K., Yagel R.: A comparison of normal estimation schemes. In *Proceedings of VIS (IEEE Conference on Visualization)* (1997), IEEE Computer Society Press, pp. 19–ff.

[MPW07] Mo Q., Popescu V., Wyman C.: The soft shadow occlusion camera. In *Proceedings of the Pacific Conference on Computer Graphics and Applications* (2007), IEEE Computer Society, pp. 189–198.

[MS95] Maciel P. W. C., Shirley P.: Visual navigation of large environments using textured clusters. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (1995), ACM Press.

[MT04] Martin T., Tan T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Springer Computer Science, Eurographics, Eurographics Association.

[MW07] Mo Q., Wyman C.: Interactive backprojected soft shadows with an occlusion camera shadow map. In *Poster at SIGGRAPH* (2007), ACM, p. 179.

[NB04] Nirenstein S., Blake E.: Hardware accelerated aggressive visibility preprocessing using adaptive sampling. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Eurographics Association, pp. 207–216.

[NBA04] Ni R., Braunstein M. L., Andersen G. J.: Interaction of optical contact, shadows and motion in determining perceived scene layout. *Journal of Vision 4*, 8 (2004), 615–615.

[NBG02] Nirenstein S., Blake E. H., Gain J. E.: Exact from-region visibility culling. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2002), pp. 191 – 202.

[ND05] Nguyen H., Donnelly W.: *GPU Gems 2*. Addison-Wesley, 2005, ch. 23: Hair Animation and Rendering in the Nalu Demo.

[NLB*05] Ng R., Levoy M., Brdif M., Duval G., Horowitz M., Hanrahan P.: *Light field photography with a hand-held plenoptic camera*. Tech. Rep. CSTR 2005-02, Stanford University, 2005.

[NPC07] Niski K., Purnomo B., Cohen J.: Multi-grained level of detail using a hierarchical seamless texture atlas. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2007), ACM, pp. 153–160.

[NT03] Nooruddin F. S., Turk G.: Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 191–205.

[ORM07] Overbeck R., Ramamoorthi R., Mark W. R.: A Real-time Beam Tracer with Application to Exact Soft Shadows . In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), vol. 18 of *Eurographics / ACM SIGGRAPH Symposium Proceedings*, Eurographics, pp. 85–98.

[OSW*05] Opengl, Shreiner D., Woo M., Neider J., Davis T.: *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, 2005.

[OTT98] Ong K. C., Teh H. C., Tan T. S.: Resolving occlusion in image sequence made easy. *The Visual Computer 14*, 4 (1998), 153–165.

[OZ06] Olson M., Zhang H.: Silhouette Extraction in Hough Space. *Computer Graphics Forum (Proceedings of Eurographics) 25*, 3 (2006), 273–282.

[PCdON04] Pagot C. A., Comba J. L. D., de Oliveira Neto M. M.: Multiple-depth shadow maps. In *Proceedings of SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing)* (2004), IEEE Computer Society, pp. 308–315.

[PD06] Paris S., Durand F.: A fast approximation of the bilateral filter using a signal processing approach. In *Proceedings of European Conference on Computer Vision (ECCV) 2006* (2006).

[PO07] Policarpo F., Oliveira M.: *GPU Gems 3*. Addison-Wesley, 2007, ch. 18: Relaxed Cone Stepping For Relief Mapping.

[POJ05] Policarpo F., Oliveira M. M., Jo a. L. D. C.: Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2005), ACM, pp. 155–162.

[PSA*04] Petschnigg G., Szeliski R., Agrawala M., Cohen M., Hoppe H., Toyama K.: Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 23*, 3 (2004), 664–672.

[PSS98] Parker S., Shirley P., Smits B.: *Single sample soft shadows.* Tech. Rep. UUCS-98-019, University of Utah, 1998.

[PV06] Plantinga S., Vegter G.: Computing contour generators of evolving implicit surfaces. *ACM Transactions on Graphics 25*, 4 (2006), 1243–1280.

[RB93] Rossignac J., Borrel P.: Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications (Proceedings of the Second Conference on Geometric Modeling in Computer Graphics* (1993), Springer-Verlag, pp. 455–465.

[RBB03] Reitinger B., Bornik A., Beichel R.: Efficient volume measurement using voxelization. In *Proceedings of SCCG (Spring Conference on Computer Graphics)* (2003), ACM, pp. 47–54.

[RGKM07] Ritschel T., Grosch T., Kautz J., Mueller S.: Interactive illumination with coherent shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), Eurographics Association, pp. 61–72.

[RGKS08] Ritschel T., Grosch T., Kautz J., Seidel H.-P.: Interactive global illumination based on coherent surface shadow maps. In *Proceedings of GI (Graphics Interface)* (2008).

[RR96] Ronfard R., Rossignac J.: Full-range approximation of triangulated polyhedra. *Computer Graphics Forum (Proceedings of Eurographics) 15*, 3 (1996), 67–76.

[RSC87] Reeves W. T., Salesin D., Cook R. L.: Rendering antialiased shadows with depth maps. *Computer Graphics (Proceedings of SIGGRAPH)* (1987), 283–291.

[RT06] Rong G., Tan T.-S.: Utilizing jump flooding in image-based soft shadows. In *Proceedings of VRST (ACM Symposium on Virtual Reality Software and Technology)* (2006), ACM, pp. 173–180.

[RWS*06] Ren Z., Wang R., Snyder J., Zhou K., Liu X., Sun B., Sloan P.-P., Bao H., Peng Q., Guo B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 25*, 3 (2006), 977–986.

[SAPP05] St-Amour J.-F., Poulin P., Paquette E.: Soft shadows from extended light sources with penumbra deep shadow maps. In *Proceedings of GI (Graphics Interface)* (2005), pp. 105–112.

[SB95] Smith S. M., Brady J. M.: *SUSAN – A new approach to low level image processing.* Tech. Rep. TR95SMS1c, Chertsey, Surrey, UK, 1995.

[Sch94] SCHLICK C.: A survey of shading and reflectance models. *Computer Graphics Forum 13*, 2 (1994), 121–131.

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 22*, 3 (2003), 521–526.

[Sch06] SCHÜLER C.: Eliminate surface acne with gradient shadow mapping. In *ShaderX 4 - Advanced Rendering Techniques*. Charles River Media, Inc., 2006.

[SCLR99] STARK M. M., COHEN E., LYCHE T., RIESENFELD R. F.: Computing exact shadow irradiance using splines. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 155–164.

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2002), 557–562.

[SDB97] SILLION F., DRETTAKIS G., BODELET B.: Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum (Proceedings of Eurographics) 16*, 3 (1997).

[SDDS00] SCHAUFLER G., DORSEY J., DÉCORET X., SILLION F. X.: Conservative volumetric visibility with occluder fusion. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 229–238.

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering) 27*, 4 (2008).

[SEH08] STROILA M., EISEMANN E., HART J. C.: Clip art rendering of smooth isosurfaces. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (2008), 135–145.

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2004), ACM, pp. 65–73.

[SGG*00] SANDER P. V., GU X., GORTLER S. J., HOPPE H., SNYDER J.: Silhouette clipping. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 327–334.

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3d distance field computation using linear factorization. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2006), ACM Press.

[SGwHS98] SHADE J., GORTLER S., WEI HE L., SZELISKI R.: Layered depth images. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1998), ACM Press.

[SH93] SCHRÖDER P., HANRAHAN P.: On the form factor between two polygons. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1993), ACM, pp. 163–164.

[Shi06] Shishkovtsov O.: *GPU Gems 2*. Addison-Wesley, 2006, ch. Deferred Shading in STALKER.

[Shr] Shrout R.: Rendering games with raytracing will revolutionize graphics - pc perspective. http://www.pcper.com/article.php?aid=455&type=expert&pid=3.

[SHSG01] Sander P., Hoppe H., Snyder J., Gortler S.: Discontinuity edge overdraw. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (2001), ACM SIGGRAPH, pp. 167–174.

[Sil08] Silva M.: Exponential shadow maps. In *ShaderX 6 - Advanced Rendering Techniques*. Charles River Media, Inc., 2008.

[Sim94] Simpson R. B.: Anisotropic mesh transformations and optimal error control. *Applied Numerical Mathematics: Transactions of IMACS 14*, 1–3 (1994), 183–198.

[SIMP06] Segovia B., Iehl J.-C., Mitanchey R., Péroche B.: Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)* (2006).

[SJW07] Scherzer D., Jeschke S., Wimmer M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), Eurographics, Eurographics Association, pp. 45–50.

[SLS*96] Shade J., Lischinski D., Salesin D. H., DeRose T., Snyder J.: Hierarchical image caching for accelerated walkthroughs of complex environments. *Computer Graphics 30*, Annual Conference Series (1996), 75–82.

[SM06] Sander P. V., Mitchell J. L.: Progressive buffers: view-dependent geometry and texture lod rendering. In *ACM SIGGRAPH Course Notes* (2006), ACM, pp. 1–18.

[Sol98] Soler C.: *Représentations hiérarchiques de la visibilité pour le contrôle de l'erreur en simulation de l'éclairage*. PhD thesis, Université Joseph Fourier (Grenoble), 1998.

[SOM04] Sud A., Otaduy M., Manocha D.: Difi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum (Proceedings of Eurographics) 23*, 3 (2004).

[Sou05] Sousa T.: *GPU Gems 2*. Addison-Wesley, 2005, ch. 19: Generic Refraction Simulation.

[SP94] Sillion F. X., Puech C.: *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc., 1994.

[SPG03] Sigg C., Peikert R., Gross M.: Signed distance transform using graphics hardware. In *Proceedings of VIS (IEEE Conference on Visualization)* (2003), IEEE Computer Society Press, pp. 83–90.

[SS95] Schaufler G., Stürzlinger W.: Generating multiple levels of detail from polygonal geometry models. In *Virtual Environments (Eurographics Workshop)* (1995), Springer-Verlag: Heidelberg, Germany, pp. 33–41.

[SS96] Schaufler G., Stürzlinger W.: A three dimensional image cache for virtual reality. *Computer Graphics Forum (Proceedings of Eurographics) 15*, 3 (1996), 227–236.

[SS98] Soler C., Sillion F.: Fast calculation of soft shadow textures using convolution. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1998), *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 321–332.

[SS07] Schwarz M., Stamminger M.: Bitmask soft shadows. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3 (2007), 515–524.

[SS08] Schwarz M., Stamminger M.: Quality scalability of soft shadow mapping. In *Proceedings of GI (Graphics Interface)* (2008), Canadian Information Processing Society, pp. 147–154.

[SSGH01] Sander P. V., Snyder J., Gortler S. J., Hoppe H.: Texture mapping progressive meshes. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (2001), ACM, pp. 409–416.

[ST90] Saito T., Takahashi T.: Comprehensible rendering of 3-d shapes. *Computer Graphics (Proceedings of SIGGRAPH) 24*, 4 (1990), 197–206.

[SU] Stanford University, University of Washington: The Michelangelo Project. http://graphics.stanford.edu/projects/mich/ .

[Sut63] Sutherland I.: *Sketchpad: A man-machine graphical communication system.* PhD thesis, Massachusetts Institute of Technology, 1963.

[SW08] Scherzer D., Wimmer M.: Frame sequential interpolation for discrete level-of-detail rendering. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering) 27*, 4 (2008),

[SWK07] Stich M., Wächter C., Keller A.: *GPU Gems 3.* Addison-Wesley, 2007, ch. 11: Efficient and Robust Shadow Volumes.

[SZL92] Schroeder W. J., Zarge J. A., Lorensen W. E.: Decimation of triangle meshes. In *Computer Graphics (Proceedings of SIGGRAPH)* (1992), ACM, pp. 65–70.

[Tes] Tessendorf D.: principal graphics scientist at rhythm and hues studios. private conversation.

[TH03] Tam R., Heidrich W.: Shape simplification based on the medial axis transform. In *Proceedings of VIS (IEEE Conference on Visualization)* (2003), pp. 481–488.

[The] The Free Dictionary: Online Dictionary. http://www.thefreedictionary.com/ .

[TM98] Tomasi C., Manduchi R.: Bilateral filtering for gray and color images. In *ICCV* (1998), pp. 839–846.

[TO96] T. Ohshima H. Yamamoto H. T.: Gaze-directed adaptive rendering for interacting with virtual space. In *Proceedings of the IEEE Virtual Reality Annual International Symposium* (1996), pp. 103–110.

[Tur92] TURK G.: Re-tiling polygonal surfaces. *SIGGRAPH Comput. Graph. 26*, 2 (1992), 55–64.

[Ura05] URALSKY Y.: Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2* (2005), Addison Wesley.

[Vis] VISCOG PRODUCTIONS INC.: initially Visual Cognition Lab at the University of Illinois (UIUC). http://www.viscog.com/ .

[WD06] WYMAN C., DAVIS S.: Interactive image-space techniques for approximating caustics. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)* (2006), ACM, pp. 153–160.

[WE03] WEISKOPF D., ERTL T.: Shadow Mapping Based on Dual Depth Layers. In *Short Paper Eurographics* (2003), pp. 53–60.

[WH03] WYMAN C., HANSEN C.: Penumbra maps: Approximate soft shadows in real-time. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2003), Springer Computer Science, Eurographics, Eurographics Association, pp. 202–207.

[WIK*06] WALD I., IZE T., KENSLER A., KNOLL A., PARKER S. G.: Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2006), 485–493.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH)* (1978), ACM Press.

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *Computer Graphics (Proceedings of SIGGRAPH)* (1983), ACM Press, pp. 1–11.

[WK93] WANG S. W., KAUFMAN A. E.: Volume sampled voxelization of geometric primitives. In *Proceedings of VIS (IEEE Conference on Visualization)* (1993), pp. 78–84.

[WLC*03] WILLIAMS N., LUEBKE D., COHEN J. D., KELLEY M., SCHUBERT B.: Perceptually guided simplification of lit, textured meshes. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)* (2003), ACM, pp. 113–121.

[WM94] WANG Y., MOLNAR S.: *Second-Depth Shadow Mapping.* Tech. Rep. TR 94-019, University of North Carolina at Chapel Hill, 1994.

[WM03] WILSON A., MANOCHA D.: Simplifying complex environments using incremental textured depth meshes. In *Proceedings of the ACM Siggraph* (2003), ACM Press.

[Woo92] WOO A.: *Graphics Gems III.* Academic Press Professional, Inc., 1992, ch. The shadow depth map revisited, pp. 338–342.

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Computer Graphics and Applications 10*, 6 (1990), 13–32.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2004), Springer Computer Science, Eurographics, Eurographics Association.

[WWHR97] Watson B., Walker N., Hodges L. F., Reddy M.: A user study evaluating level of detail degradation in the periphery of head-mounted displays. *Presence: Teleoperators and Virtual Environments 6*, 6 (1997).

[WWS00] Wonka P., Wimmer M., Schmalstieg D.: Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (2000), pp. 71–82.

[WWZ*06] Wonka P., Wimmer M., Zhou K., Maierhofer S., Hesina G., Reshetov A.: Guided visibility sampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH) 25*, 3 (2006), 494–502.

[Wym05] Wyman C.: An approximate image-space approach for interactive refraction. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2005), ACM Press.

[XTP07] Xie F., Tabellion E., Pearce A.: Soft Shadows by Ray Tracing Multilayer Transparent Shadow Maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* (2007), Eurographics Association, pp. 265–276.

[XV96] Xia J. C., Varshney A.: Dynamic view-dependent simplification for polygonal models. In *Proceedings of VIS (IEEE Conference on Visualization)* (1996), pp. 335–344.

[YSG05] Yoon S.-E., Salomon B., Gayle R.: Quick-vdr: Out-of-core view-dependent rendering of gigantic models. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 369–382. Member-Dinesh Manocha.

[YTD02] Ying Z., Tang M., Dong J.: Soft shadow maps for area light by area approximation. In *Proceedings of the Pacific Conference on Computer Graphics and Applications* (2002), IEEE Computer Society, p. 442.

[ZCEP07] Zhang L., Chen W., Ebert D. S., Peng Q.: Conservative voxelization. *The Visual Computer 23*, 9-11 (2007), 783–792.

[ZG02] Zelinka S., Garland M.: Permission grids: practical, error-bounded simplification. *ACM Transactions on Graphics 21*, 2 (2002), 207–229.

[Zha98] Zhang H.: Forward shadow mapping. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)* (1998), Springer Computer Science, Eurographics, Eurographics Association, pp. 131–138.

[ZHL*05] Zhou K., Hu Y., Lin S., Guo B., Shum H.-Y.: Precomputed shadow fields for dynamic scenes. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2005).

[ZT02] Zhang E., Turk G.: Visibility-guided simplification. In *Proceedings of VIS (IEEE Conference on Visualization)* (2002), IEEE Computer Society, pp. 267–274.

[ZXTS06] Zhang F., Xu L., Tao C., Sun H.: Generalized linear perspective shadow map reparameterization. In *Proceedings of VRCIA (International Conference on Virtual Reality Continuum and Its Applications)* (2006), ACM, pp. 339–342.

List of Figures

Curriculum Vitae

Elmar EISEMANN

Marsdorfer Str. 23

50858 Köln

Born on the 25th July 1978
Nationality : German

Web: http://artis.imag.fr/Members/Elmar.Eisemann/
Email: Elmar.Eisemann@imag.fr

Formation

Since 2005	**PhD. student** at INRIA Rhône-Alpes under the supervision of Xavier Décoret and François Sillion
2004	**Magistère in Computer Science** from Paris XI
2003-2004	**Master** IVR (Image Vision Robotics) in Computer Science from INPG (Institut National Polytechnique de Grenoble), best student
2001-2002	**Licence** in Computer Science from Paris XI **Maîtrise** in Computer Science from Paris XI
2001	Accepted at École Normale Supérieure, Paris (**Concours ENS-Europe)**
2001	**"Vordiplom der Mathematik"** at the University of Cologne
1999-2001	**Student in Mathematics** at the University of Cologne
1998-1999	Civil service
1998	"**Abitur**": Georg-Büchner Gymnasium

Professional Experience

2007	**Internship**: three months at Adobe Systems Inc. (Seattle) under the supervision of David Salesin and John C. Hart
2006	**Internship**: three months at the University of Illinois (Urbana-Champaign) under the supervision of John C. Hart
2003	**Internship**: six months at the Massachusetts Institute of Technology (MIT) under the supervision of Frédo Durand
2002	**Internship**: two months at INRIA Rhône-Alpes (Grenoble) under the supervision of Cyril Soler

Publications

Elmar Eisemann, Holger Winnemöller, John C. Hart, David Salesin:
Stylized Vector Art from 3D Models with Region support
Computer Graphics Forum (Proceedings of EGSR), volume 27, number 4 - 2008

Erik Sintorn, Elmar Eisemann, Ulf Assarsson:
Sample-Based Visibility for Soft Shadows Using Alias-Free Shadow Maps
Computer Graphics Forum (Proceedings of EGSR), volume 27, number 4 - 2008

Hedlena Bezerra, Elmar Eisemann, Xavier Décoret, Joëlle Thollot:
3D Dynamic Grouping For Guided Stylization,
Proceedings of NPAR-Symposium on Non-Photorealistic Animation and Rendering - 2008

Elmar Eisemann, Xavier Décoret:
Single-Pass GPU Solid Voxelization for Real-Time Applications,
Proceedings of Graphics Interfaces (GI) - 2008

Elmar Eisemann, Xavier Décoret:
Occlusion Textures for Plausible Soft Shadows,
Computer Graphics Forum, volume 27, number 1 - 2008

Matei Stroila, Elmar Eisemann, John C. Hart:
Clip Art Rendering of Smooth Isosurfaces,
Transactions on Visualization and Computer Graphics (TVCG), volume 14, number 1 - 2008

Elmar Eisemann, Xavier Décoret:
Visibility Sampling on GPU and Applications,
Computer Graphics Forum (Proceedings of Eurographics), volume 26, number 3 - 2007

Elmar Eisemann, Xavier Décoret:
On Exact Error Bounds for View-Dependent Simplification,
Computer Graphics Forum, volume 26, number 2 - 2007

Elmar Eisemann, Xavier Décoret:
Fast Scene Voxelization Revisited,
Technical Sketch ACM SIGGRAPH - 2006

Elmar Eisemann, Xavier Décoret:
Plausible Soft Shadows using Occlusion Textures,
Proceedings of SIBGRAPI-Brazilian Symposium on Computer Graphics and Image Processing - 2006

Elmar Eisemann, Xavier Décoret:
Fast Scene Voxelization and its Applications,
Proceedings of I3D-ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - 2006

Elmar Eisemann, Frédo Durand:
Flash Photography Enhancement via intrinsic relighting,
ACM Transactions on Graphics (Proceedings of SIGGRAPH), Volume 23 - 2004

Awards and Recognition

2006 SIBGRAPI **best conference paper award** (2 papers have been selected)
2006 I3D **paper reprised at SIGGRAPH** (special session with three I3D papers), new version is part of the SIGGRAPH digital library
1998 **Award** from the industry for the best chemistry student

Teaching at the University of Grenoble

The course names have been translated from French to English

2007 / 2008:

Project leader: "**Human-Computer Interaction**" - Master 1
Assistant: "**Plasticity (Human-Computer Interaction)**" - Master 1
Assistant: "**Multimedial Information Retrieval**" - L3

Additional teaching activities:
"**Classes passerelles**" system for students with schoolphobia
"**Image, Vision, Robotics**" - Master 2

2006 / 2007:

Responsible for the course: "**Image Synthesis**" - Master 2
Assistant: "**Multimedial Information Retrieval**" - L3

2005 / 2006:

Responsible for the course: "**Computer Graphics**" - Master 1
Responsible for the course: "**Image Synthesis**" - Master 2