ADAPTIVE TECHNIQUES FOR HARDWARE SHADOW GENERATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Pemith Randima Fernando

May 2002

ABSTRACT

This thesis presents two adaptive algorithms for shadow generation. Both algorithms utilize commercial graphics hardware to accelerate the rendering process. The first algorithm, called Adaptive Shadow Maps, deals with removing the aliasing artifacts that typically result when using shadow maps for hard shadow generation. It achieves this by varying the shadow map resolution spatially throughout the scene based on the eye position. The second algorithm, called Adaptive Soft Shadows, attempts to generate soft shadows efficiently by varying the number of samples needed over the different regions of the scene. More samples are devoted to soft shadow regions that subtend a large portion of the image plane, and fewer samples are devoted to hard shadow regions.

We show that Adaptive Shadow Maps enable dramatic improvements in shadow quality while maintaining interactive rates and being constrained to a user-specified memory limit. In the case of Adaptive Soft Shadows, we examine the reasons why the approach was not as successful as we had envisioned and discuss possible avenues for improvement. The motivation for each algorithm is presented along with the corresponding theoretical foundation, implementation, results, conclusions, and directions for improvement.

BIOGRAPHICAL SKETCH

The author was born on December 25, 1978 in Colombo, Sri Lanka. After four years, his family moved to Manila in the Philippines, where he spent the next thirteen years being educated at the International School, Manila. In the fall of 1996, after applying to various colleges in the United States, the author perplexingly decided to enroll in the outstanding educational institution known as Cornell University (located in glacial Ithaca, New York). After completing his Bachelors degree in computer science, he joined the Cornell University Program of Computer Graphics as a Masters Student in January 2000.

To my parents and my sister for their unwavering support and belief.

ACKNOWLEDGMENTS

First of all, I must thank Professor Donald P. Greenberg, who gave me the opportunity to join the Cornell Program of Computer Graphics and to try my hand at research. Don, I am also grateful to you for the excellent advice you have given me and for the many lessons I have learned from you. I hope that I have provided you with at least a little entertainment during my stay here.

I am also thankful to a number of people for their kindness during my time here. To be concise, I wanted to mention something small for each person, so here goes… As the lawyers say, this is a representative list that includes but is not limited to the following people and reasons:

- Peggy Andersen for helping to keep the payroll running smoothly and for sharing various pieces of wisdom with me.
- Mary Armstrong for keeping our offices clean and for bringing in many tasty treats from time to time.
- Kavita Bala for listening to my ideas, giving comments, and for a great deal of excellent advice.
- Martin Berggren for teaching me something about video equipment.
- Steve Berman for helping me to test out my new tennis racket in the gorgeous summer of 2001.
- Mitch Collinsworth for keeping the network running smoothly.
- Jonathan Corson-Rikert for swiftly handling any and all administrative issues during his tenure here.

- Phil Dutré for stimulating my mind with various odd board games that I grew to enjoy, and for sharing delectable Belgian chocolates with all of us.
- Reynald Dumont and his wife for giving me some early practice in baby sitting, running around the fifth floor of Rhodes Hall with their son.
- Sebastian Fernandez for making me feel comfortable at PCG from the time I began here, and for being a very understanding and empathetic friend. I am also grateful to Spiff's fiancée Sylvina, who sent many a delectable concoction through Spiff to satisfy our voracity.
- Jim Ferwerda for patiently listening and nodding politely while I explained my various theories relating performance in chess games to… just about anything.
- Suanne Fu for her cheerful ebullience, which helped to offset Spiff's surly nature (just kidding Spiff!)
- Vikash Goel for being a calm and pacifying force. I hope he will still be the same way when he leaves…
- Eric Haines for taking the time to comment on the original shadow map paper, his excellent advice, and for spreading hope with his cheerful, uplifting nature.
- Ryan Ismert for his helpful advice during the job hunting process.
- Rich Levy for helping me to make the original submission video for Adaptive Shadow Maps during a very pressing time.
- Hongsong Li for imparting much ancient wisdom, and for many interesting table tennis conversations and practices.
- John Mollis for being eager to play golf. It's an interesting game…

- Sumant Pattanaik for two very entertaining table tennis matches during the First PCG Table Tennis Tournament.
- Fabio Pellacini for always taking time out to explain concepts in clear and precise detail to me, and for always being willing to hop over and try to help, particular when I had questions about Microsoft products.
- Moreno Piccolotto for dispensing coffee to Kavita, Spiff, Ryan, and Suanne, helping to keep them happy and satiated, and thereby preserving the lives of the others in the lab.
- Jeremy Selan for being a wonderful supporter of the mini-golf movement. Jeremy, I hope you will preserve it, as difficult as it may seem to do so.
- Hurf Sheldon for believing in me and giving me the equipment I needed to try out new ideas.
- Linda Stephenson for making it easier to meet with Don despite his extremely busy schedule. Linda, I am quite sure that I would not have been able to graduate without your help.
- Parag Tole for suggesting some excellent improvements that I implemented for the ASM algorithm, including the idea of using a mesh-like object to illustrate extreme aliasing.
- Ben Trumbore for being the greatest supporter of my golfing education.
- Bruce Walter for sharing a great deal of wisdom, stimulating my mind, and being wonderfully patient with me. I don't know if Bruce will miss our tangential conversations and debates, but I know I will.
- Steve Westin for a number of useful suggestions during my ASM practice talks.

- Hector Yee for his great help during my job search in a difficult job market.

I wish everyone the very best for the future.

I would also like to thank David Kirk, Nick Triantos, John Spitzer, and Mark Kilgard from NVIDIA for their responses to my various questions over the course of my two years here.

I am grateful to my friends (most of whom have since graduated), and particularly to Joy Alamgir and Virantha Ekanayake for their support during my time at Cornell. In addition, I would like to thank Professor Bruce Land for his support and advice over the many years that I have known him here at Cornell. Thank you also to Professor Keshav Pingali for being a very accommodating and understanding member of my degree committee.

Finally, I would like to thank my parents and my sister who have been there for me all along. I am especially grateful to my parents for teaching me so much and for giving me the best opportunity they could in the great odyssey of life.

The work in this thesis was supported in part by the National Science Foundation Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219) and the generous support of the Intel and NVIDIA corporations.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 3

LIST OF FIGURES

CHAPTER 1

CHAPTER 2

CHAPTER 3

Chapter 1

Introduction

We begin with a look at shadows and their importance in realistic images. Next, we look at different shadow generation algorithms, classified by the type of shadows they produce. Following this, we introduce shadow maps and explain their advantages and disadvantages, concluding with a brief look at the organization of this thesis.

A Brief History of Shadows

The Importance of Shadows

Shadows provide important information about the spatial relationships among objects [Wanger92]. They provide clues about the objects that allow us to perceive depth, the distances between objects, the illuminating light sources, and their occluders as well as their shapes. Figure 1.1 shows that shadows are an important part of making realistic images. The first panel of the figure shows that without shadows, it is impossible to make any conclusions about the positions of the red and blue teapots. Even though they are at the same location in the image plane, we cannot discern their positions in world space. The second panel shows the same scene with shadows from a

point light source. Now we can see that the blue teapot is actually nearer to us, and is floating in space. Finally, in the third panel, an area light source has been used, resulting in realistic-looking soft shadows. Clearly the existence and quality of shadows plays a large role in making images look realistic.

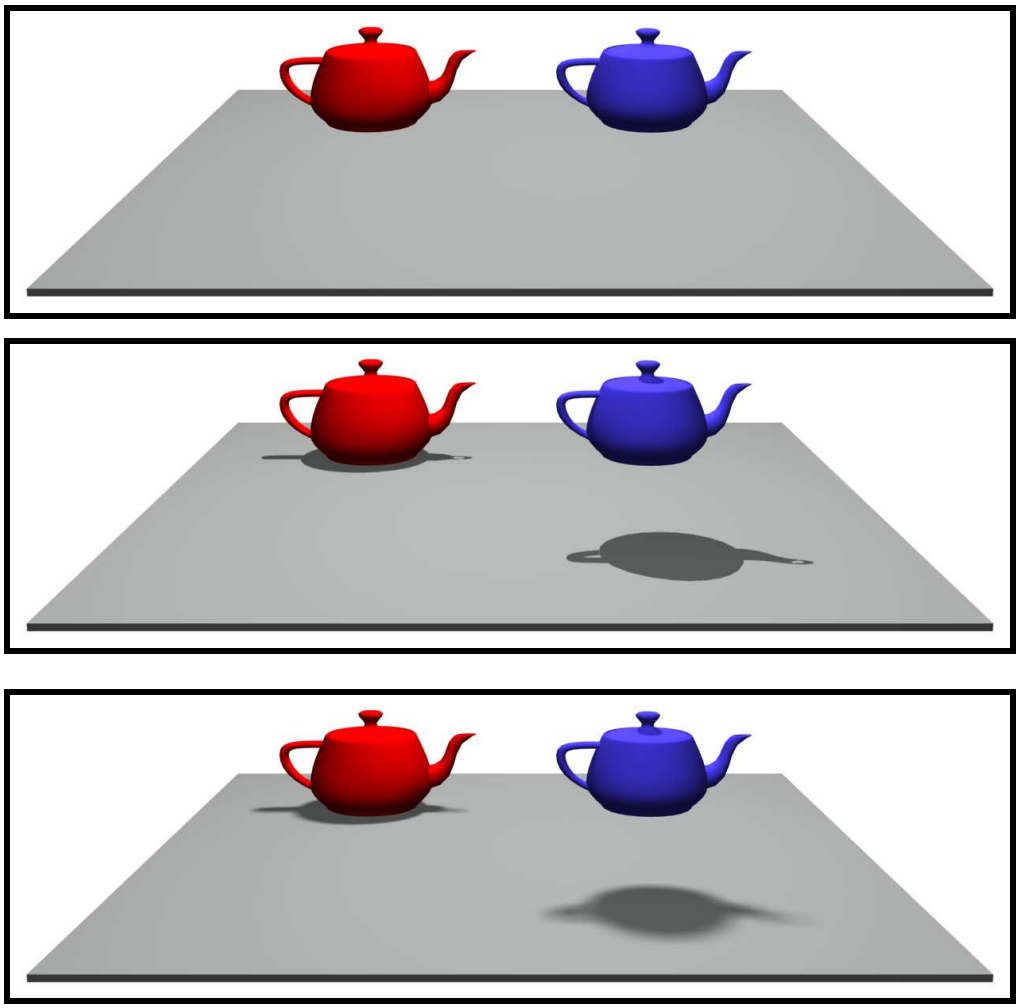


Figure 1.1: The Tale of Two Teapots

A simple scene with, from top to bottom, no shadows, shadows from a point light source (also known as "hard shadows"), and shadows from an area source (also known as "soft shadows"). Note the difficulty in assessing distance without the shadows.

Because shadows depend on so many different factors, shadow generation has been an important and difficult problem that has been addressed in many different ways. [Möller99] and [Woo90] together present a comprehensive review of the most popular methods, but the following sections will provide a brief historical look at shadow generation techniques. The approach we shall take

will be to divide the various techniques into two categories: hard shadow (point light source) techniques and soft shadow (area light source) techniques.

Hard Versus Soft Shadows

"Hard shadows" are those shadows that are generated from a point light source. Because a point light source is used, each sample point in the scene is characterized as either "lit" (meaning that a ray cast from the sample point to the light source is unobstructed), or "unlit" (meaning that a ray cast from the sample point to the light source is obstructed). There are no intermediate cases since the light is considered an infinitesimally small point (no area), and the end result is a "hard" shadow edge, an example of which is shown in Figure 1.2.

Figure 1.2: A Scene with Hard Shadows

The image is generated using a point light source assumption, causing a sharp edge. Note that jagged edges appear when only one point sample is used per pixel.

"Soft shadows" are shadows that are generated from light sources with finite area. In this case, each sample point in the scene sees some fraction of the light source. This fraction can vary in a continuous fashion, resulting in "soft" shadow edges, as depicted in Figure 1.3.

Figure 1.3: A Scene with Soft Shadows

This image is generated using an area light source. Note that the shadow consists of both an umbra and a penumbra and the intensity gradient at the shadow edge is continuous.

Hard Shadow Techniques

One of the first techniques to be used for shadow generation was described by Appel [Appel68] and Bouknight and Kelley [Bouknight70]. They presented methods for projecting polygon edges from the light's viewpoint onto polygons that were further away. Each polygon had a list of those polygons that

could potentially shadow it. This list was analyzed to reduce the number of polygon pairs by removing pairs that did not interact. The information in the list was then used when rendering each scan line.

[Sutherland74] used a two-pass approach. The first pass identified surfaces visible from the light and split these polygons into lit and unlit polygons if they were partially in shadow boundaries. Now, with all polygons either lit or unlit, a second pass displayed the new collection of surfaces using a hidden surface algorithm.

Another method introduced by Atherton et al. [Atherton78] used clipping techniques to divide polygons along shadow boundaries into lit and shadowed pieces. The lit portions were shaded and the shadowed portions were then added. This allowed camera movement without having to recompute the shadows.

Blinn [Blinn88] introduced the projection of occluders as a simple method for creating shadows on planar receivers. Despite its lack of generality, this technique has been widely used to cast shadows onto large polygons such as floors or walls of rooms because only a simple projection is involved.

One method of hard shadow generation that has become extremely popular is Crow's shadow volume technique [Crow77]. For this technique, the volume of space delimited by the light source is defined using polygons. However, these polygons are not displayed. Rather, they are used to identify whether scene polygons are inside or outside the light's influence. The shadow volume tech-

nique can be easily implemented using the stencil buffer in today's graphics hardware [Fuchs85] [Heidmann91], which allows a counter to be maintained for each pixel in the image plane [Woo99]. The counters are used to decide whether each pixel is inside a shadow volume of not, and hence to shade each pixel appropriately. The main disadvantages of shadow volumes are that implementations using graphics hardware tend to be fill-rate intensive, and that there are a number of boundary cases and other implementation details to deal with to achieve correct results.

Another hard shadow generation technique that is widely used today is the shadow buffer (also called the "shadow map," which is the term used in this thesis). This technique was introduced by Williams [Williams78] and is based on a two-pass approach. The basic idea is to generate a depth-buffered image from the light source and to use the information in the depth image to shade the eye view. Segal et al. [Segal92] showed how to implement shadow mapping using the texture mapping capabilities of high-end graphics hardware, which in turn led to the support of shadow mapping in commercial graphics hardware today. Because this thesis is very closely related to the shadow map technique, we shall return to the shadow map algorithm in the Background on Shadow Maps section.

Ray tracing, first conceived by Appel [Appel67] and put into practice by Whitted [Whitted80], is a technique that can be used to generate highly realistic images. Ray tracing, as the name implies, follows the path of rays from the eye through the scene. This procedure is done for each pixel in the image plane. After determining the intersected surface point seen through the pixel, a

shadow ray is then cast from that point to the light source to compute the lighting for each pixel. For point light sources, the result is a binary determination (because the ray will either be blocked or not) and thus, the shadows that are generated will have hard edges. Ray tracing is versatile and accurate, even for area light sources if sufficient point samples are taken, but has high computational costs. However, recent work by Parker et al. [Parker99] and Wald et al. [Wald01] has shown that carefully optimized ray tracers can be used to achieve interactivity for point light sources even on single processors.

Soft Shadow Techniques

Many hard shadow techniques can be applied to soft shadow generation by using multiple point samples on an area light source and accumulating them. The problem with these approaches is that they tend to be very expensive since they do not take advantage of any coherence to reduce the cost of the soft shadow generation. In the realm of graphics hardware, Haeberli and Akeley [Haeberli90] introduced the accumulation buffer, which could average multiple scene renderings. Thus, several hard shadow images created by any shadowing algorithm could be averaged to produce soft shadows. Brotman and Badler [Brotman84] combined the contributions of multiple shadow volume samples with the depth buffer to simulate area lights. Heckbert and Herf [Heckbert97] combined images from different samples on an area light source to create per-polygon textures. However, their approach is computationally inefficient since it can scale quadratically in the worst case with the number of self-shadowed polygons in the scene.

Nishita and Nakame [Nishita85] showed how to extend shadow volumes to area lights by computing samples either at both ends of a linear light source or at every vertex of a convex area light. Interpolation was then used to compute visibility for the intermediate locations.

Chen and Williams [Chen93] constructed shadow maps from several representative points on the light source (such as its vertices) and used interpolation to calculate the average visibility. The problem with this approach was that it required a lookup into each of the representative shadow maps, which can be costly.

Ray tracing approaches can be extended to take many samples on an area light source instead of using a single sample representing a point light source when casting shadow rays. The technique of distributed ray tracing, introduced by Cook [Cook84] allows this approach, and results in soft shadows.

Hart et al. [Hart99] presented a view-dependent technique that accelerated shadow computation by using coherence in blockers, which are the surfaces that prevent light from reaching other surfaces. For each pixel in the image plane, a set of blockers was computed. Neighboring pixels were then checked for the same blocker recursively. The main problem with this approach was that unless samples were taken with sufficient frequency, small blockers could be missed, resulting in inaccurate shadow computations.

Keating and Max [Keating99] used multi-layered depth images as a basis for their shadow computations. They discretized the recorded depths to reduce

"light leaking" and used a depth-based percentage-closer filtering [Reeves87] operation on the depth images (see Figure 1.4) to produce blurred shadows.

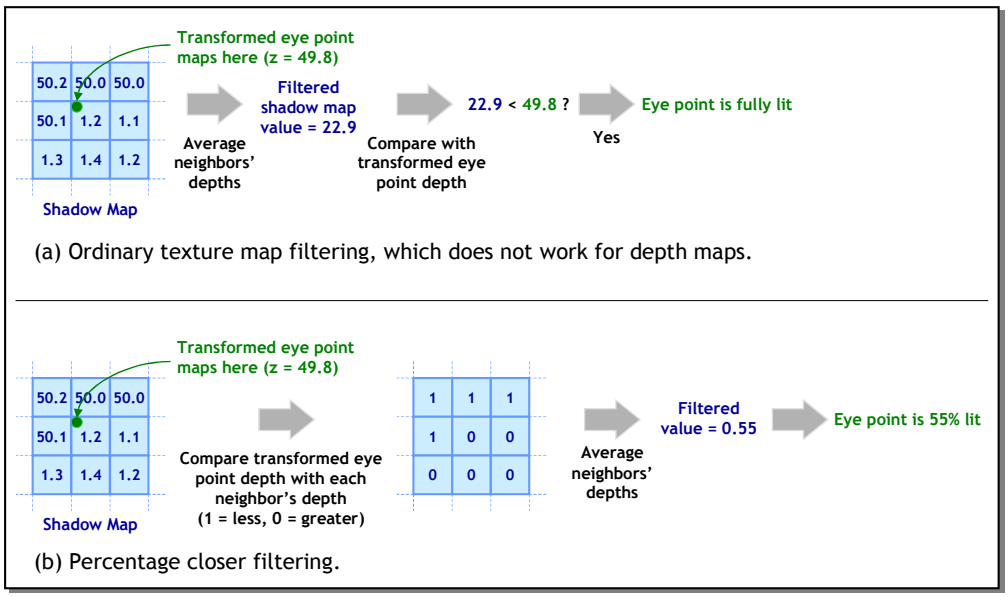


Figure 1.4: Percentage Closer Filtering

This diagram shows how percentage closer filtering works. Here we are examining the query process for a particular eye view sample point, which maps into the shadow map at the texel with depth 1.2. The eight neighbors of this texel are shown, since they are to be used for filtering. The sample point being tested has depth equal to 49.8. As shown in part (a), ordinary filtering would simply average the surrounding depth values and produce a simple binary result. Part (b) shows percentage closer filtering, which tests the sample point's depth value individually against each neighboring texel of the shadow map. The fraction of successful tests determines the sample point's final attenuation.

Agrawala et al. [Agrawala00] presented two image-based techniques for computing soft shadows. The first, layered attenuation maps, warped and combined depth information from a number of samples on the light source into a layered depth image [Shade98], which is essentially an image that stores, at each pixel, the surfaces that exist at various depths. The resulting attenuations were stored in the layered depth image and queried during shading. Al-

though shading could be accomplished at several frames per second on an SGI Onyx2 InfiniteReality, the pre-processing required prevented this technique from supporting dynamic scenes. The second technique introduced by Agrawala et al. was called coherence-based ray tracing. Similar to the work of Hart et al. [Hart99], this approach took advantage of the fact that visibility changes occur slowly in neighboring regions. The end result was a significant reduction in the time taken to generate ray-traced images with soft shadows, though the technique is not interactive.

The radiosity technique, introduced by Goral et al. [Goral84], models the interactions between diffuse surfaces by dividing the polygons of the scene into patches and computing the radiosity at each patch. Since the approach accounts for visibility as part of its form factor calculation, the computed solution will contain shadow information. However, due to the large number of patches, the visibility calculations are too expensive to allow interactivity, and the level of meshing required to produce accurate shadows is often prohibitive. Discontinuity meshing [Lischinski92] can be used to improve the accuracy of the meshing process by finding discontinuity events such as umbra and penumbra boundaries and then subdividing around those region boundaries.

Soler and Sillion [Soler98] introduced a technique that used convolution to perform filtering on blocker images to produce approximate soft shadows. The convolution produces superior image quality when compared to averaging hard shadows, but the approach relies on clustering objects that do not

shadow themselves. The clustering operation becomes difficult for complex scenes that could potentially have a large amount of self-shadowing.

BACKGROUND ON SHADOW MAPS

Introduced by Williams in 1978 [Williams78], conventional shadow mapping is a common technique used to generate shadows. Since it is an image-space technique, it confers advantages in terms of generality, speed, and ease of implementation. A shadow map is a depth image generated from the light source view. When rendering, points in the eye view are transformed into the light source view. The depths of the transformed points are then compared with the depths in the shadow map to determine if the transformed points are visible from the light source. A transformed point is considered lit if it is closer to the light than the corresponding point in the shadow map. Otherwise, the point is considered to be in shadow. This information is then used to shade the eye view image. Figure 1.5 illustrates an example of the process we have just described. In this case, we want to shade the red pixel on the image plane (which has been exaggerated in size). This pixel corresponds to a point on the green box, as shown in the figure with a green circle. The eye view and light view boxes illustrate the corresponding locations of this point on the image plane and shadow map, respectively. In this case, since the point is visible in the light view, the eye view pixel is not darkened. If the point were not visible in the light view, the eye view pixel would be appropriately attenuated to show that it is in shadow.

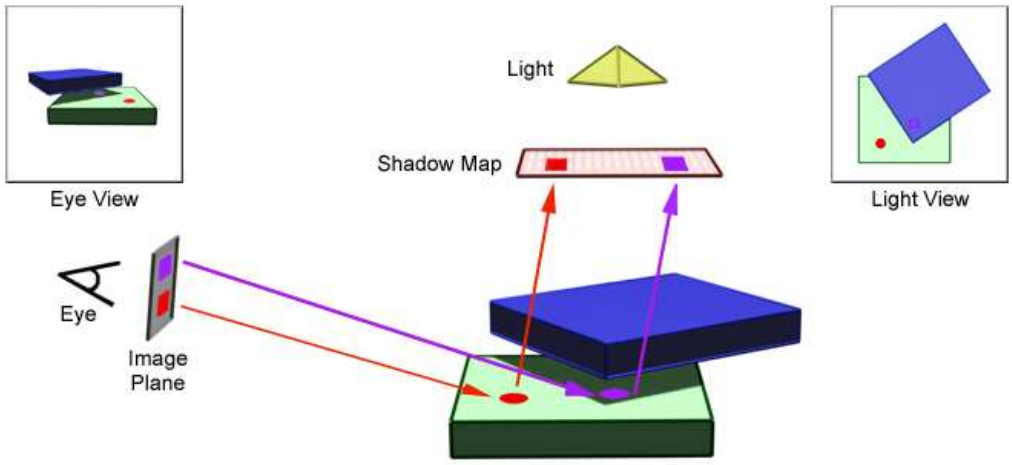


Figure 1.5: How Shadow Maps Work

This example shows two sample points being transformed and shaded. The red point corresponds to a lit region on the green box, while the purple point is in shadow. When the points are transformed into the light view for comparison with the depths in the shadow map, the red point has the same depth as the surrounding pixels from the green box. Hence, it is deemed "lit". The purple point, however, is found to have a depth greater than the blue box, which means that the blue box occludes it. Therefore, the purple point is deemed to be "in shadow" and is darkened appropriately during shading.

Advantages and Disadvantages of Shadow Maps

Shadow maps have a number of useful characteristics. Since they are an image-space technique, they tend not to be affected by the number of polygons in the scene. They can be modified to easily deal with different types of primitives. In addition, they are easy to implement and can deliver relatively high frame rates because of their simplicity.

Unfortunately, shadow maps do have some significant drawbacks. The two principal problems are aliasing (blocky shadow edges that result from insufficient shadow map resolution) and bias determination (finding a bias factor to

add to the depth comparison to prevent erroneous self-shadowing). This thesis does not address the bias determination problem, although a discussion of some of the issues can be found in Williams' original work [Williams78].

Chapter 2 explains how aliasing arises in shadow maps, followed by a discussion of the subsequent difficulties that arise. The chapter also introduces the Adaptive Shadow Map (ASM), which is a solution to the aliasing problem in shadow maps.

Another problem with shadow maps is that they are inherently a hard shadow technique. It is possible to use an approach of taking many point samples over an area light source and combining the results to simulate an area source. However, such an approach can be slow. In Chapter 3, we introduce an adaptive technique, Adaptive Soft Shadows, which helps to make the soft shadow computation more efficient. This approach is not constrained exclusively to shadow maps, but we have chosen to use shadow maps as the basis for our implementation because of their versatility and because they are supported in commercial graphics hardware.

Chapter 2

Adaptive Shadow Maps

In this chapter we shall first explain the aliasing problem in shadow maps and describe its repercussions. Next, we introduce the Adaptive Shadow Map and explain the theory behind it. Following this is a description of the implementation we used along with the results we obtained. The chapter concludes with some of the future work that could be done to improve the algorithm we describe.

How Aliasing Arises in Shadow Maps

As mentioned in Chapter 1, one of the major drawbacks of shadow maps is aliasing. *Aliasing* refers to artifacts caused by insufficient sampling. When dealing with hard shadows, aliasing is particularly noticeable and produces jagged edges because of the binary and therefore discontinuous nature of the shadow map. This map, generated from the light's view, is actually a point-sampled representation of what can be seen from the light, with each point sample being represented by a pixel in the shadow map. As we transform each pixel in the eye view into the shadow map for comparison and shading, the resulting footprint of the pixel can be larger or smaller than the pixels in the shadow map. If the footprint is larger, we could average the values of sev-

eral shadow map pixels (as in percentage-closer filtering) to ascertain an appropriate value for the eye view pixel. However, if the footprint in the shadow map is smaller than the pixels in the shadow map, multiple eye view pixels will map to the same shadow map pixel, resulting in the same value for all of these pixels. *Thus, aliasing arises when the sampling resolution of the shadow map is less than the resolution of the eye view*. According to the Nyquist theorem [Nyquist28], the shadow map resolution must be at least twice that of the eye view resolution to prevent aliasing.

Figure 2.1 illustrates how aliasing arises in shadow maps. In the figure, we see views from two different locations in a simple scene: Viewpoint A and Viewpoint B. The grid in each picture shows the projected pixel area of the shadow map as seen by the current viewpoint. Since the projected area of the light source pixels in Viewpoint A is roughly equal to the area of the shadow map pixels in the shadow map, aliasing is minimal in this case. In contrast, Viewpoint B is quite close to the scene geometry, resulting in large projected areas for the shadow map pixels in this view. This means that large areas of the image generated from Viewpoint B are shaded using relatively small amounts of information from the shadow map. The result is significant aliasing artifacts, depicted here as jagged edges.

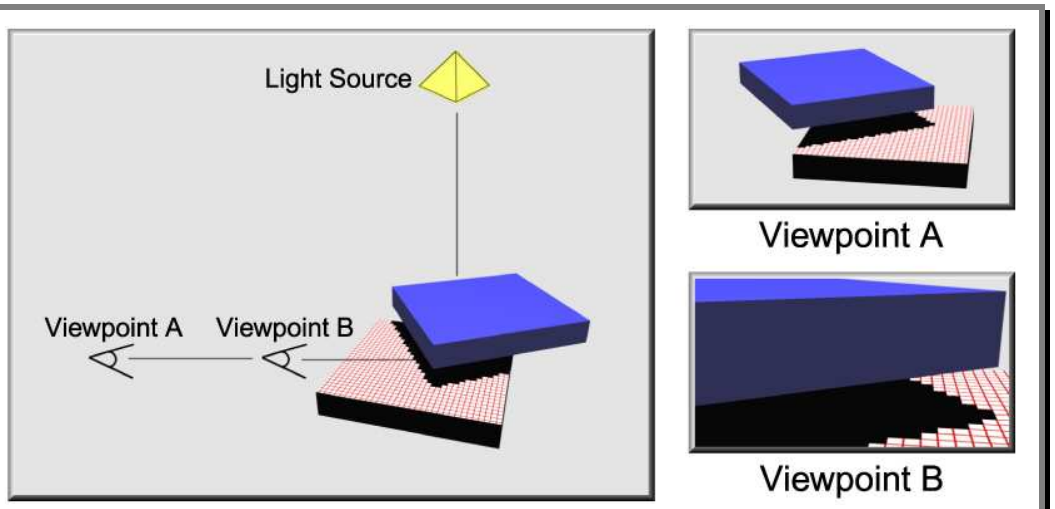


Figure 2.1: Aliasing in Shadow Maps

A simple scene is viewed from two viewpoints. While aliasing is negligible at Viewpoint A, it causes large jagged edges at Viewpoint B.

The Implications of Aliasing in Shadow Maps

The aliasing problem has many inconvenient repercussions. When using shadow maps in an interactive application, light positions and shadow map resolutions have to be chosen carefully so that aliasing is minimized in all potential camera views. The common solution is to place lights close to the objects that will be viewed and to use high-resolution shadow maps. However, this does not work in many situations. One such problematic scenario occurs when a single light source illuminates a room. In this case, the light might be much farther from the objects in the scene than the camera will be, meaning that to prevent aliasing, the required shadow map resolution could be very high. Since shadow map memory usage is proportional to its total resolution (e.g., a 1024 by 1024 pixel shadow map would use 4MB of memory, assuming a 32-bit value is used to store the depth at each pixel), the memory required can become prohibitive.

PREVIOUS WORK

Removing aliasing in shadow maps has been a difficult problem to resolve. Percentage-closer filtering, described in Chapter 1, is one solution to the aliasing problem (see Figure 1.4). With this approach, each lookup into the shadow map is filtered using the surrounding region of the shadow map, and the fraction of the samples that is shadowed returned. Percentage-closer filtering incorporates more samples into the visibility calculation and allows lookups into the shadow map to have results that are not just binary. The visual result is softer, antialiased shadow boundaries. Although this approach is very useful and easy to implement, it does not address the problem of inadequate shadow map resolution as the fundamental cause of aliasing. Therefore, in cases where the shadow map resolution is significantly lower than necessary, percentage-closer filtering only masks aliasing by blurring.

Figure 2.2 illustrates the light buffer [Haines86], which resolves the aliasing problem within the context of a non-interactive ray tracer by using a flat shadow map with analytical testing of shadows. The problem with this approach is the lack of interactivity because of its dependence on expensive ray casting operations.

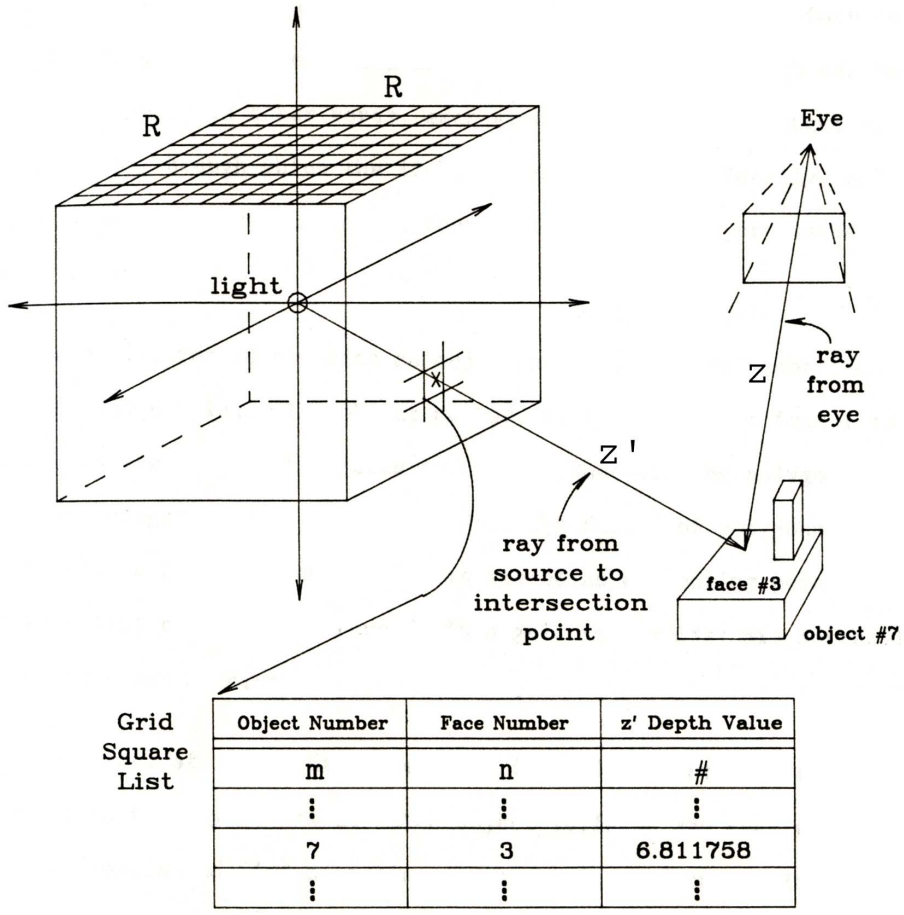


Figure 2.2: Schematic Representation of the Light Buffer

A cube-like structure is constructed around the light source. Each face of the cube is divided into cells, and each cell contains a list of the surfaces potentially visible through the cell. A scan-line algorithm is used to take advantage of this information and to thereby accelerate ray tracing.

Most recently, Deep Shadow Maps [Lokovic00] address aliasing by using jittered samples and pre-filtering them. Like percentage-closer filtering, Deep Shadow Maps are able to improve shadow quality by using more samples for each lookup. Again, Deep Shadow Maps also do not deal with aliasing caused by insufficient shadow map resolution.

A COMPARISON

To address the aliasing problem described above, we introduce the Adaptive Shadow Map (ASM). The basic idea behind the ASM is to use a hierarchical structure to store the shadow map instead of the conventional flat, regular structure. As we will see, the new approach gives us many advantages. Before going on to describe ASMs in detail, let us first look at an illustration of the

Figure 2.3: A Comparison

On the left is an image generated by a 2048 by 2048-pixel conventional shadow map using 16 MB of memory. On the right is an image of the same scene using an ASM and an equivalent 16 MB of memory. Note the vastly improved image quality on the right.

ASM's utility. Figure 2.3 presents a comparison between conventional 2048 by 2048 pixel shadow map and an ASM. In this example, the "bunny" is just one object in a large room that is lit by a single light. With both algorithms using 16 MB of memory, we can see that the ASM has refined the shadow map for the camera position, resulting in higher shadow quality. On the other hand,

the conventional shadow map exhibits aliasing artifacts because of its inadequate resolution for the current viewpoint.

In practice, lights are often placed artificially close to objects and camera paths are carefully constructed to ensure that aliasing does not arise. This involves considerable tweaking with respect to picking the appropriate light position and resolution. Our goal with ASMs was to address the aliasing problem in shadow maps while minimizing user intervention and maintaining an interactive frame rate.

DESCRIPTION

ASMs are based on the observation that a high-quality shadow map need not be of uniform high resolution; only regions that contain shadow boundaries need to be sampled densely. In those regions, the resolution of the shadow map should be at least as high as the corresponding region in the eye view to avoid aliasing artifacts. Figure 2.4 illustrates the intuition behind the previous statements. A simple scene consisting of a table on a plane is shown. There are four cases that can arise. In fully lit and fully occluded regions, there is no detail in the shadow to refine. These regions have no high intensity gradients and thus are visually unimportant with respect to shadowing. However, since the eye detects edges, shadow boundaries are of great interest and could potentially require refinement to improve their shadow quality. But we do not want to refine areas that are already of adequate resolution and do not perceptually reveal aliasing artifacts. This means that we do not want to refine the distant shadow boundary at the back of the table, though we do want to refine the shadow boundary that is close to the eye, near the front of the table.

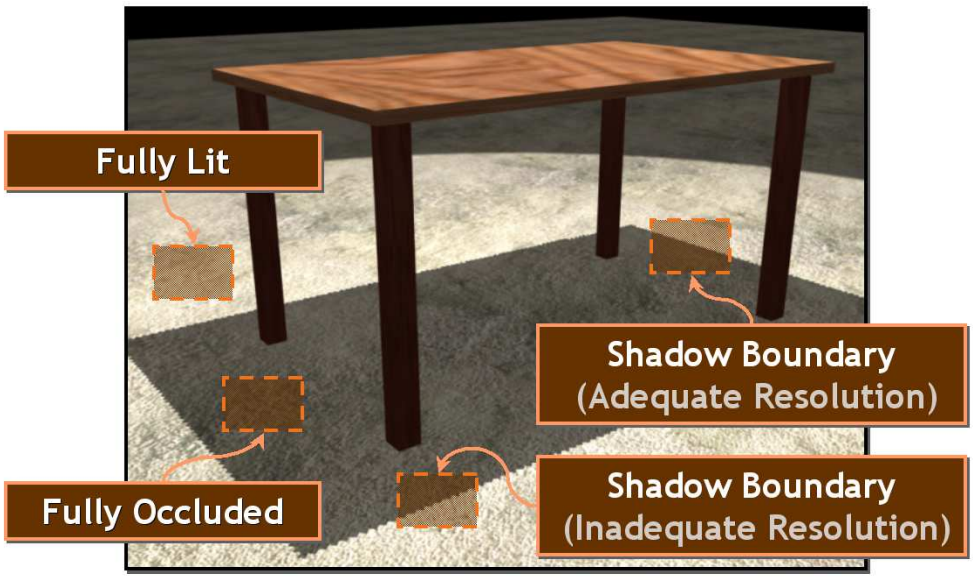


Figure 2.4: Deciding Where to Refine

The four classifications for regions in the image are shown. To improve image quality, the algorithm needs only to focus its effort on shadow boundary regions of inadequate resolution.

An ASM hierarchically subdivides an ordinary shadow map, providing higher resolutions in visually important regions. Like a conventional shadow map, an ASM takes as input a set of transformed eye view points and allows shadow queries on this set, returning true if a particular point is lit and false otherwise. In software systems, ASMs can seamlessly replace conventional shadow maps.

An ASM has three main characteristics:

- It is view-driven, meaning that the hierarchical grid structure is updated based on the user's viewpoint.
- It is confined to a user-specifiable memory limit. Memory is managed efficiently and the ASM avoids the explosive growth in memory usage

that would be required by a conventional shadow map of the same visual quality.

- It is progressive, meaning that once a particular viewpoint is established, image quality continues to improve until the user-prescribed memory limit is reached.

PROCESS FLOW

Figure 2.5 shows the ASM's process flow. ASMs interface with applications in the same way as conventional shadow maps. That is, they take as input the set of pixels in the eye view (the *Transform Eye Pixels* stage) and they allow a query for each pixel that returns a binary value indicating whether that pixel is lit or in shadow (the *Query* stage). However, ASMs do perform some additional processing, which can be divided into two stages that we call the *Analyze* and *Refine* stages. The *Analyze* stage identifies potential portions of the shadow map that need refinement, and evaluates a cost-benefit metric for these portions. Next, based on the information generated by the *Analyze* stage, the *Refine* stage updates the data structure. The most valuable new portions are generated, and if the memory limit has been reached, the least valuable shadow map portions are removed.

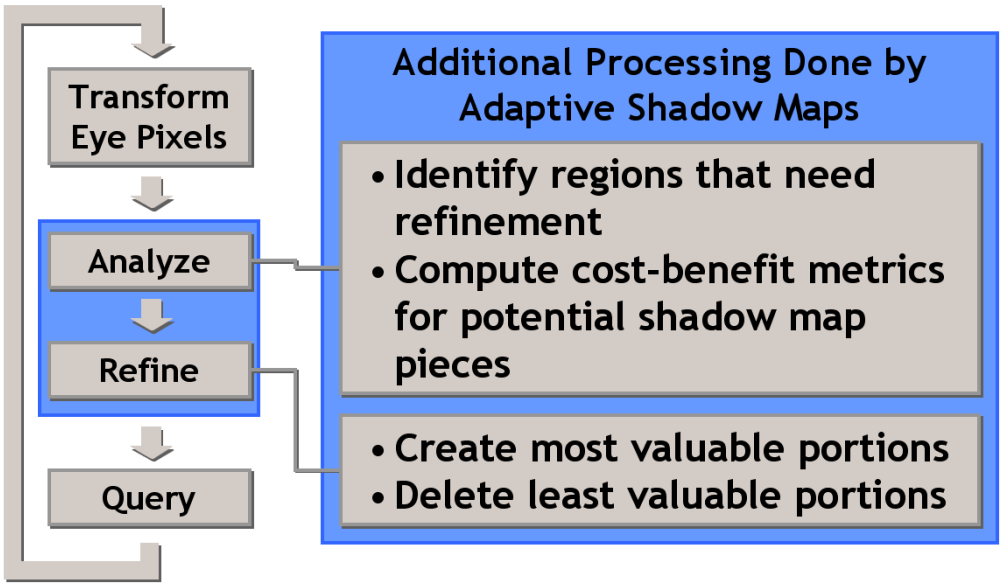


Figure 2.5: The ASM Process Flow

The additional processing performed by the ASM is shown.

DATA STRUCTURE

Organization

The data structure we use is a simple quad-tree structure. Each node in an ASM tree has a shadow map of a fixed resolution and a partitioning of that shadow map into a fixed number of cells. Each of these cells may contain another tree node. Figure 2.6 illustrates ASM data structure for a particular viewpoint. In this case, the yellow squares represent regions of the shadow map that are close to the eye and which contain discontinuities. Because of this, they have been created at a high resolution. In contrast, the more distant regions are in blue, indicating that they are of low resolution. The algorithm

has deemed that subdividing these regions would not increase the eye view image quality sufficiently, given the current memory constraints.

The number of children that a node contains can be varied to tune the performance characteristics of the algorithm. For example, if a small number of children is used, the tree depth will be correspondingly higher, but less checks have to be made at each level of the tree. Conversely, a large number of children can be used, resulting in a lower tree depth, but requiring more checks to be made at each level. For computational efficiency, the parameters should be tuned based on the needs of the intended application.

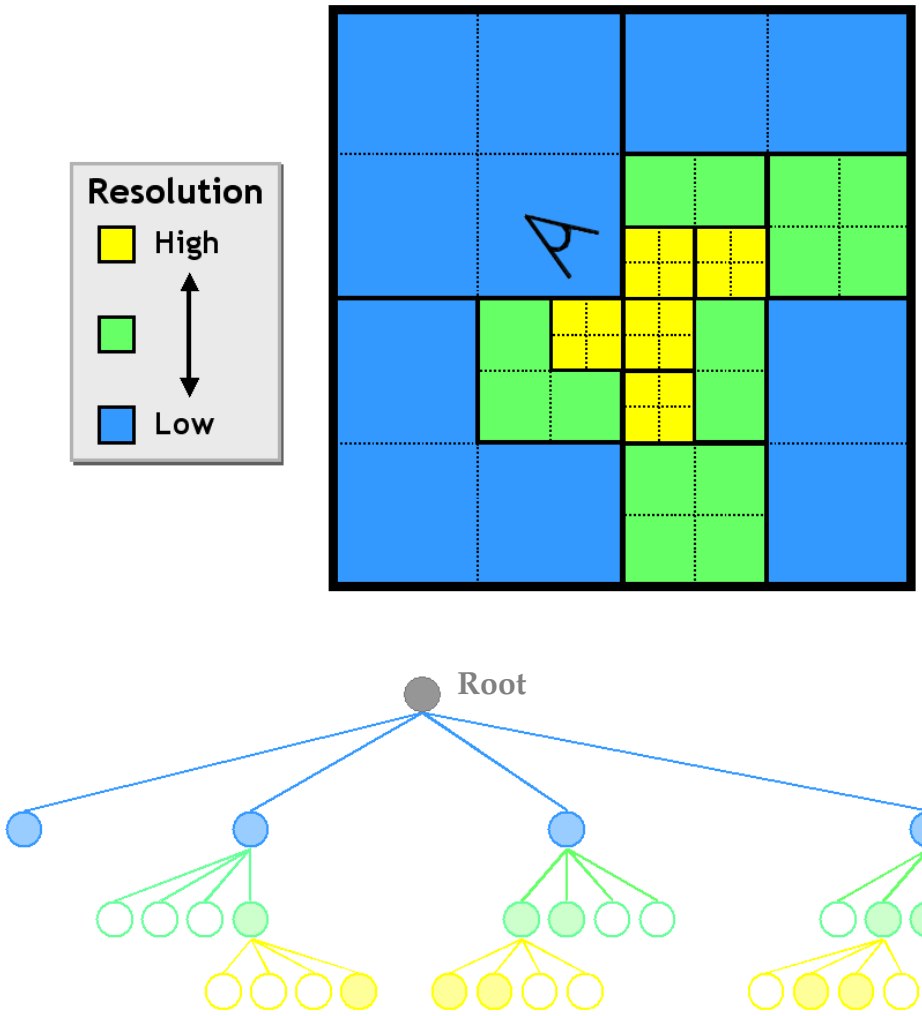


Figure 2.6: The ASM Tree Structure

The top diagram shows the ASM shadow map structure. The yellow squares represent regions that are close to the eye and that contain shadow discontinuities. The bottom diagram shows the same situation, but represented as a tree.

Two operations may be performed on a cell in the tree. An empty cell may have a new node assigned to it when it is determined that the resolution of the shadow map region corresponding to the cell is not high enough to provide the required image quality. A cell containing a node may also have that node

and its descendants deleted. This is done in response to the user-specified restrictions on memory usage.

Creating Nodes

At any one time there are many cells that require new nodes to be assigned to them. In an interactive application, it is not always possible to fulfill all of these requirements. Therefore, we use a cost-benefit metric to determine which cells to satisfy. It is only beneficial to create a new node (and hence a higher resolution shadow map piece) if it causes a perceived improvement in shadow quality. We quantify this perceived benefit by counting the number of transformed eye pixels within the cell that straddle a depth discontinuity *and* whose shadow map resolution is lower than the eye view resolution. We use the mip-mapping capability of commodity graphics hardware to estimate the resolution in the eye view and in the shadow map. The Mip-mapping section below explains this in detail.

The cost of creating a new node is the amount of time required to generate its new shadow map. Using the ratio of eye view pixel size to shadow map pixel size, the resolution required for the new shadow map to match the eye view resolution is:

$$N_{required} = N_{current} \frac{Eye\ View\ Pixel\ Projected\ Area}{Shadow\ Map\ Pixel\ Projected\ Area}$$

where N is the number of pixels in a shadow map cell. To understand the origin of this equation, consider Figure 2.7, which shows a simple case where an area in world space is being projected into both the eye view and shadow map. The area being projected is shown in bright green, on the blue box. We want to compare the projection of this area in the eye view and shadow map.

The projected areas are shown in green. We can see that in this case, the projected area in the eye is larger than the projected area in the shadow map. This means that there is insufficient information to shade the eye view correctly, and therefore, that the shadow map quality needs to be improved.

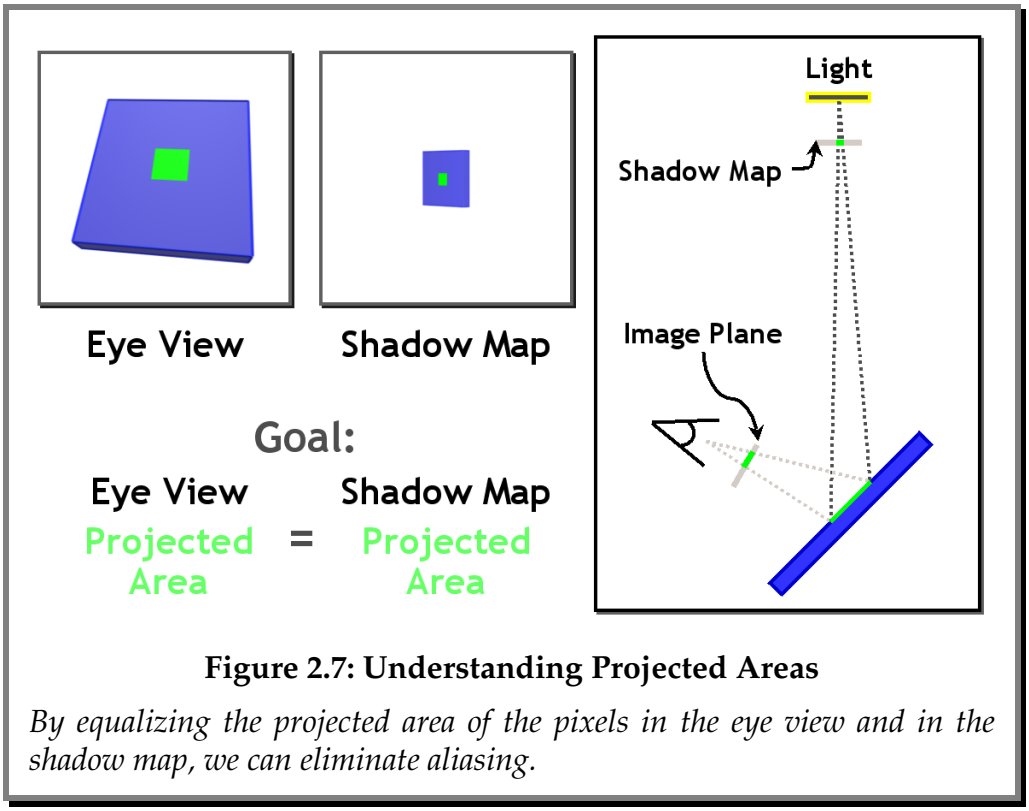


Figure 2.7: Understanding Projected Areas

By equalizing the projected area of the pixels in the eye view and in the shadow map, we can eliminate aliasing.

The cost of generating a new shadow map can be approximated by:

$$\cos t = aN_{required} + b$$

This cost model is based on the fact that hardware read-back performance varies roughly linearly with the size of the buffer being read back, becoming increasingly inefficient as the read-back size gets very small. We perform a calibration test as a preprocess to evaluate *a* (the per-pixel rendering cost) and *b* (the constant overhead) for a given hardware setup (more information about this appears in the Implementation Details section).

The benefit of a new shadow map is the number of resolution-mismatched eye pixels that could be resolved. Once the cost-benefit ratio is computed for all prospective cells, the cells are sorted according to this ratio. To maintain consistent frame rates, new shadow map nodes are only generated for the most profitable cells from this list until a given time limit has passed.

Removing Nodes

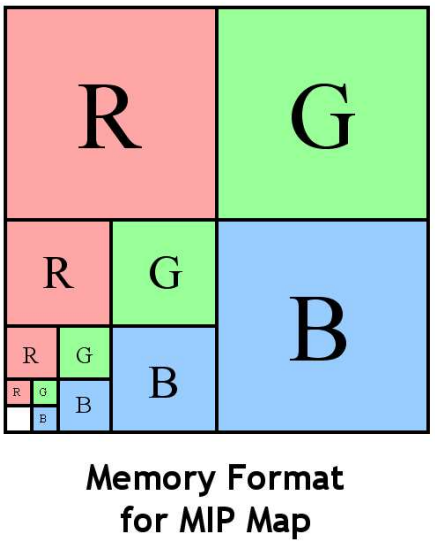
Since the ASM only uses a fixed amount of memory, a particular node's memory might need to be reclaimed. In order to do so, we find all nodes that were not used in the last frame and remove the least recently used nodes. If there are no such nodes, then all nodes are currently visible. In this case, we remove an existing node only if a new node that needs to be created has a greater benefit than the existing node. We require that all descendants of a node be deleted before the node itself is deleted. In this way, we ensure that the tree structure builds up gradually and is deleted gradually as well. One disadvantage of using this scheme is that memory resources can be consumed simply because of one small region that needs high magnification, which would necessitate building up a deep tree to achieve the required magnification. In addition, a tree with such depth would diminish the ASM’s query performance since many pointers would have to be followed in order to reach a leaf node.

IMPLEMENTATION DETAILS

Mip-mapping

To determine when resolution mismatches occur, the algorithm must calculate the projected area of a pixel (i.e., the area that the pixel covers in world-space). Performing this calculation in software would be too expensive for interactive rates, so we approximate this calculation using mip-mapping.

Mip-mapping [Williams83] is traditionally used to avoid aliasing artifacts associated with texture-mapping (Figure 2.8). Current graphics hardware implements perspectively-correct mip-mapping, which interpolates between textures of different resolutions based on the projected area of the pixels being rendered. The basic idea is that if we assign a uniform texel size in world space to each polygon in our scene, we can then use mip-mapping to compute the projected size in the eye view or in the shadow map.



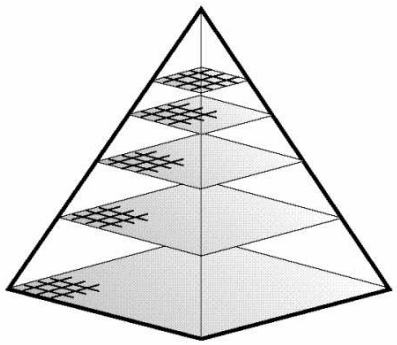


Figure 2.8: Mip-mapping

A mip-map stores versions of the original texture at different resolutions. Starting with the original texture, each level is minified using a filter to create the next, smaller level. When texturing with mip-mapping enabled, linear interpolation is used to mix lookups from the most relevant two levels of the mip-map.

We use this feature to quickly approximate the projected pixel area. The algorithm places the resolution of each mip-map level in all the texels of that level. Texture coordinates are set up so that world-space texel sizes are uniform, and every polygon is then drawn with this mip-mapped texture. When the frame is read back, each pixel contains its trilinearly interpolated mip-map level, which is a reasonable approximation of its projected area. Anisotropic filtering, which helps to account for oblique texels, is used to improve the accuracy of the approximation.

As a further optimization, the mip-map level is encoded only in the alpha channel, which is an extra channel in the frame buffer that is normally used for transparency. The rest of the mip-map is white. This allows the read-back

to be done simultaneously with the polygon identifier (ID) read-backs described below, eliminating an extra rendering pass.

Combining ID and Depth Comparisons

Conventional shadow maps commonly use a depth comparison to check if transformed pixels are lit, with a bias factor. The bias factor is necessary because of limited depth buffer precision, which causes depth values to be quantized. Quantization can cause adjacent pixels of the same surface to have markedly different depth values, resulting in erroneous self-shadowing. Although adding a bias factor helps to address this problem, the approach still exhibits artifacts on surfaces oblique to the light and has difficulties in scenes with varying geometric scale. Using per-polygon identifiers (IDs) to determine visibility instead of depth comparisons was proposed in [Hourcade85]. This approach is better in many cases but results in artifacts along mesh boundaries. Therefore, we use a combination of per-polygon IDs and depth comparisons to perform the visibility determination for transformed pixels. If the ID test fails, the conventional depth comparison allows us to avoid artifacts along polygon boundaries. This simple modification is more robust than using just per-polygon IDs or depth comparisons, although the bias problem persists.

Hardware Calibration

Because ASMs rely heavily on the graphics board to quickly render and read-back sections of the shadow map, a calibration process should be performed at initialization to determine the current hardware's characteristics. In particular, the geometry rendering rate and read-back times for varying viewport sizes

should be measured. This information can then be used in the cost-benefit equation of the ASM.

One of the key enabling technologies for ASMs has been the performance of commercial graphics boards. Rendering performance has improved exponentially (and somewhat predictably) over the past years, but the real surprise has been the improvement in read-back performance. Fast read-backs allow algorithms that use the graphics board to generate information and the CPU to quickly obtain and examine this information. In the case of commercial hardware, the improvement can be attributed largely to improved drivers. Early in 2000, a 512 by 512 pixel read-back took more than 900 milliseconds on a 733-MHz Pentium III with AGP 4X, an NVIDIA GeForce board, and early versions of the NVIDIA drivers. Over the next few months, this number decreased with each new driver release from 900 ms to 30 ms to 15 ms and even further. With these improved driver releases, the key factor that determines read-back performance is CPU speed. On a 733-MHz Pentium III with AGP 4x, an NVIDIA GeForce3 board, and NVIDIA's 21.81 Detonator drivers, it takes only about 6 ms to read-back the same 512 by 512 pixel region. In general, read-backs are an expensive operation because they stall the graphics pipeline and because they are relatively slow compared to other operations that can be executed on the graphics board. Because of this, we hope to move more of the ASM computations onto the graphics board in the future, as described in the Future Work section at the end of this chapter.

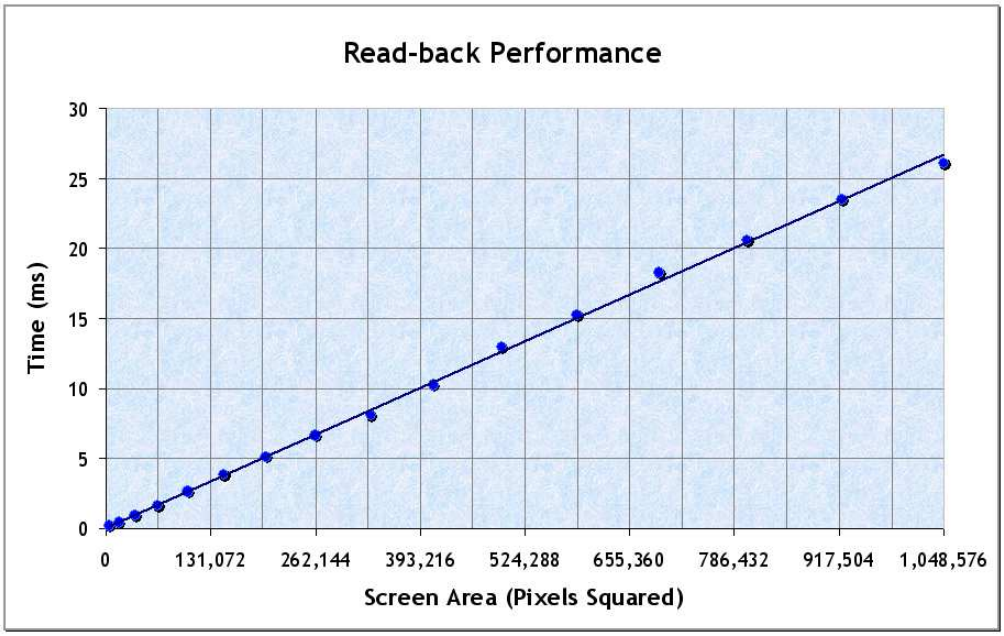


Figure 2.9: Read-back Performance

Timings for read-backs of various sizes are shown. The fitted line illustrates the linear behavior of read-backs as the area being read back increases.

The other key observation with respect to read-back performance is its linearity. Figure 2.9 shows the time taken for read-backs of various sizes. We can see that there is almost no loss in linearity as we reduce the read-back size until we reach a very small size of around 32 by 32 pixels. This means that the approach used in ASMs of rapidly creating many small shadow map pieces is indeed viable.

Optimizations

Goals

We will now describe some optimizations that were made to improve the efficiency of the ASM algorithm. Since the algorithm is dependent on the CPU for transforming points and analysis, and on the graphics board for rendering geometry, we tried to introduce specific techniques that would address each of these stages.

Min-Max Depth Culling

It is possible to accelerate ASM queries using a depth culling technique similar to that described in [Greene93]. The basic idea is to take advantage of the ASM data structure by storing the minimum and maximum depths found in each cell of the ASM (including its descendants). Now, when a sample point needs to be queried, the transformed depth of the point can easily be compared with the minimum and maximum depths stored at each cell as the hierarchy is traversed. If the sample point's depth is less than the minimum depth or greater than the maximum depth stored in the cell, there is no need to traverse further in the hierarchy. In addition, the conversion from shadow map to cell coordinates can also be avoided. The end result is a speedup, particularly in fully shadowed regions.

Caching of Most Recently Queried Node

Because the tree traversal process is expensive (many pointers need to be followed, and coordinate conversions performed), we use a cache to store the most recently queried ASM leaf node. As we are transforming pixels in the

eye view, it is likely that there will be a great deal of coherence with respect to which ASM cell the pixels map to. A multiple level cache could be used to further improve performance.

Low-level Optimizations

Low-level optimizations such as using Pentium family SSE/SSE2 instructions could be used to speed up pixel reprojection from the eye view to the shadow map. One of the difficulties with implementing these optimizations is that they require an inspection of the algorithm from a very data-oriented point of view. When initially formulating an algorithm, it is natural to think in terms of process flow instead of data flow. However, once algorithmic efficiency is achieved in performance-critical applications, exploiting the remaining low-level optimizations and parallelism can require significant restructuring. In the case of ASMs, a good way to proceed with making these optimizations would be to transform points four at a time. This would allow the use of SSE optimizations, which typically perform one operation simultaneously on four floats. Since projections are done in software, there is a great deal of performance that can be extracted by using known and efficient optimizations for the matrix multiplies, divisions, and rounding operations that are involved. In addition, the pixel reprojection phase fits well into future processor architectures, which further encourage explicit parallelism.

Frustum Culling

Our approach requires frequent renderings and read-backs as cells are refined. As the size of the shadow maps associated with the various cells becomes smaller and smaller, the scene drawing time becomes increasingly dominant.

To alleviate this problem, we use frustum culling for each cell in the topmost level of the hierarchy. This allows us to redraw only the geometry that is visible in a particular cell as we refine the cell. The implemented version of the ASMs uses eight by eight cells at the top level for frustum culling. This was a simple approach that was easy to implement. If necessary, a more complex frustum culling approach could be used. Finally, it is important to note that having a very large fraction of the overall scene geometry in one cell would hurt the ASM's performance and make its performance more dependent on scene complexity than a conventional shadow map.

Rendering Optimizations

Related to frustum culling is the need to draw the scene geometry as fast as possible. To do this, the OpenGL rendering commands have to be coded carefully. Basic rules such as grouping of similarly textured geometry need to be used. One of the most significant speed-ups comes from the use of the vertex array range extension, which allows the storage of vertex arrays on the graphics board itself. Traditionally, display lists were the fastest way to store and draw data, but they reside in main memory. This means that precious time is spent shipping data across the AGP bus when using display lists. The vertex array range extension requires no transfer of data, and the result is vastly higher performance. One slightly annoying caveat, however, is that for textured objects, the objects (and hence the data in the vertex arrays) must be grouped by texture. Two solutions exist to this problem. First, one can use a separate vertex array for the objects of each texture. The alternative is to use a single large vertex array that holds all the relevant data, grouped by texture. In this case, the array needs to be padded because the starting point of a vertex

array range must be 32 byte aligned. Either option can be used without any problem.

Sampling

Since analysis of all pixels in the image can be expensive, our algorithm performs the cost-benefit analysis only on a fraction of the transformed pixels. This choice trades frame rate for slower convergence to an accurate solution. In our implementation, we found that analyzing one-eighth of the pixels gives good performance without significantly affecting the rate of convergence. The choice of what ratio of pixels to use when sampling depends on the target frame rate. For example, at a low eye view resolution (e.g., 256 x 256 pixels), the frame rate will be very high, and thus the fraction of pixels that is sampled can be reduced. The reason for this is that in a given amount of time, many frames will be drawn and the user would be less likely to notice any objectionable artifacts. However, at a high eye view resolution with a correspondingly low frame rate, the fraction of pixels sampled has to be relatively higher, because the user is more likely to notice the changes with each frame. One way to control this trade-off would be to allow the user to indirectly set the sampling ratio by interactively choosing the desired frame rate.

RESULTS

Description of Test Scenario

To demonstrate the ASM at work, we used a 31,000-polygon scene and an image resolution of 512×512 pixels. Figure 2.10 shows the scene as viewed from the light, with the key objects labeled. Figure 2.11 shows the same scene

viewed from the eye. The scene features three different objects designed to test different aspects of our algorithm. The light source is a point light with a 122-degree field of view. It is placed on the room ceiling, far from the objects. The first object is a 20,000-polygon “bunny model”, which illustrates the algorithm's ability to deal with small triangles and frequent variations in polygon ID. The other two objects are a robot and a sculpture with a fine mesh, which demonstrate the algorithm's ability to find and refine intricate shadow details of varying scale.

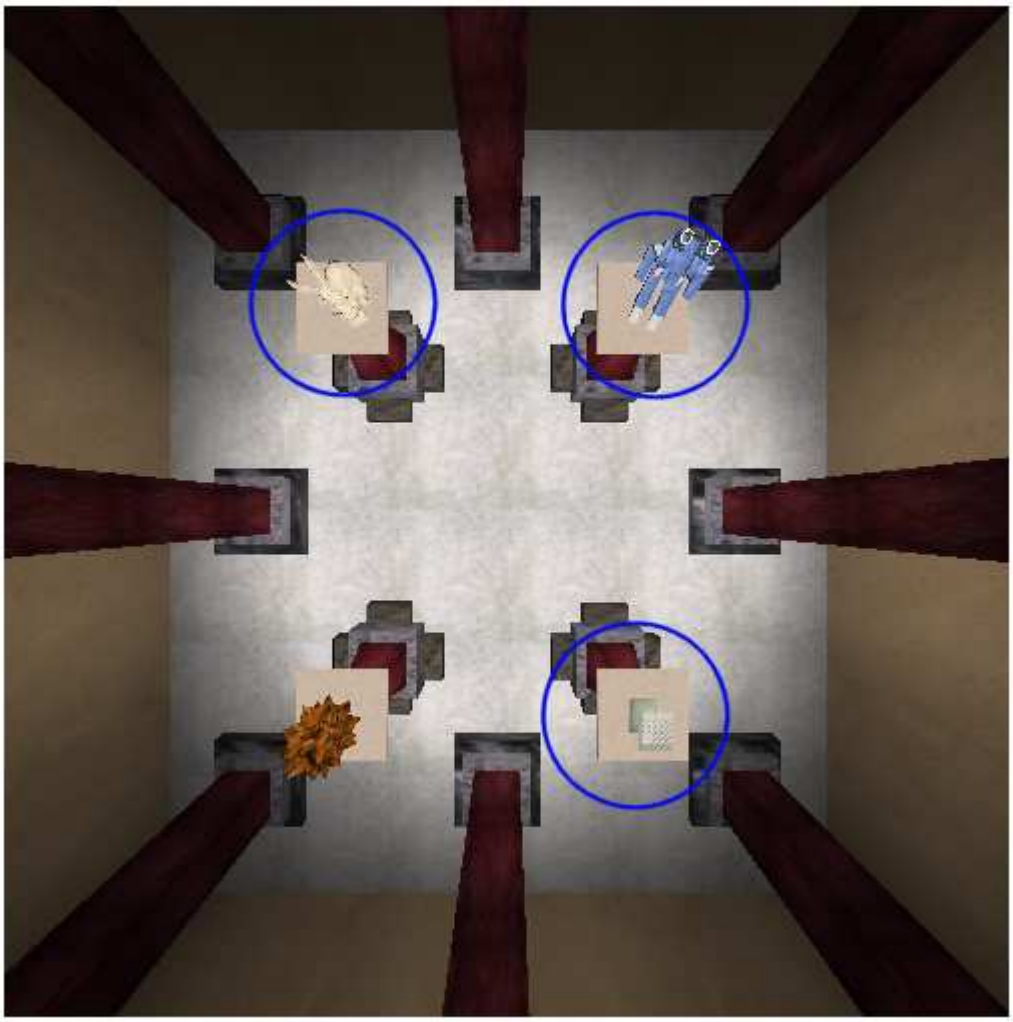


Figure 2.10: The Test Scene (Light View)

The test scene as viewed from the light. The key objects in the scene are circled. Clockwise from the top left, they are: a bunny, a robot, and a mesh sculpture.

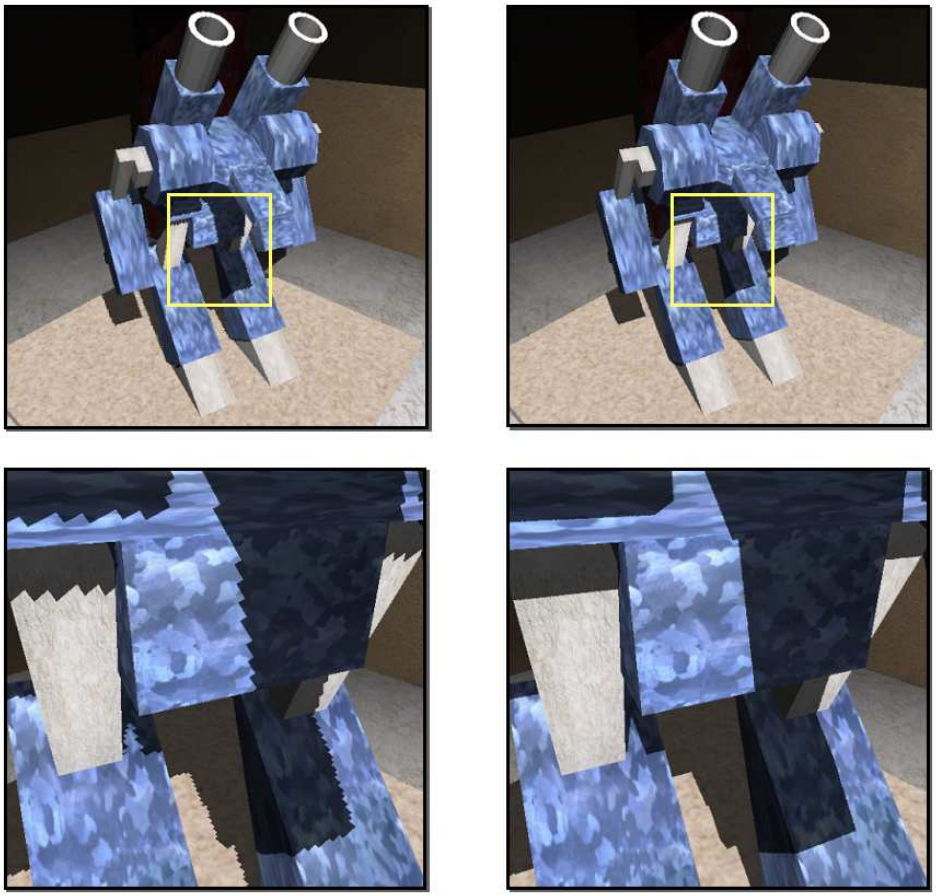
Figure 2.11: The Test Scene (Eye View)

The test scene as viewed from the eye, with the bunny, robot, and mesh sculpture in view.

Results

Our timings were performed on a 1 GHz Pentium III with a NVIDIA GeForce2 Ultra graphics board. During the walkthrough, a conventional 2,048 × 2,048 pixel shadow map (using 16 MB of storage with 32 bits of depth per pixel) averaged 8.5 frames per second, while our algorithm (also using 16 MB of memory) averaged 4.9 frames per second. Figure 2.12 and Figure 2.13 illustrate the

dramatic improvement in image quality achieved by the ASM. For close-ups of objects, the equivalent conventional shadow map size is very large (65,536 × 65,536 pixels in Figure 2.12 and up to 524,288 × 524,288 pixels in Figure 2.13). Creating such a shadow map in practice for an interactive application would be infeasible not only because of the long creation time, but also because of the enormous storage requirements.

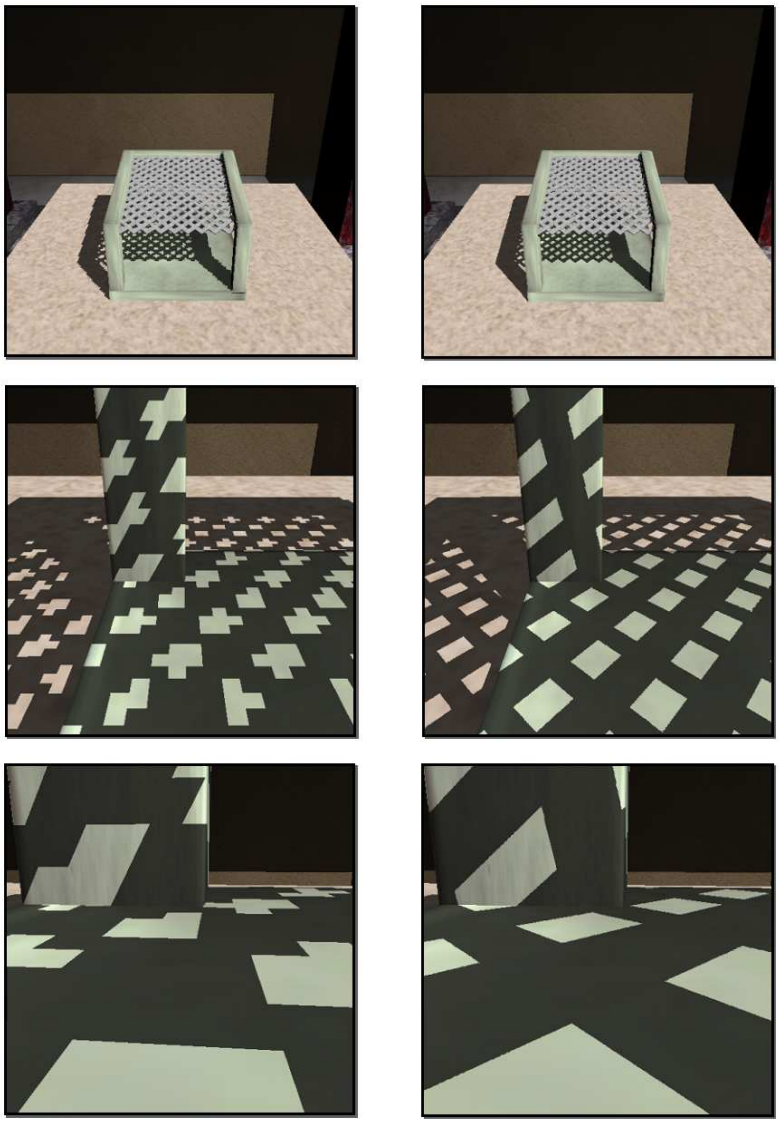


CONVENTIONAL SHADOW MAP
(16 MB MEMORY USAGE)

ADAPTIVE SHADOW MAP
(16 MB MEMORY USAGE)

Figure 2.12: Comparison for Robot Sculpture

The panels show images taken at various distances from the robot. The yellow box in the upper panel shows the region that we have moved towards in the lower panel. The equivalent conventional shadow map size for the ASM in the lower panel is ***65,536 x 65,536*** *pixels.*



CONVENTIONAL SHADOW MAP (16 MB MEMORY USAGE)

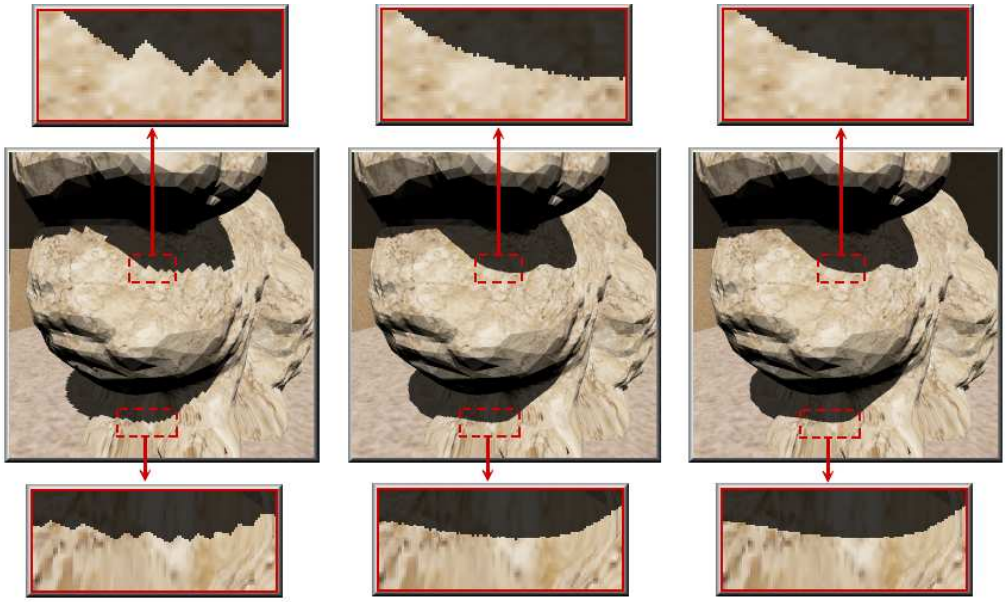
ADAPTIVE SHADOW MAP (16 MB MEMORY USAGE)

Figure 2.13: Comparison for Mesh Sculpture

The panels show images taken at various distances from the mesh sculpture. The equivalent conventional shadow map size for the ASM is ***65,536 × 65,536*** *pixels in the middle panel and* ***524,288 × 524,288*** *pixels in the bottom panel. Note the vastly improved image quality of the Adaptive Shadow Map approach.*

Our results also demonstrate the ASM's ability to accommodate a wide field of view. Because of the ASM's view-driven nature, the starting shadow map size can be relatively small and its field of view can be relatively large. In our walkthrough, the starting resolution of the ASM was 512×512 pixels.

Figure 2.14 illustrates the algorithm's memory management. From left to right, we show images generated with a $2{,}048 \times 2{,}048$ pixel conventional shadow map, an ASM using 8 MB of memory, and an ASM using 16 MB of memory. The differences between the two ASM images are small, but both show considerable improvement in image quality when compared to the image on the left. To highlight the improvement in image quality from an 8 MB ASM to a 16 MB ASM, we have magnified two sections of each image.



16 MB Conventional Shadow Map *8 MB ASM* *16 MB ASM*

Figure 2.14: The ASM's Memory Management

A 16 MB conventional shadow map, an 8 MB ASM, and a 16 MB ASM are compared to show the superior image quality that the ASM delivers, even while using a small amount of memory.

The ASM used approximately 203 ms per frame, while a conventional 2,048 × 2,048 pixel shadow map used 117 ms for the same total memory usage (16 MB). The extra time was spent on cost-benefit analysis (30 ms), node creation (5 ms), traversals through the hierarchy for queries (35 ms), and an extra rendering and read-back of the scene to gather per-polygon ID information (16 ms). To summarize this information succinctly, the ASM uses about twice the computation time as a conventional shadow map, but is able to deliver vastly superior image quality.

CONCLUSIONS

Summary

The ASM is a new technique that uses adaptive subdivision to address aliasing in shadow maps caused by insufficient shadow map resolution. ASMs are view-driven, progressive, and run in a user-specifiable memory footprint. They interface with applications in the same way as conventional shadow maps, allowing them to be easily integrated into existing programs that use software shadow mapping.

Potential Applications

Since ASMs automatically adapt to produce high-quality images and do not require experienced user intervention, they should be useful in interactive modeling applications and in off-line renderers. In addition, a faster version of ASMs, as described below, could potentially be used in applications demanding higher frame rates such as games.

Extensions and Future Work

The algorithm presented in this thesis can be extended to pre-filter and compress refined shadow map cells using techniques described in [Lokovic00]. This would allow for better antialiasing characteristics. Alternatively, percentage closer filtering [Reeves87] could be used for antialiasing, though the filtering across boundaries of shadow map pieces could be troublesome. In addition, perceptual masking [Ferwerda97] can be used to refine less in heavily masked areas.

At the time that the original work on ASMs was done, commercial hardware did not yet have support for shadow mapping. However, the latest generation of graphics cards (such as NVIDIA's GeForce3 and ATI's Radeon) offer hardware-accelerated shadow mapping. We could use this feature to avoid transforming points from the eye view to the light view, which was one of the big bottlenecks in the original implementation. The end result would be a significantly higher frame rate. Although this new implementation is not yet complete, the framework and infrastructure used in Chapter 3 can be easily extended to incorporate the ASM algorithm.

Chapter 3

Adaptive Soft Shadows

This chapter introduces a new technique that assists in soft shadow generation. The technique intelligently varies the number of light source samples used to generate shadows. The concept of a resolution map is introduced to help estimate the number of samples needed, and the methods for calculating the resolution map are explained. The overall implementation of the algorithm is described and future work is discussed in the chapter's conclusion.

MOTIVATION

Over time, many different soft shadow algorithms have been introduced. However, even with all of these approaches, generating soft shadows at interactive rates for dynamic scenes continues to be an extremely difficult problem. Having an interactive tool with such capabilities would allow modelers and lighting designers to evaluate the effects of factors such as light size, location, and geometry on the shadows in an environment. This in turn could dramatically reduce the amount of time spent on modeling and lighting design.

Goals

With the aforementioned observations, the primary goal of our Adaptive Soft Shadow approach is to achieve the highest possible level of interactivity. By interactivity, we refer not only to the frame rate but also to the ability to change as many parameters as possible on the fly. For example, we would like to be able to move objects, move the light, or to change the size of the light. Although this goal might seem lofty, being able to achieve it is genuinely valuable since modelers and lighting designers could then experiment freely during the scene creation process. These goals dictate that pre-processing and caching be kept to a minimum so that full interactivity can be achieved.

Observations

To approximate the variation in visibility over shadow regions, we sample the light source adaptively. Approaches for soft shadow generation typically use a uniform number of light source samples for each pixel, but this does not necessarily have to be the case. Our Adaptive Soft Shadows approach is based on the observation that there are many parts of the scene that require only a few samples to capture the necessary soft shadow details and many other parts of the scene that might not even have shadow details at all. Figure 3.1 shows a scene rendered with 1, 16, 32, 64, 128, and 256 samples per pixel.

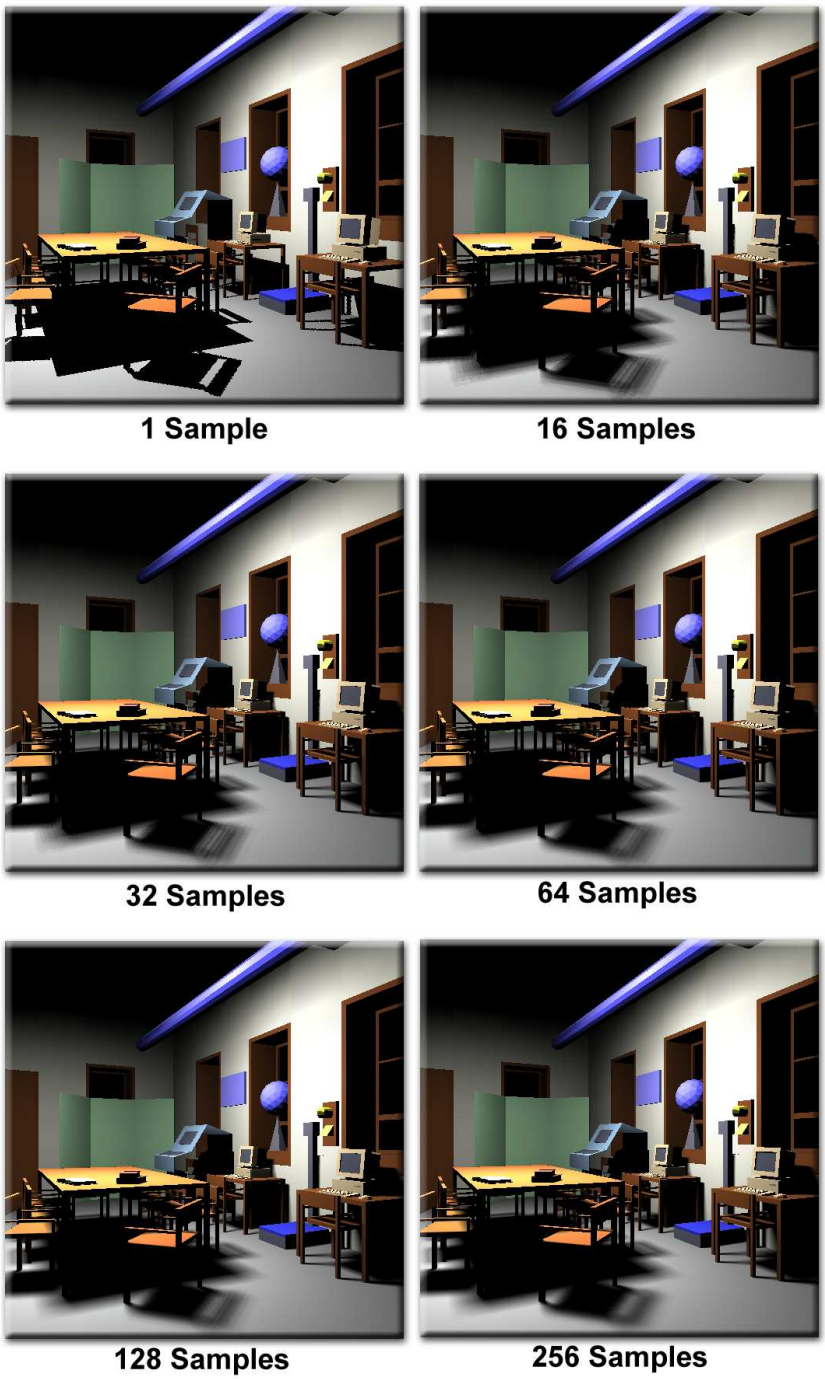


Figure 3.1: Different Samples for Different Situations

We can vary the number of samples used to compute soft shadows based on the projected size of penumbrae.

Comparing the different renderings, it is evident that just a single sample is sufficient in the fully lit (*antiumbral*) or fully occluded (*umbral*) parts of the scene. In the case of partially occluded (*penumbral*) regions, the required number of samples can vary. If we use too few samples in a penumbral region, banding will result. This can be seen in the 16-sample case, where many of the large penumbrae in the scene exhibit banding. As we increase the number of samples used, the banding decreases until it eventually becomes unnoticeable. In some cases, we can remove banding by using only a small number of samples. This concept is illustrated in Figure 3.2. The top image in the figure shows the scene from Figure 3.1, but with two particular regions highlighted with yellow rectangles. The first region shows the thin shadows cast by the chair legs on the right of the scene. In this case, 16 samples are sufficient to remove banding. In the second region, however, the shadow cast on the floor by the chair nearest the eye has a large penumbra that would require between 128 and 256 samples to eliminate banding artifacts.

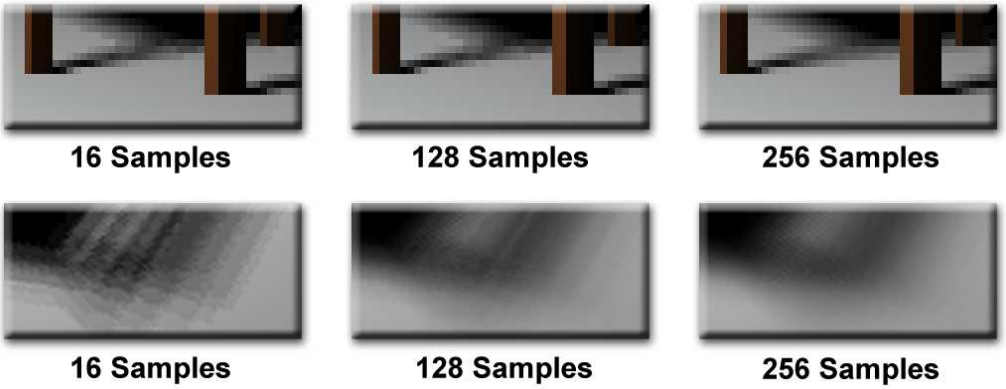


Figure 3.2: Saving Samples

Two magnified sections of a scene are shown, magnified 6.5 times and 4 times, respectively. In the first case, there is practically no observable difference between 16, 128, and 256 samples, meaning that a 16-sample rendering would be adequate. In the second case, however, we would need somewhere between 128 and 256 samples to completely remove banding. Note that the jagged appearance of the top row of pictures is due to their high level of magnification. This jagged effect is not noticeable at the original unmagnified resolution.

The number of samples that is needed to avoid banding artifacts (assuming a regular sampling pattern on the light) depends on how many pixels a penumbra occupies in the eye view. In particular, to have the highest possible image

quality, we need to have n^2 samples for a soft shadow whose gradient covers n pixels in the eye view [Mitchell96].

The Resolution Map

To take advantage of the preceding observations, we need to have some way of varying the sampling frequency as we move spatially through the scene. We do this by introducing an analysis pass prior to the rendering of each frame. There are potentially several ways to estimate penumbra extents, but we need a method that is computationally efficient and sufficiently accurate to eliminate objectionable artifacts in many useful scenarios. We present an approach that attempts to satisfy these criteria and that is also well suited to be used in conjunction with the shadow mapping capabilities of commercial graphics hardware. The hardware-based implementation gives us the ability to approach interactive frame rates as well as to support dynamic environments.

In our algorithm, the analysis pass creates what we call a *resolution map*, which is similar to a shadow map except that instead of storing a depth per texel, it stores the estimated number of samples needed to shade the geometry seen through the texel. Figure 3.3 shows a scene, the resolution map that was generated for it, and the same scene with the resolution map projected onto it. We can see from the figure that the resolution map mandates more effort in regions with large penumbras close to the eye, and very little effort in fully lit or fully occluded regions. The resolution map provides three important functions:

- **Identification of fully lit and fully occluded regions.** Identifying fully lit and fully occluded regions can potentially provide the largest gains, because these are regions where a naïve approach would use a large number of samples to calculate visibility when in reality only one is needed. However, we have to be prudent in our identification process, since we are working from a single shadow map. A single shadow map does not correctly tell us about umbral or antiumbral regions, but we use heuristics to make reasonable estimates.

- **Accounting for the projection of light view pixels into the eye view.** Calculating the projection of light view pixels in the eye view allows us to use less effort in the distance and on surfaces that are oblique to the eye. Without identifying where penumbrae exist, this metric will lead us to expend too much effort on regions near the eye where in fact there are no penumbrae, but where the projection of pixels is large.

- **Estimation of penumbra widths in the eye view.** After the fully lit and fully occluded regions have been identified, the remaining regions contain penumbrae. Heuristics can be used to estimate the penumbra widths and to estimate the number of pixels occupied in the eye view by each penumbra. We can then use as many samples as are needed to remove banding in the penumbra.

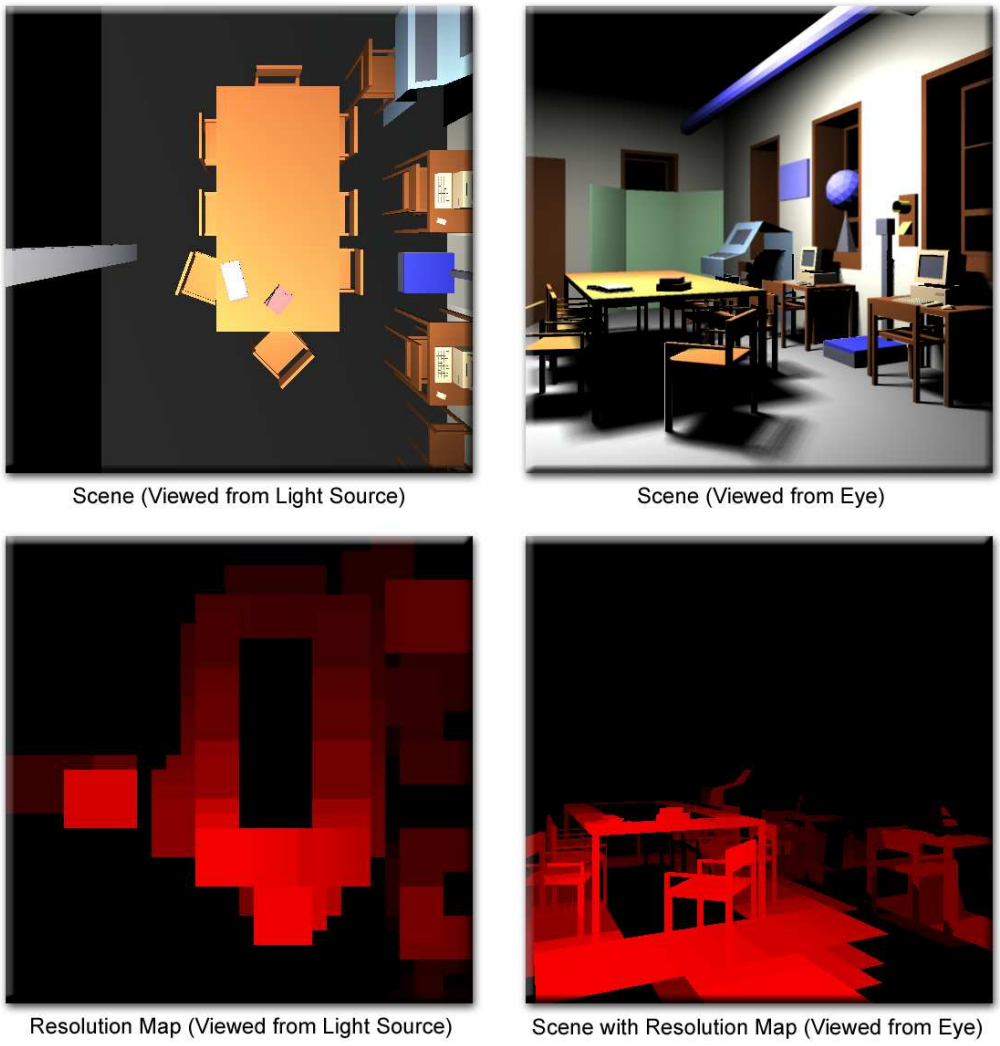


Figure 3.3: Scene with Resolution Map

The top two panels show a scene as viewed from the light source and from the eye. The bottom panels show the same views, but with the resolution map projected onto the scene. Brighter regions of the resolution map require more effort.

PROCESS FLOW

Figure 3.4 shows the process flow for Adaptive Soft Shadows. For each frame, the eye position is updated and a resolution map is created. Then, using the resolution map as a guide, the visibility is computed for each pixel in the scene. This visibility is then modulated with a lit rendering of the scene, producing a rendering with soft shadows.

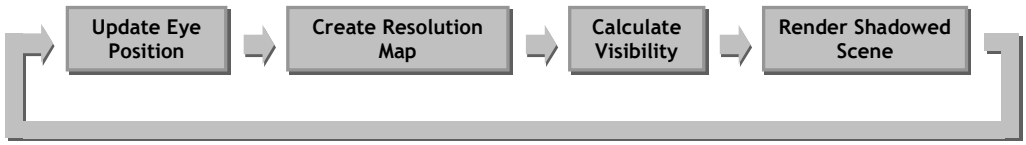


Figure 3.4: Process Flow for Adaptive Soft Shadows

The main steps that the algorithm takes for each frame are shown.

CREATING THE RESOLUTION MAP

To comply with our stated goal of interactivity, it is important that the resolution map be created on the fly in only a fraction of a frame time. Ideally, we would like it to provide a conservative estimate of the number of samples required in each of its cells. This would ensure that using our two-pass technique would not result in noticeably lower image quality than a reference image.

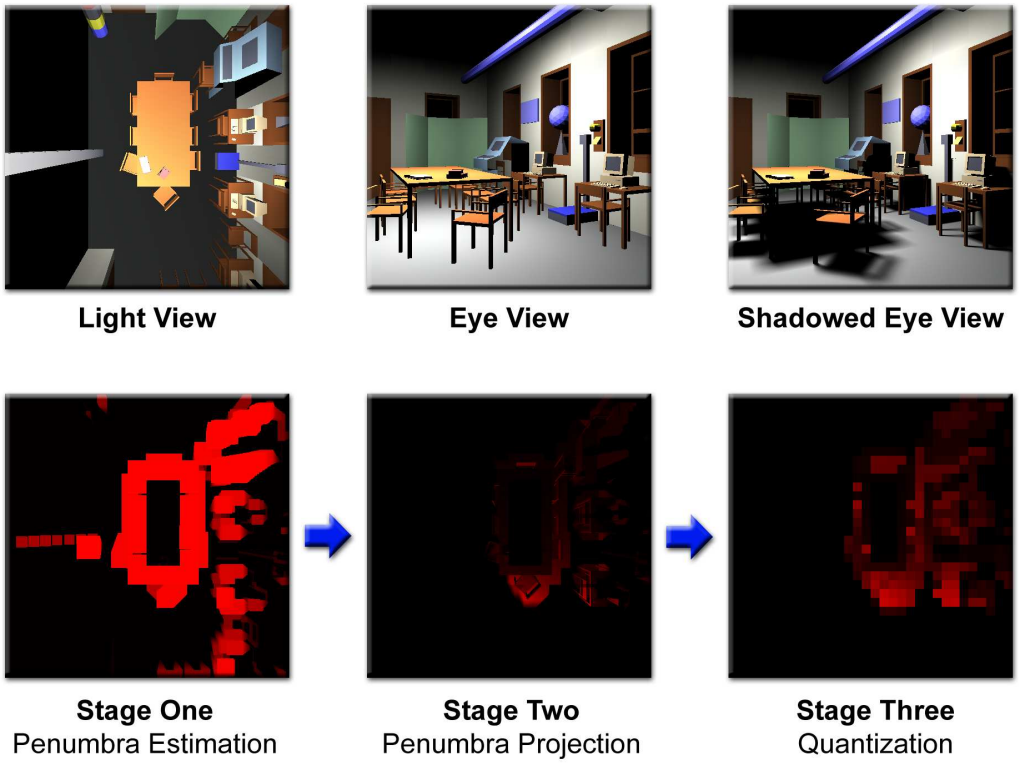


Figure 3.5: The Resolution Map Creation Process

The top row of images shows the light view, the eye view, and a shadowed version of the eye view, to provide context. The bottom row of images shows how the resolution map is constructed for this particular viewpoint. Brighter regions of the resolution map represent areas requiring more effort. After the quantization stage, it is clear that the largest number of samples is required for the large penumbral region near the eye.

Figure 3.5 depicts the three-step process we use to create the resolution map. In the first stage, we estimate the penumbra width from the light's view. Next, we calculate the projection of the penumbrae in the eye view. Finally, in the third stage, we quantize the resolution map into a finite number of cells.

Stage 1: Penumbra Estimation in the Light View

In the first stage, a depth image is generated from the center of the light source and used to estimate the penumbra width at each texel of the resolution map.

To do this, the algorithm loops over all pixels in the depth image and searches for depth discontinuities by comparing the depth of each pixel with its neighbors' depths. When a depth discontinuity is found, the size of the resulting penumbra is estimated. In the interest of speed, we assume that the blocker and receiver are parallel to the light, though this assumption could be relaxed as an improvement. The parallel plane approximation also ensures that the penumbra width on either side of the receiver is identical. Figure 3.6 illustrates the geometry of the situation.

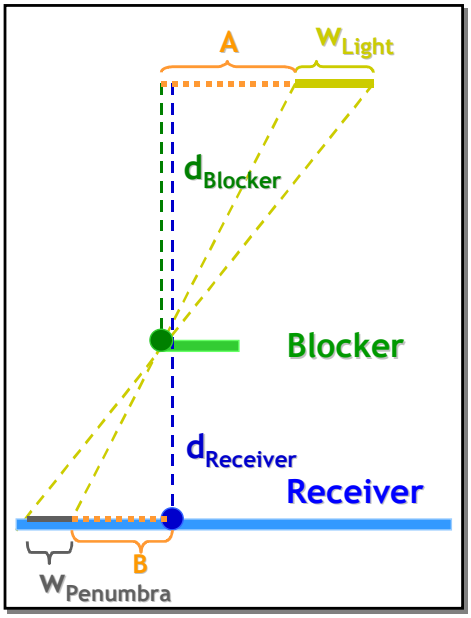


Figure 3.6: Geometry for Penumbra Estimation

The geometry used to estimate the penumbra width is shown. Note the parallel plane approximation. d is used for distances, and w is used for widths.
w_{Light} *is the width of the light source.*
$w_{Penumbra}$ *is the width of the penumbra.*
$d_{Blocker}$ *is the perpendicular distance from the light source to the blocker.*
$d_{Receiver}$ *is the perpendicular distance from the light source to the receiver.*

Because we are dealing with a depth map that has been generated in the light's screen space, we must first transform the blocker and receiver points into world space to perform our analysis. Once we have the world space points, we calculate the penumbra size assuming a simplified parallel geometric configuration. In the parallel case, the penumbra width depends only on the blocker and receiver distances and the width of the light. The derivation is given below:

$$\frac{A + w_{Light}}{d_{Blocker}} = \frac{w_{Penumbra} + B}{d_{Receiver} - d_{Blocker}} \quad (1) \quad \text{Similar Triangles}$$

$$\frac{A}{d_{Blocker}} = \frac{B}{d_{Receiver} - d_{Blocker}} \quad (2) \quad \text{Similar Triangles}$$

$$A = \frac{B \cdot d_{Blocker}}{d_{Receiver} - d_{Blocker}} \quad (3) \quad \text{Manipulating (2)}$$

$$\frac{\frac{B \cdot d_{Blocker}}{d_{Receiver} - d_{Blocker}} + w_{Light}}{d_{Blocker}} = \frac{w_{Penumbra} + B}{d_{Receiver} - d_{Blocker}} \quad (4) \quad \text{Substituting (3) into (1)}$$

$$\frac{B}{d_{Receiver} - d_{Blocker}} + \frac{w_{Light}}{d_{Blocker}} = \frac{w_{Penumbra}}{d_{Receiver} - d_{Blocker}} + \frac{B}{d_{Receiver} - d_{Blocker}} \quad (5) \quad \text{Simplifying (4)}$$

$$\frac{w_{Light}}{d_{Blocker}} = \frac{w_{Penumbra}}{d_{Receiver} - d_{Blocker}} \quad (6) \quad \text{Simplifying (5)}$$

$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}} \quad (7) \quad \text{Manipulating (5)}$$

Now, since our goal is to draw the shadow extents into the resolution map, we must convert the penumbra width into pixels. Figure 3.7 shows the geometry of the situation. Using similar triangles, we can find the penumbra width in pixels at the near plane:

$$w_{Resmap} = \frac{w_{Penumbra} \cdot d_{Resmap} \cdot ResmapWidth}{d_{Projected}}$$

In this equation, *ResmapWidth* is the width of the resolution map, in pixels. Once we have the penumbra width in pixels, we can draw a filled square with the penumbra width as its width into the resolution map. For each pixel in the square, we check to see if the new width value is larger than the previous value in the resolution map. If so, we replace the old value with the new, larger value.

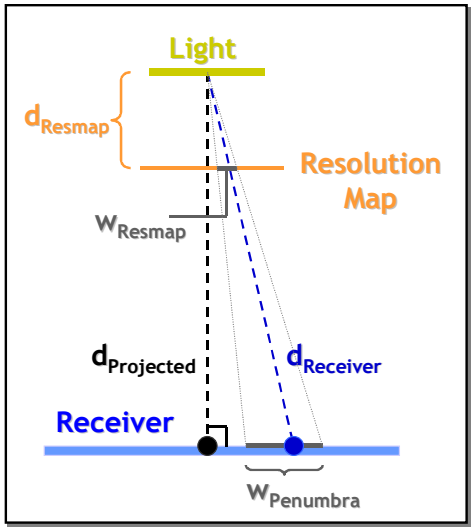


Figure 3.7: Converting the Penumbra Width to Pixels

Once the penumbra width is calculated in world coordinates, it must be projected onto the resolution map. This figure shows the geometry used for this projection. d is used for distances, and w is used for widths.
$w_{Penumbra}$ is the size of the penumbra on the receiver.
w_{Resmap} is the projected size of the penumbra onto the resolution map.
d_{Resmap} is the perpendicular distance of the resolution map from the light source.
$d_{Projected}$ is the perpendicular distance of the receiver from the light source.
$d_{Receiver}$ is the distance from the center of the light source to the receiver.

Stage Two: Projecting Penumbrae into the Eye View

In Stage Two of the resolution map creation, we calculate the projection of penumbra widths that are stored in the resolution map into the eye view. Because penumbra widths (measured in world units) tend to fall off in proportion to the distance from the eye, we use an OpenGL light with attenuation set to 1/r in conjunction with a white rendering of the scene (generated from the point of view of the light source) to calculate the fall-off. We then multiply

each texel of the resolution map from Stage One with the corresponding fall-off value calculated at each pixel in stage two. At this point, we have successfully estimated the projection of all penumbrae in the eye view.

Stage Three: Quantizing the Resolution Map

Stage Three quantizes the resolution map into a fixed number of cells. For each cell, we find the maximum value over all pixels in the cell and use that value for the cell. This quantization step serves two purposes. First, it allows us to efficiently use hardware shadow mapping to render our soft shadows. Second, it helps to mask error in the resolution map caused by the parallel plane assumption in Stage One because the highest value is used for each cell. In return, however, the effort needed to render the soft shadows can increase.

HARDWARE SOFT SHADOW GENERATION

Calculating Visibility

Once we have constructed the resolution map, we are ready to render soft shadows. Since our approach accounts only for the variation in visibility over the scene, we first render a lit version of the scene without shadows. We then calculate the visibility fraction for each pixel:

$$Visibility = \frac{Number\ of\ Visible\ Light\ Source\ Samples}{Total\ Light\ Source\ Samples\ Used}$$

In the simple case where a uniform number of samples is used throughout the scene, the visibility of each pixel can be simply computed as the count at that

pixel divided by the maximum number of samples. However, to take advantage of the resolution map, we must be able to vary the number of samples used for various regions of the scene. Because of this, we need to have a way of counting, for each region, the number of light source samples that actually *light* the region (the numerator), and the number of light source samples that could *potentially* light that region (the denominator). Dividing the numerator count by the denominator count would give an estimate of the light source fraction that is visible at each pixel.

We use a progressive scheme for our samples, meaning that the set of *n* samples is a superset of the set of *n-1* samples. This scheme allows us to use some optimizations when rendering the shadows.

As mentioned previously, we use graphics hardware in our implementation to achieve interactivity. The subsequent sections, we will explain how we calculate the numerator and denominator to find visibility. Before proceeding, however, the following section provides some brief background about hardware shadow mapping. In addition, [Woo99] provides an excellent overview of OpenGL and hardware rendering for readers who may not be familiar with these areas.

Hardware Shadow Mapping Background

A shadow mapping implementation can be classified as either forward or backward. In forward shadow mapping, each texel of the shadow map is projected onto the eye view, a depth comparison is made, and the eye view is shaded appropriately. In backward shadow mapping, the pixels in the eye

view are transformed into the light's view and then compared with the value of the corresponding texel of the shadow map. The original eye view pixel is then shaded based on the depth comparison. The original description of shadow mapping in [Williams78] was a backward shadow mapping approach. Forward shadow mapping was introduced in [Zhang98] and is the technique that is used in hardware because it takes advantage of already existing projective texturing capabilities. This means that we must always think of shadow generation in terms of generating and projecting shadow map pieces from the light view into the eye view. As we will see, this has consequences in terms of efficiency and accuracy.

Since graphics hardware uses forward shadow mapping and the resolution map is generated from the light's point of view, the resolution map is compatible with graphics hardware. To simulate an area light source, we create point light samples that are distributed over the source and combine their effects. Using point light samples allows us to decompose the rendering of soft shadows into multiple hard shadow renderings. Since commercial graphics hardware now supports the shadow mapping technique that was originally introduced in [Williams78], we can use it to quickly generate the shadows for each point light source. As the fragments in the image are rasterized, we can test to see if any particular fragment is lit or in shadow. The stencil buffer can be used to cheaply count the number of light samples that light each pixel. With this technique and the 8-bit stencil buffer that is typically available on most commercial graphics boards today, we are able to track the contributions of up to 256 samples.

To calculate the numerator of the visibility fraction, the algorithm loops over each cell of the resolution map and casts shadows onto the scene from each sample point on the light that is needed for that cell. We use the stencil buffer to count the number of samples that contribute light to each pixel in the eye view. Thus, we could take the following approach, which is illustrated in Figure 3.8:

```
For each cell in the resolution map {
      For each sample required by the cell {
         ▪ Set up a viewing frustum from the sample's position on the light through the cell in the resolution map
         ▪ Render the portion of the scene visible through the frustum
         ▪ Store the resulting image as a depth texture
         ▪ Set up shadow mapping parameters
         ▪ Project the depth texture onto the scene
         ▪ Increment the stencil buffer value for each pixel that is deemed to be lit
      }
}
```

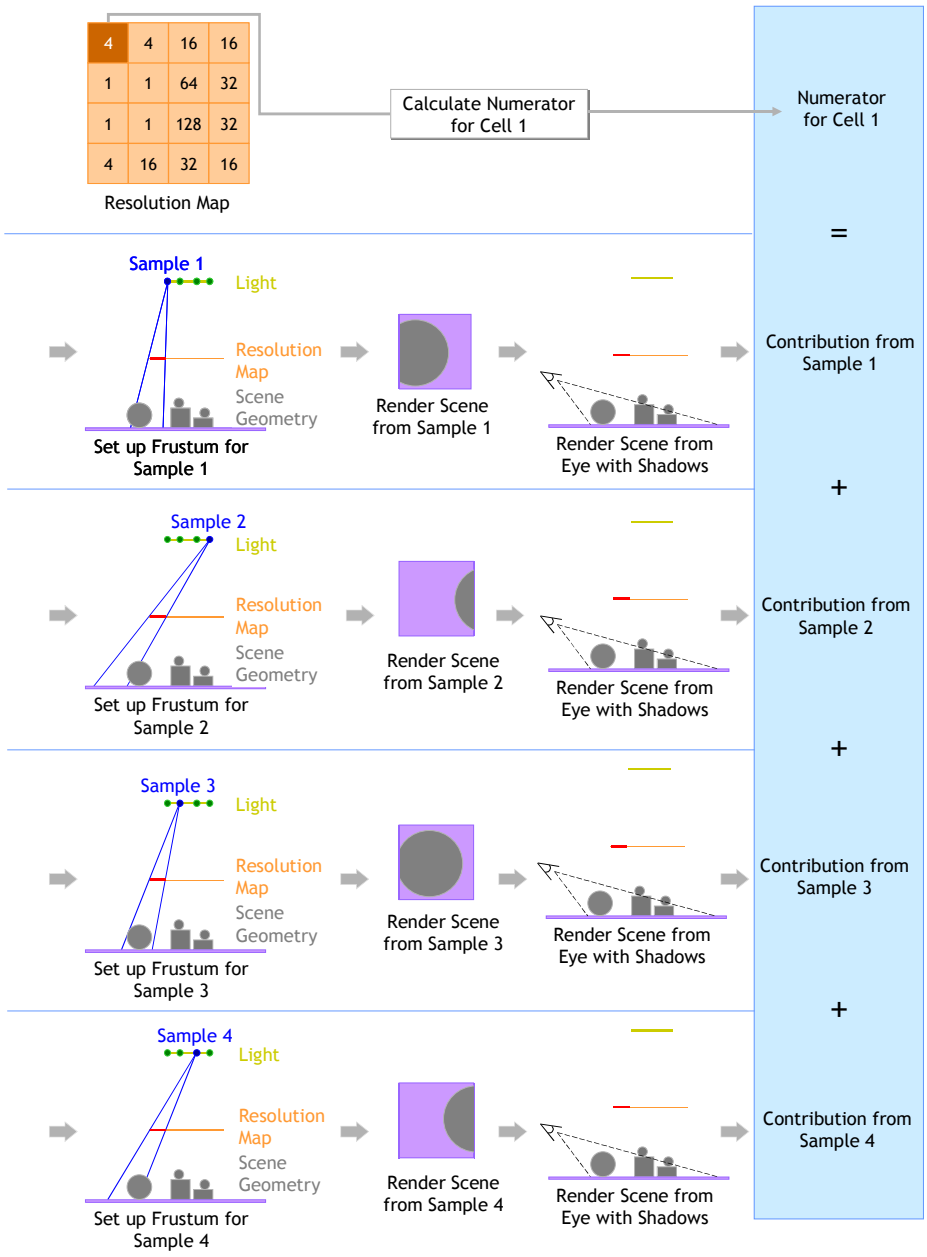


Figure 3.8: Numerator Calculation

The basic steps used to calculate the numerator for a particular cell of the resolution map are shown. In this case, the contributions of four samples are summed for the cell.

This naïve implementation works but is inefficient for two reasons. First, it creates a large number of very small textures, one for each sample of each cell. With graphics hardware, it is more efficient to deal with a smaller number of large textures because texturing overhead can be amortized. The second problem with the naïve implementation is that the eye view geometry needs to be drawn once for each sample of each cell. In practice, we found that the rendering of eye view geometry was one of the slowest parts of the rendering process. Taking these observations into account, we restructured the rendering loop to be more efficient, as follows:

```
For sampleNumber = 1 to MaxSamples {
    ▪ Create a depth texture T with all texels initialized to the minimum depth of 0.0
    ▪ For each cell that requires sampleNumber or more samples {
        o Set up a viewing frustum from the sample's position on the light through the cell in the resolution map
        o Render the portion of the scene visible through the frustum
        o Copy the resulting depth image into the subregion of the depth texture T that corresponds to the current cell
    }
    ▪ Set up shadow mapping parameters
    ▪ Project the depth texture T onto the scene
    ▪ Increment the stencil buffer value for each pixel that is deemed to be lit
}
```

In this pseudocode, MaxSamples is the maximum number of samples required by any cell of the resolution map. Using this version of the algorithm allows us to minimize the number of times the eye view geometry is drawn as well as to minimize the number of texture copies we have to make.

The preceding paragraphs have explained the general procedures of the numerator calculation. More specific implementation details are explained in the following paragraph.

For each sample through a cell of the resolution map, we compute the corresponding region on the near plane. This region is used to define the frustum that we use for computing the shadow contribution of that sample. After rendering the scene from the light's point of view and storing the corresponding depth texture, we then project the depth texture onto the geometry in the eye view. As each piece of geometry in the eye view is drawn, the shadow mapping test colors each incoming fragment as white if its depth is less than the corresponding depth in the shadow map, or black if its depth is greater (meaning that it is in shadow). At this point, we increment the stencil buffer if the incoming fragment is white, and leave it unchanged if the incoming fragment is black. We do this using the alpha test, which culls fragments based on their alpha value. To take advantage of this, we use the register combiner mechanism in the GeForce3 to set the alpha component of the incoming fragment based on its RGB color. We do this by calculating the dot product between the fragment's color triplet and (1,1,1). The resulting value is then placed in the alpha channel. Therefore, if the incoming fragment was white, its alpha component will now be 1.0. If the incoming fragment was black, its alpha compo-

nent will now be 0.0. With the alpha component now set, we can use the alpha test to keep just those fragments with alphas of 1.0. The stencil test is then configured to increment by one at each pixel that receives a fragment. Additionally, we want to make sure that we perform counting only on the surfaces nearest to the eye, to prevent double-counting in the stencil buffer. We achieve this by drawing the scene ahead of time into the depth buffer, and then disabling depth buffer writing.

Merging of Resolution Map Cells

For a resolution map with a significant number of cells (e.g., 32 by 32 divisions), it can be more efficient to group the cells and their corresponding samples instead of rendering them individually. This is because texturing and frustum culling overhead is amortized when dealing with larger cells. To account for this, we use a "mipmap-like" scheme to identify the more efficient groupings. The first level of the mipmap is initialized to contain all the cells of the resolution map. We then perform a filtering process similar to mipmap construction: for each group of four cells at the first level, we check each sample to see if it would be more efficient to propagate the sample upwards to a higher level. To do this, we use calibration data that has been gathered as a pre-process (this can be done for each level or even for each cell of each level, if desired). We repeat this filtering process for every level.

To calculate the denominator value of the visibility fraction, we could simply use the same technique as the numerator calculation uses, except that for each resolution map cell, we would initialize each texel of the shadow map depth texture to 1.0 (the maximum depth value). This would ensure that all geome-

try within the cell would be lit, and the stencil buffer updated appropriately. Thus, we would have identified all geometry that was *potentially* lit by the resolution map cell.

Though the technique just described is simple, it would be inefficient since the scene geometry would have to be drawn repeatedly to perform the projective texturing operation. In reality, we can achieve the same effect in a much simpler way using a technique that is inspired by shadow volumes. We start by clearing the depth and stencil buffers and rendering the scene from the eye. We then disable depth buffer writes. Next, we loop over each cell of the resolution map and, for each of the samples that are needed for that cell, we draw a pyramid from the corresponding sample point on the light through the cell. This is the same pyramid that was used when calculating the numerator. Now, we count the number of frusta that enclose each pixel in the image plane by setting the stencil function appropriately for each face of the pyramid as is done in the shadow volume technique [Crow77]. After doing this for each cell, every pixel in the image plane contains a value equal to the maximum number of samples that could light a particular pixel.

As an additional detail, the stencil buffer must be initialized correctly to account for frusta containing the eye position. We do this by turning the depth test off, inverting the stencil test (i.e., incrementing where we were decrementing before, and vice versa), and then drawing all of the frusta that we use in the denominator counting process described above. When we draw a frustum not containing the eye, the values in the stencil buffer will remain unchanged because both a decrement and increment will be performed for each pixel cov-

ered by the frustum. However, for any frustum that contains the eye, only an increment operation will occur for those pixels that the frustum covers.

IMPLEMENTATION ISSUES

Two important issues that complicate our approach are discussed in this section. Both stem from the fact that we vary the number of samples used throughout the scene.

The Frustum-Matching Problem

One consequence of using hardware shadow mapping in conjunction with a resolution map is that error can be introduced by samples interacting with geometry outside of their intended scope. Figure 3.9 illustrates how this can arise. It shows a simple scenario where we have a light, a resolution map, and the scene geometry. For simplicity, let us consider a 2D case with two samples placed on either end of the light. The resolution map has been divided into cells, and we are currently rendering the shadows corresponding to a particular cell, which has been highlighted on the resolution map plane. First let us look at the shadow map frustum used for sample 0. The frustum originates at the sample position, and the cell of the resolution map that we are working with defines its extent. In this case, we can see that light (and shadows) will be calculated for a region to the right of the scene geometry. Similarly, if we consider sample 1, we can see that lighting and shadows will be calculated for a region to the left of the scene geometry. This means that if were to actually use a scenario like this, our scene geometry would be unlit even though we used two samples from the light!

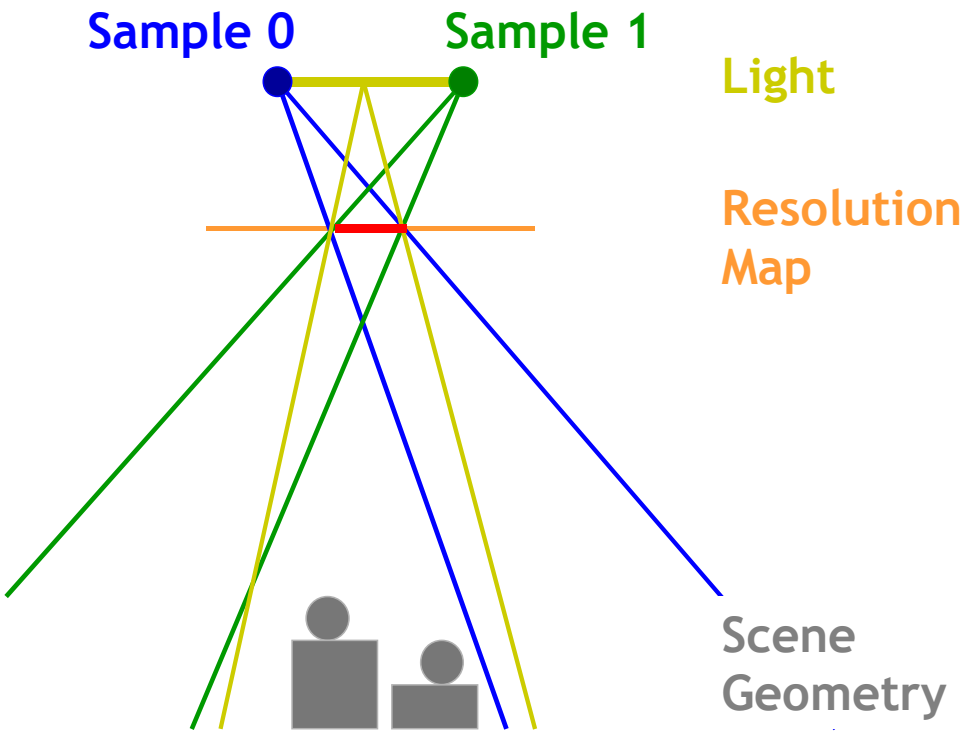


Figure 3.9: The Frustum Matching Problem

The frustum matching problem is shown for a case with just two samples. In this case, the intention was to light the scene geometry with both samples. However, it actually receives light from neither sample because light is misplaced due to mismatched frusta. Note that the level of mismatch has been exaggerated for illustration.

It is important to note that Figure 3.9 is an exaggerated worst-case scenario. In fact, it points out several factors that determine the amount of mismatching that occurs. Figure 3.10 illustrates these factors. The top part of the figure shows the effect of moving the resolution map closer to the scene. With the original parameters, a large amount of light is "lost" or misplaced and no light reaches the scene geometry. In contrast, once the resolution map is moved closer to the scene geometry, far less light is misplaced and the scene geome-

try is actually lit. Therefore, we should aim to place the resolution map as close to the scene geometry as possible. The bottom part of Figure 3.10 shows the effects of changing the resolution map cell sizes. By increasing the size of the divisions, less light is misplaced and again the scene geometry is actually lit whereas it was not with the smaller divisions. Therefore, the smaller the divisions, the larger the error introduced by the frustum matching problem. Finally, the use of more samples that are well distributed on the light source can also help to decrease the effects of the frustum matching problem.

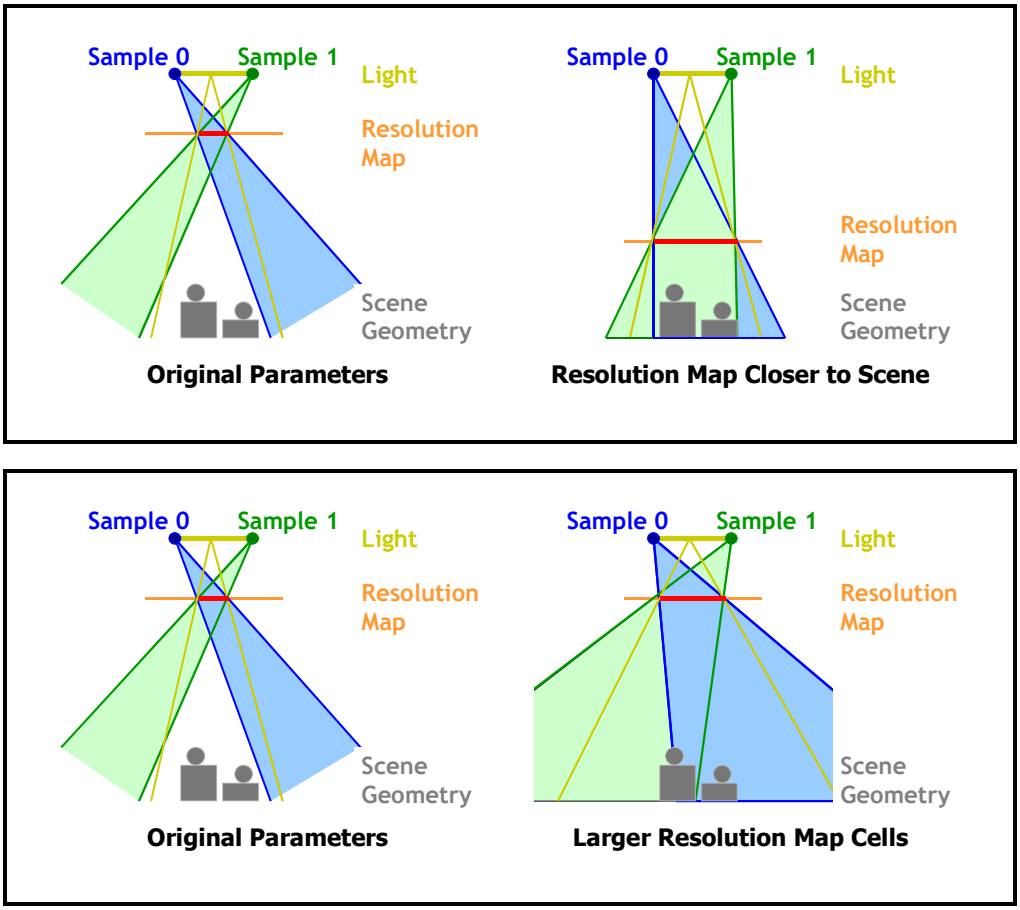


Figure 3.10: Factors that Affect Frustum Matching

The top panel shows the effect of moving the resolution map closer to the scene. The bottom panel shows the effect of increasing the resolution map cell size. In both cases, more light reaches the scene geometry, as originally intended.

The resolution map creation process we described did not account for the effects of the frustum matching problem. In particular, cells that transition from umbra to penumbra or from penumbra to antiumbra can have large differences in the number of samples required, and hence introduce significant artifacts due to misplaced samples. To address this problem, we introduce a "smoothing" stage that attempts to identify large differences in the resolution map cells and adds samples to make the transitions more gradual. Naturally,

there is a trade-off between the reduction of artifacts and the loss of performance caused by using more samples in the resolution map. Thus, we want to smooth the resolution map only where it is absolutely necessary. A user-specifiable threshold can be used to determine where smoothing should be used.

Effective Frustum Culling

Another consequence of using hardware shadow mapping in conjunction with a resolution map is that each piece of the resolution map has to be rendered one at a time to generate the shadows in the eye view. Unfortunately, there is some overhead that is introduced because of this. Ideally, the total cost of rendering each of the cells would add up to the cost of rendering the scene without the resolution map. However, in reality, the ideal behavior is hard to achieve, if not impossible. To get close to the ideal behavior for geometry rendering, an efficient frustum culling scheme needs to be used. This ensures that when a cell is being rendered, only the relevant geometry (the geometry that is visible in that cell) is rendered. Further complicating the issue in the case of Adaptive Soft Shadows is that we do not want any geometry outside a particular cell to be rendered when the cell is being rendered, as this would cause erroneous incrementing of the values in the stencil buffer. To comply with this restriction, geometry must be clipped carefully. Unfortunately, the clipping process introduces additional polygons that can slow the rendering process substantially if the number of added polygons is significant when compared to the original number of polygons.

The frustum culling scheme that we have implemented for Adaptive Soft Shadows is octree based. Initially, a bounding box is defined for all geometry in the scene. This bounding box is then regularly subdivided into eight boxes, and each piece of geometry is tested to see which of the child octree cells it belongs to. This is repeated until a threshold is met, such as a certain number of triangles belonging to a leaf octree cell. One problem with this approach is that the acceleration structure is created as a pre-process, but we want to support dynamic scenes. To circumvent this problem, we store dynamic geometry in a separate list. Of course, our simple frustum-culling scheme can be replaced with another that offers superior performance and support for dynamic scenes.

RESULTS

To demonstrate our algorithm, we show images taken from various viewpoints in several different scenes. These images are shown in Figures 3.11 through 3.14. We compare our algorithm with a brute-force approach also using graphics hardware. We chose the brute-force approach because it is the only other existing approach that is able to generate accurate soft shadows relatively quickly while retaining the ability to change any scene attributes such as the size and position of the light source and the scene geometry. In our examples, the brute-force approach uses the same number of samples as the maximum number requested by our resolution map, to ensure a fair comparison.

The first scene is an outdoor scene consisting of 266,000 polygons. This scene demonstrates the algorithm's ability to deal with large models. The next scene

is a conference room that is made up of about 80,000 polygons. The third scene is a simple room that consists of just 7,000 polygons. The final scene is a room that is sparsely populated with teapots. However, in this case, there are beams blocking the light source, which cause large penumbrae in the room. Each of the figures consists of four rows of images. In the first row, the image on the left shows the image generated by the brute force approach and the image on the right shows the image generated by our approach. In the second row, we color all polygons in the scene white and remove textures, so that masking effects are removed and artifacts are more pronounced. The image from the brute force approach is on the left and the image from our approach is in the center. The right image of the row is the resolution map that was used when rendering with our approach. The third and fourth rows repeat this information for a different viewpoint in the scene.

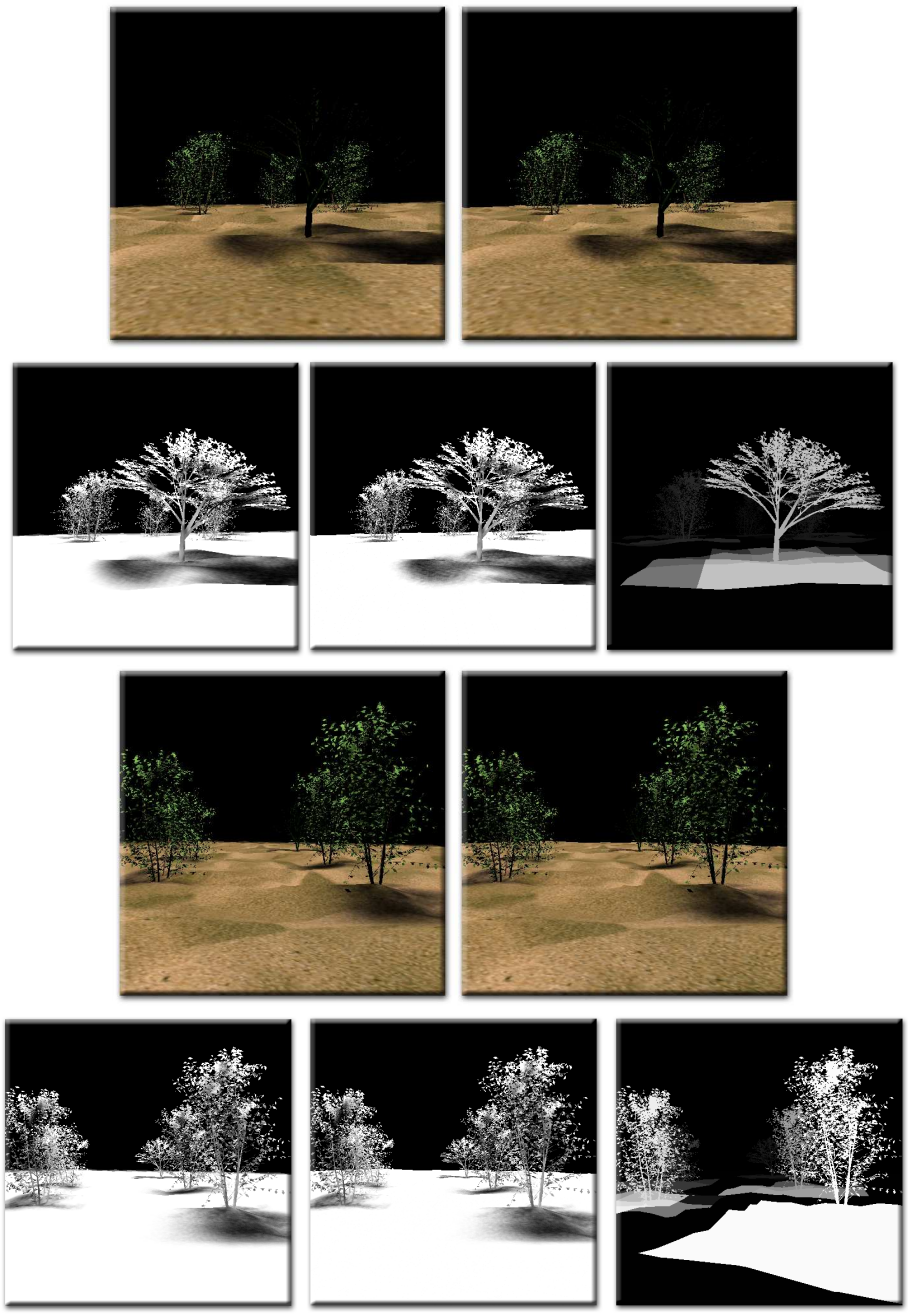


Figure 3.11: Results for Terrain Scene

The first row shows Viewpoint 1. The image on the left uses the brute-force approach, and the image on the right shows our approach. The second row shows a white version of the scene, with the resolution map on the far right. The third and fourth rows show the same information for Viewpoint 2.

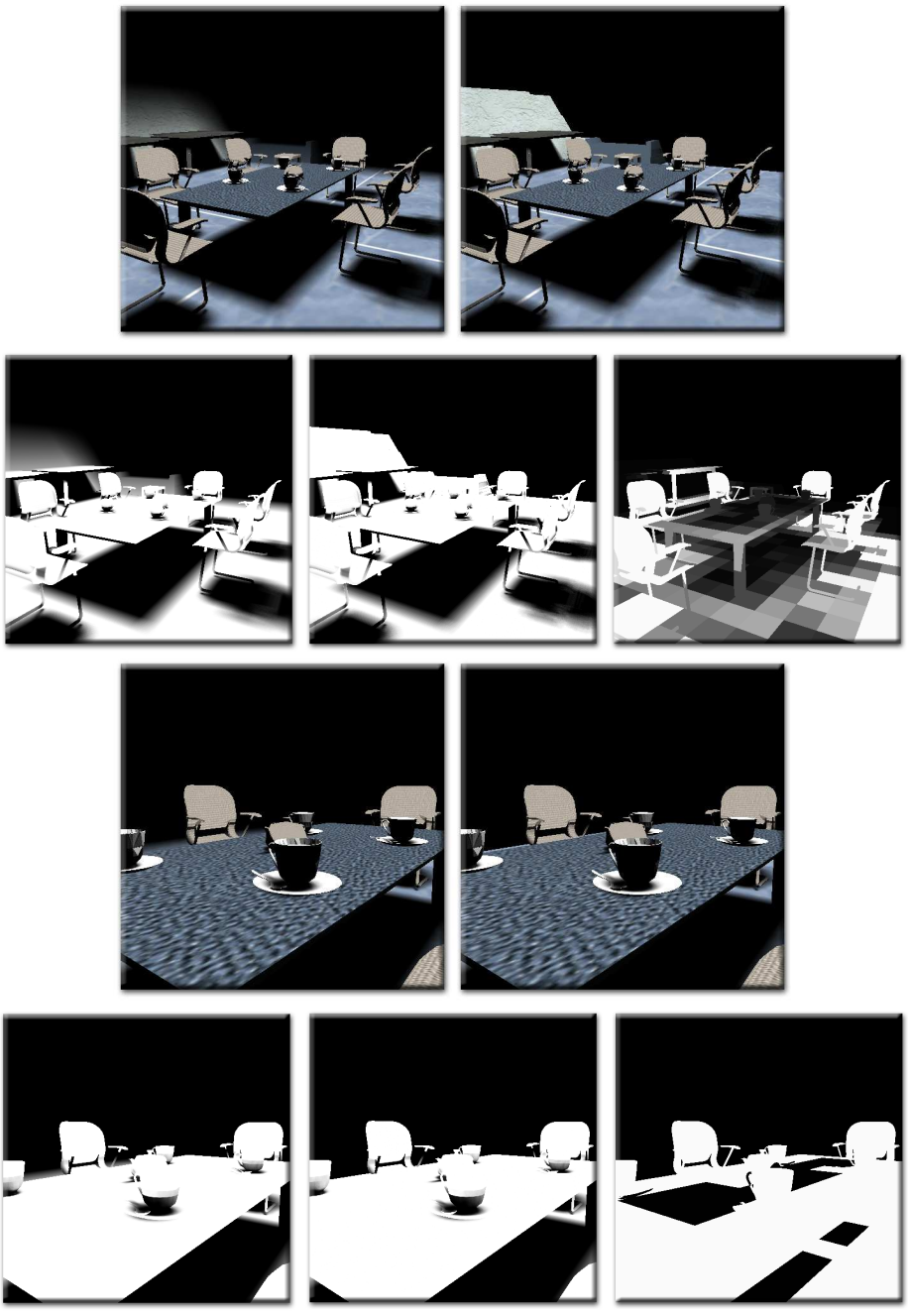


Figure 3.12: Results for Conference Room Scene

The first row shows Viewpoint 1. The image on the left uses the brute-force approach, and the image on the right shows our approach. The second row shows a white version of the scene, with the resolution map on the far right. The third and fourth rows show the same information for Viewpoint 2.

Figure 3.13: Results for Science Center Scene

The first row shows Viewpoint 1. The image on the left uses the brute-force approach, and the image on the right shows our approach. The second row shows a white version of the scene, with the resolution map on the far right. The third and fourth rows show the same information for Viewpoint 2.

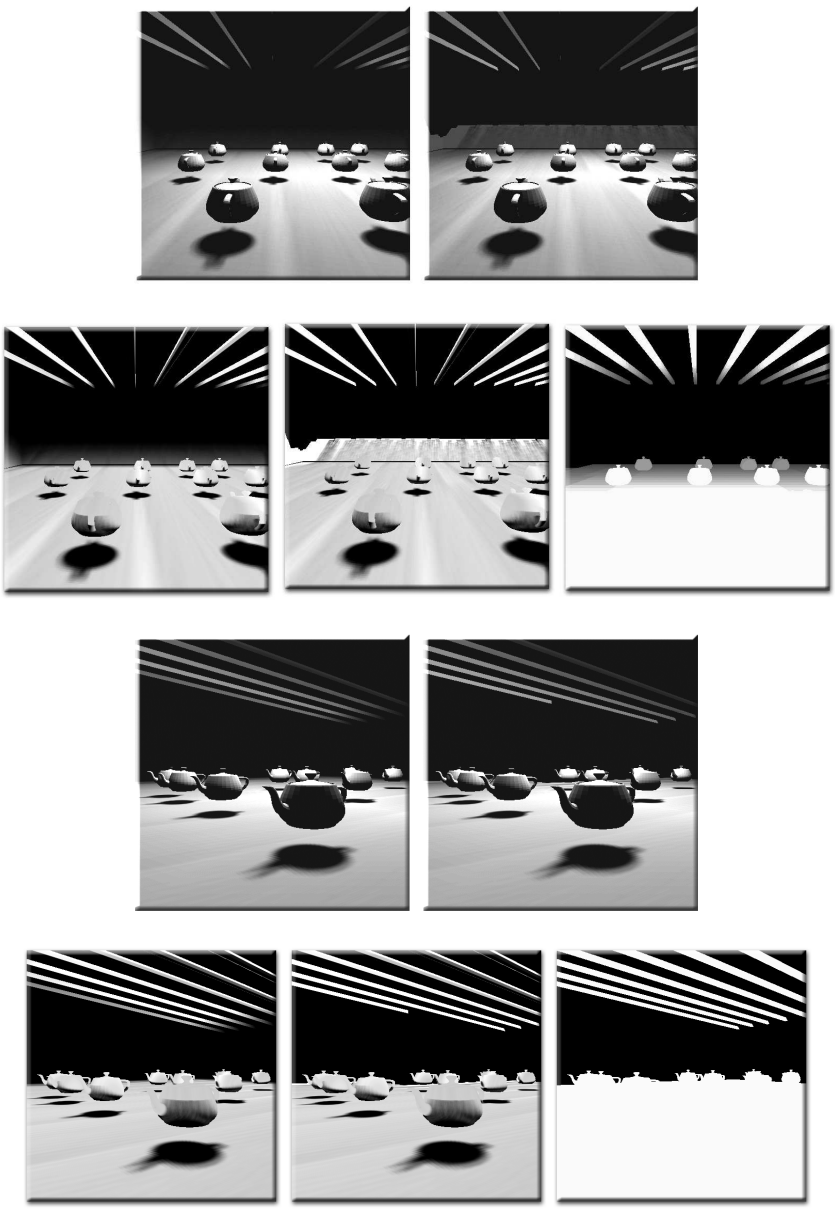


Figure 3.14: Results for Teapot Scene

The first row shows Viewpoint 1. The image on the left uses the brute-force approach, and the image on the right shows our approach. The second row shows a white version of the scene, with the resolution map on the far right. The third and fourth rows show the same information for Viewpoint 2.

The following tables show some relevant performance-related information for each viewpoint in each scene: the number of polygons drawn, the estimated frame time as predicted by the algorithm, the theoretical effort as predicted by the resolution map, and the overall frame time. For each scene, the brute force performance is compared to the performance of our approach. The *Ratio* column in each table represents the performance of the Adaptive Soft Shadows Approach divided by the performance of the brute force approach. Analyzing the statistics, we can evaluate the effectiveness of the resolution map and the overhead introduced by the hardware implementation, along with identifying potential areas of improvement.

Table 3.1: Number of Polygons Drawn Per Frame

	Number of Polygons Drawn per Frame		
Scene	Brute Force	ResMap	Performance Ratio
Terrain View 1	67,968,649	25,281,441	2.69
Terrain View 2	68,120,906	86,288,720	0.79
Conf. Room View 1	23,441,547	10,441,054	2.25
Conf. Room View 2	22,539,733	34,599,675	0.65
Science Center View 1	3,445,690	1,577,367	2.18
Science Center View 2	3,575,763	1,389,280	2.57
Teapots View 1	21,968,288	20,939,996	1.05
Teapots View 2	21,122,876	29,984,486	0.70

Table 3.2: Estimated Frame Time

	Estimated Frame Time (in milliseconds)		
Scene	Brute Force	ResMap	Performance Ratio
Terrain View 1	3125	1429	2.19
Terrain View 2	3125	3602	0.87
Conf. Room View 1	2126	1080	1.97
Conf. Room View 2	2126	2873	0.74
Science Center View 1	1095	683	1.60
Science Center View 2	1095	770	1.42
Teapots View 1	2146	1432	1.50
Teapots View 2	2146	2166	0.99

Table 3.3: Theoretical Effort

	Theoretical Effort		
Scene	Brute Force	ResMap	Performance Ratio
Terrain View 1	100%	23%	4.41
Terrain View 2	100%	55%	1.81
Conf. Room View 1	100%	56%	1.80
Conf. Room View 2	100%	81%	1.23
Science Center View 1	100%	47%	2.13
Science Center View 2	100%	56%	1.80
Teapots View 1	100%	91%	1.10
Teapots View 2	100%	98%	1.02

Table 3.4: Actual Frame Time

	Actual Frame Time (in milliseconds)		
Scene	Brute Force	ResMap	Performance Ratio
Terrain View 1	8035	5760	1.39
Terrain View 2	17152	16960	1.01
Conf. Room View 1	7552	3584	2.11
Conf. Room View 2	7104	9088	0.78
Science Center View 1	1742	1434	1.21
Science Center View 2	2432	1843	1.32
Teapots View 1	6720	6016	1.12
Teapots View 2	6464	8896	0.73

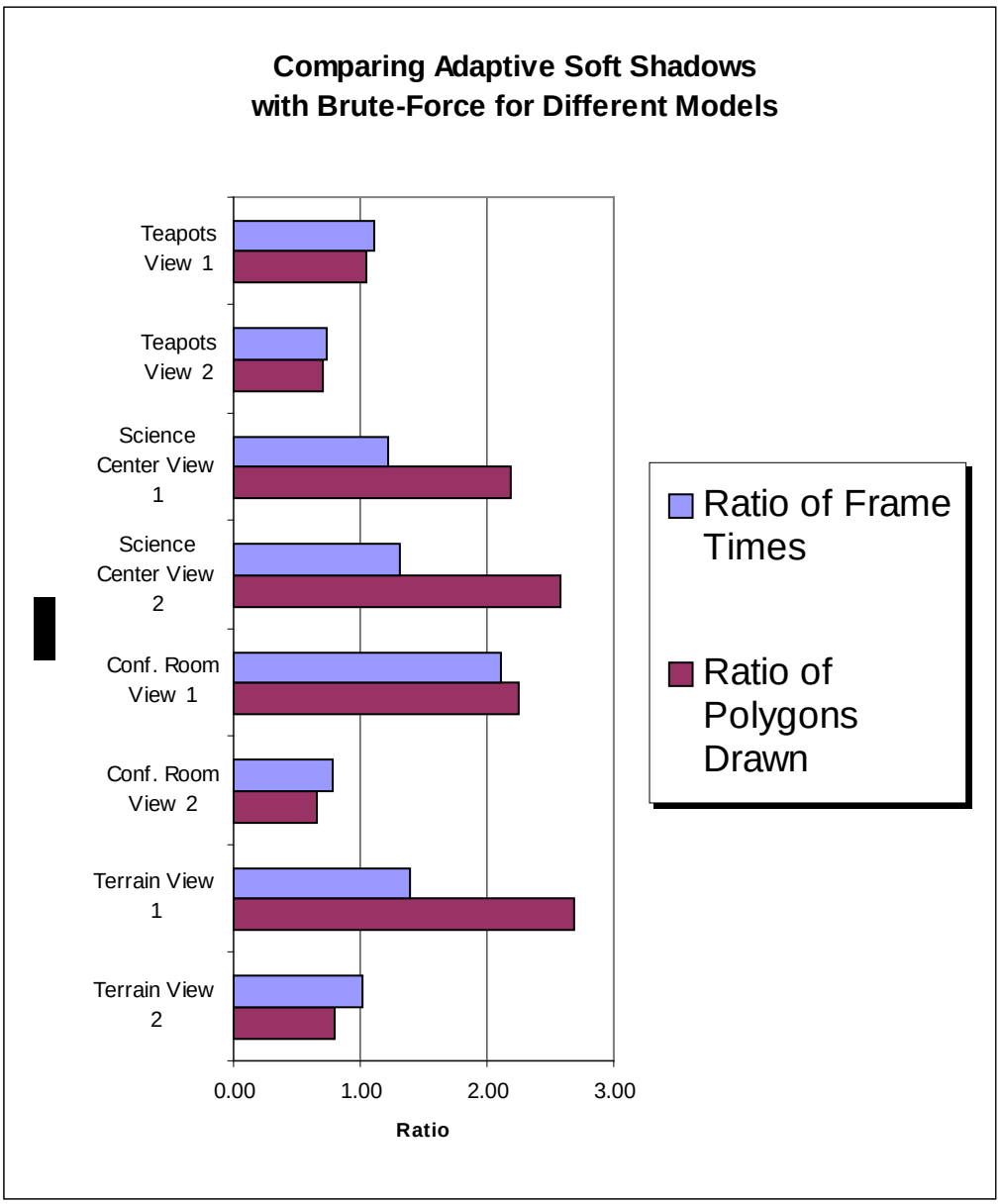


Figure 3.15: Graphing the Data

The figure shows a graph of the data presented in the previous tables. Ratios greater than one represent speedups, and ratios less than one indicate that the brute force approach was better.

Analysis

The results show that the Adaptive Soft Shadows approach was able to deliver performance gains in many of the test scenarios. In each case, the approach was able to construct the resolution map on the order of 100 milliseconds, which was less than 10% of the overall frame time. This is consistent with the original performance goals of the algorithm.

The widely varied results indicate that the algorithm's performance depends greatly on the geometric structure of the test scenes. The largest gains were for the first viewpoint of the conference room scene (a 111% improvement) and the first viewpoint of the terrain scene (a 39% improvement). In these cases, the resolution map was able to effectively identify regions of the scene that needed only a few samples, and thereby provide some significant savings. However, our algorithm fared worse for the second viewpoint for each of these scenes. For these viewpoints, the Adaptive Soft Shadows approach was unable to generate resolution maps with sufficient savings to offset its overhead.

The second observation we can make is that, in most of the cases, even the *theoretical* savings for the algorithm are minimal (around or less than a factor of two). This means that the algorithm is not varying the number of samples sufficiently throughout the scene. This fact in turn points to the resolution map itself as the fundamental cause of poor performance. The problem is that to achieve higher performance, we would like to use a large number of divisions in the resolution map. Smaller cells would result and would allow fine control of the number of samples used throughout the scene, thereby saving

samples. However, using more divisions has two serious drawbacks. First, the image quality can suffer because of the frustum matching problem. As the cells in the resolution map become smaller, the number of misplaced samples grows and this can result in banding. Second, the overhead required by the implementation increases with smaller cells. As was mentioned previously, we use a calibration process to help estimate the overhead, and to allow the algorithm to choose the optimal cell sizes to use in various parts of the scene.

Creating an accurate calibration process proved to be difficult because of the complexity of the implementation. Comparing the values in the Estimated Frame Time table with the values in the Actual Frame Time table, it is clear that the estimation process needs improvement. This fact is somewhat surprising given that the calibration process aimed to simulate the actual algorithm. However, it was difficult to accurately reproduce the algorithm's performance faithfully. The current implementation uses a simplified version of the algorithm for timing, so using a more accurate simulation could improve the timing estimates. One other problem with the timings is that explicit graphics pipeline flushes need to be called for precision (using OpenGL's glFinish() command), and this may result in timings that are different from what the algorithm uses in practice. Without reliable calibration information, the algorithm will create sub-optimal resolution maps.

Finally, we can look at the number of polygons actually drawn by the two approaches for each viewpoint. Comparing the ratio of polygons drawn to the ratio of actual frame times, we can determine whether geometry is a bottleneck. The relevant data is reproduced in Table 3.5. In all cases except for Ter-

rain View 2, both ratios are either greater than one or less than one. This correlation means that the number of polygons drawn is a significant factor in determining the algorithms performance. In fact, the poor performance in Terrain View 2 and Beams View 2 can be attributed at least partly to the large amount of geometry that was drawn by our technique.

Table 3.5: Correlating Geometry and Performance

SCENE	Ratio of Polygons Drawn	Ratio of Actual Frame Times
Terrain View 1	2.69	1.39
Terrain View 2	0.79	1.01
Conf. Room View 1	2.25	2.11
Conf. Room View 2	0.65	0.78
Science Center View 1	2.18	1.21
Science Center View 2	2.57	1.32
Teapots View 1	1.05	1.12
Teapots View 2	0.70	0.73

These results indicate a number of directions that could be explored to improve the fundamental algorithm:

- **Improving the cost estimation and calibration models.** This would allow the algorithm to construct resolution maps that allow higher performance, although visual artifacts could be increased when excessively small cells are used.

- **Incorporating lost samples due to frustum mismatching into the cost model.** This would help to ensure that the algorithm would not use cell sizes that would degrade image quality beyond an acceptable threshold. However, accomplishing this goal would be difficult because it would involve being able to estimate the effects of frustum matching quickly, during the construction of the resolution map.

- **More efficient frustum culling code.** Our algorithm relies heavily on frustum culling to limit the number of polygons drawn as it works through individual cells of the resolution map. Improving the current frustum culling scheme could therefore improve performance. The octree-based culling that we implemented relies on some very important threshold parameters, and an automated scheme to find the optimal values for these parameters would be useful.

- **More efficient rendering code.** Since the drawing of geometry is a bottleneck in our application, it would help to use faster mechanisms to transfer data to the graphics board and to draw the geometry. The current system uses display lists for the majority of its operations because of their ease of use, but vertex arrays could also be used in conjunction with storing data in video memory to deliver higher performance.

DISCUSSION

Contributions

Although the Adaptive Soft Shadows project failed to achieve the desired results, the theory and implementation presented are an interesting study in themselves. The idea of decoupling the rendering of soft shadows into two phases (*the estimation of penumbral extents* and *rendering of the penumbrae using the estimation as a guide*) allows a different and novel approach to the soft shadow problem. In addition, it is interesting to note that there is a very substantial amount of overhead involved with using a piece-wise approach to soft shadow generation. Without our carefully structured implementation (which also uses both shadow maps and shadow volumes in interesting ways), it would be impossible to use a resolution map to achieve any speedup at all in comparison to the brute force approach.

Applications and Extensions

The two-pass approach to soft shadow rendering that we have introduced can be useful in a number of applications. The structure of the resolution map is ideal for texture masking [Ferwerda97] because the savings from texture masking can be easily incorporated into the resolution map construction process. In addition, perceptual evaluations such as the relative intensity between lights, masking due to relative illumination, and the required accuracy of soft shadows can be used to decrease the number of samples used in regions of the resolution map. Texture caching can also be used in conjunction with resolution maps because the required texture storage is significantly reduced in comparison with the brute-force approach. Taking advantage of this observa-

tion could result in an approach similar to that used in Adaptive Shadow Maps [Fernando2001]. For a scene with multiple lights, a resolution map can be created for each light. The resulting information could then be used to reduce the overall rendering effort. Similarly, resolution maps can be used to prioritize effort in distributed systems such as WireGL [Humphreys2001]. The resolution map idea can also be used for other effects, such as motion blur, where the number of samples used for slower or more distant objects could be significantly reduced.

CONCLUSION

We have presented a new approach to soft shadow rendering that we implemented using commercial graphics hardware. Our approach supports dynamic scenes and should scale well as graphics hardware improves. We believe that this approach opens up some interesting new areas for research such as improving the penumbra estimation process and looking at new approaches that can take advantage of the information provided by the resolution map to render soft shadows more efficiently.

Chapter 4

Conclusion

This thesis has presented two techniques for shadow generation based on adaptive schemes. Both algorithms use commercial graphics hardware as a tool to achieve interactive performance. The first algorithm, called Adaptive Shadow Maps, adaptively varies the resolution of a shadow map throughout the scene to provide hard shadow edges without aliasing artifacts. The second algorithm, called Adaptive Soft Shadows, adaptively varies the number of samples used for soft shadow generation throughout the scene to efficiently produce soft shadows.

We showed that Adaptive Shadow Maps are able to dramatically improve image quality with a process that is view-driven, progressive, and confined to a user-specifiable memory footprint, while maintaining interactive rates. However, in the case of Adaptive Soft Shadows, we found that on average the algorithm was able to offer less than a 21% improvement in frame time. This was because the implementation overhead outweighed the performance gains from the scene-specific resolution map.

In implementing the two algorithms, we found that using an adaptive approach in conjunction with graphics hardware can have some significant drawbacks. These drawbacks stem from the fact that graphics hardware is optimized to deliver the highest performance in the large-scale or brute force scenarios. For example, it is always more efficient to clear the screen in one large clear operation rather than to break the clear operation into a corresponding number of smaller clear operations. In the case of read-backs, linearity of this operation is preserved fairly well (see Figure 2.9). Adaptive Shadow Maps take advantage of this fact and therefore do not suffer greatly by using a piecewise approach to shadow map generation. On the other hand, the Adaptive Soft Shadows approach relies heavily on clearing small regions of the screen as well as on copying small textures, which makes it more difficult to achieve the desired level of performance.

In conclusion, the idea of using a resolution map is novel and might find better application in conjunction with a rendering system that scales better as the algorithm renders small pieces of the scene. In this situation, the benefits of the resolution map would not be adulterated by the shortcomings of the rendering system.

Bibliography

[Agrawala00] M. Agrawala, R. Ramammorthi, A. Heirich, L. Moll. *Efficient Image-Based Methods for Rendering Soft Shadows.* Proceedings of SIGGRAPH 2000, pp. 375-384, 2000.

[Appel67] A. Appel. *The Notion of Quantitation Invisibility and the Machine Rendering of Solids.* Proceedings of ACM National Meeting, Pages 387-393, 1967.

[Appel68] A. Appel. *Some Techniques for Shading Machine Renderings of Solids.* AFIPS 1968 Spring Joint Computer Conference, Volume 32, pages 37–45, 1968.

[Atherton78] P. Atherton, Kevin Weiler, Donald P. Greenberg. *Polygon Shadow Generation.* Computer Graphics (Proceedings of SIGGRAPH 78). 12 (3), pp. 275-281, 1978.

[Blinn88] James F. Blinn. *Jim Blinn's Corner: Me and My (Fake) Shadow.* IEEE Computer Graphics & Applications. 8 (1), pp. 82-86, 1988.

[Bouknight70] W.J. Bouknight and K. Kelley. *An Algorithm for Producing Half-Tone Computer Graphics Presentation with Shadows and Movable Light Sources.* AFIPS Conference Proceedings, Vol. 36, 1970, AFIPS Press, Reston, VA, pp. 1-10.

[Brotman84] L. S. Brotman, N. I. Badler. *Generating Soft Shadows with a Depth Buffer Algorithm.* IEEE Computer Graphics & Applications. 4 (10), pp. 71-81, 1984.

[Chen93] Shenchang Eric Chen, Lance Williams. *View Interpolation for Image Synthesis.* Proceedings of SIGGRAPH 93. pp. 279-288, 1993.

[Cook84] Robert L. Cook, Thomas Porter, Loren Carpenter. *Distributed Ray Tracing*. Computer Graphics (Proceedings of SIGGRAPH 84). 18 (3), pp. 137-145, 1984.

[Crow77] Franklin C. Crow. *Shadow Algorithms for Computer Graphics.* 11 (2), pp. 242-248, 1977.

[Fernando2001] Randima Fernando, Sebastian Fernandez, Kavita Bala, Donald P. Greenberg. *Adaptive Shadow Maps.* Proceedings of ACM SIGGRAPH 2001. pp. 387-390, 2001.

[Ferwerda97] J. Ferwerda, S.N. Pattanaik, P. Shirley, and D. P. Greenberg. *A Model of Visual Masking for Computer Graphics*. Proceedings of SIGGRAPH 1997, pp. 143-152, Los Angeles, 3-8 August, 1997.

[Fuchs85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks Jr., John G. Eyles, John Poulton. *Fast Spheres, Shadows, Textures, Transpar-*

encies, and Image Enhancements in Pixel-Planes. Computer Graphics (Proceedings of SIGGRAPH 85). 19 (3), pp. 111-120, 1985.

[Goral84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile. *Modelling the Interaction of Light Between Diffuse Surfaces.* Computer Graphics (Proceedings of SIGGRAPH 84). 18 (3), pp. 213-222, 1984.

[Greene93] Ned Greene, M. Kass. *Hierarchical Z-Buffer Visibility.* Proceedings of SIGGRAPH 93. pp. 231-240, 1993.

[Haeberli90] Paul E. Haeberli, Kurt Akeley. *The Accumulation Buffer: Hardware Support for High-Quality Rendering.* Computer Graphics (Proceedings of SIGGRAPH 90). 24 (4), pp. 309-318, 1990.

[Haines86] E. A. Haines, D. P. Greenberg. *The Light Buffer: A Shadow Testing Accelerator.* IEEE Computer Graphics & Applications, 6 (9), pp. 6-16, 1986.

[Hart99] David Hart, Philip Dutré, Donald P. Greenberg. *Direct Illumination With Lazy Visibility Evaluation.* Proceedings of SIGGRAPH 99. pp. 147-154, 1999.

[Heckbert97] P. S. Heckbert and M. Hurf. *Simulating Soft Shadows with Graphics Hardware.* Technical Report CMU-CS-97-104, CS Dept., Carnegie

Mellon University, January 1997.

[Heidmann91] T. Heidmann. *Real Shadows, Real Time*. Iris Universe, 18:28-31, 1991. Silicon Graphics, Inc.

[Hourcade85] J.C. Hourcade and A. Nicolas. *Algorithms for Antialiased Cast Shadows.* Computers and Graphics, vol. 9, no. 3, pp. 259-265, 1985.

[Humphreys2001] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, Pat Hanrahan. *WireGL: A Scalable Graphics System for Clusters.* Proceedings of ACM SIGGRAPH 2001. pp. 129-140, 2001.

[Keating99] Brett Keating, Nelson L. Max. *Shadow Penumbras for Complex Objects by Depth-Dependent Filtering of Multi-Layer Depth Images.* Eurographics Rendering Workshop 1999. 1999.

[Lischinski92] D. Lischinski, F. Tampieri, and D. P. Greenberg. *Discontinuity meshing for accurate radiosity.* IEEE Computer Graphics and Applications, 12(6):25-39, November 1992.

[Lokovic00] T. Lokovic, E. Veach. *Deep Shadow Maps*. Proceedings of SIGGRAPH 2000, pp. 385-392, July 2000.

[Mitchell96] D. P. Mitchell. *Consequences of Stratified Sampling in Graphics.* Proceedings of SIGGRAPH

96. pp. 277-280, 1996.

[Möller99] T. Möller and E.A. Haines. *Real-Time Rendering*. A.K. Peters, Massachusetts, 1999.

[Nishita85] T. Nishita, E. Nakamae. *Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection.* Computer Graphics (Proceedings of SIGGRAPH 85). 19 (3), pp. 23-30, 1985.

[Nyquist28] H. Nyquist. *Certain Topics in Telegraph Transmission Theory*. AIEE Transactions, 47, 1928.

[Parker99] S. Parker, W. Martin, Peter-Pike J. Sloan, Peter S. Shirley, Brian Smits, Charles Hansen. *Interactive Ray Tracing.* 1999 ACM Symposium on Interactive 3D Graphics. pp. 119-126, 1999.

[Reeves87] W. T. Reeves, D. H. Salesin, and R. L. Cook. *Rendering Antialiased Shadows with Depth Maps*, Proceedings of SIGGRAPH 1987, pp. 283-291, July 1987.

[Segal92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul E. Haeberli. *Fast shadows and lighting effects using texture mapping.* Computer Graphics (Proceedings of SIGGRAPH 92). 26 (2), pp. 249-252, 1992.

[Shade98] Jonathan Shade, Steven J. Gortler, Li-wei He,

Richard Szeliski. *Layered Depth Images*. Proceedings of SIGGRAPH 98. pp. 231-242, 1998.

[Soler98] Cyril Soler, François X. Sillion. *Fast Calculation of Soft Shadow Textures Using Convolution.* Proceedings of SIGGRAPH 98. pp. 321-332, 1998.

[Sutherland74] I.E. Sutherland, R.F. Sproull and R.A. Schumacker. *A Characterization of Ten Hidden-Surface Algorithms*. Computing Surveys, 6(1), pp. 1-55, March 1974.

[Wald01] Ingo Wald, Philipp Slusallek, Carsten Benthin. *Interactive Distributed Ray Tracing of Highly Complex Models*. Rendering Techniques 2001: 12th Eurographics Workshop on Rendering. pp. 277-288, 2001.

[Wanger92] L. R. Wanger, J. A. Ferwerda and D. P. Greenberg. *Perceiving Spatial Relationships in Computer-generated Images*. IEEE Computer Graphics and Applications, 12(3):44-58, May 1992.

[Whitted80] T. Whitted. *An Improved Illumination Model for Shaded Display.* Communications of the ACM, 23(6), pp. 343-349, June 1980.

[Williams78] L. Williams. *Casting Curved Shadows on Curved Surfaces.* Proceedings of SIGGRAPH 1978, pp. 270-274, August 1978.

[Williams83] L. Williams. *Pyramidal Parametrics*. Computer Graphics (SIGGRAPH 1983 Proceedings), 17(3):1-11, July 1983.

[Woo90] A. Woo, P. Poulin and A. Fournier. *A Survey of Shadow Algorithms.* IEEE Computer Graphics and Applications, 10(6):13-32, November 1990.

[Woo99] M. Woo, J. Neider, T. Davis and D. Shreiner. *OpenGL Programming Guide (Third Edition),* Addison-Wesley, 1999.

[Zhang98] Hansong Zhang. *Forward Shadow Mapping*. Eurographics Rendering Workshop 1998. pp. 131-138, 1998.