December 16, 2005

To the Graduate School:

This dissertation entitled "Penumbra Volumes for Real-Time Generation of Perceptually Correct Soft Shadows on Modern Graphics Hardware" and written by Nathan Cournia is presented to the Graduate School of Clemson University. I recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy with a major in Computer Science.

Dr. Andrew Duchowski

We have reviewed this dissertation
and recommend its acceptance:

Dr. Robert Geist

Dr. John Kundert-Gibbs

Dr. Mike Westall

Accepted for the Graduate School:

PENUMBRA VOLUMES FOR REAL-TIME GENERATION OF PERCEPTUALLY CORRECT SOFT SHADOWS ON MODERN GRAPHICS HARDWARE

A Dissertation

Presented to

the Graduate School of

Clemson University

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

Computer Science

---

by

Nathan Cournia

December 2005

Advisor: Dr. Andrew Duchowski

Abstract

Two new shadow generation techniques are introduced to produce realistic, perceptually correct soft shadows. *Penumbra volume mapping* and *penumbra volume mask filtering* exploit modern programmable graphics hardware and are able to shadow dynamic scenes at real-time rates. Penumbra volumes extend previous sampling based approaches by masking regions where expensive sampling and filtering operations take place. Relevant past work is reviewed and benefits and shortcomings of the new techniques are discussed.

Acknowledgements

I would like to thank the members of my committee for their guidance over the course of this project. I would also like to thank the members of Clemson University's Computer Graphics Group for many fruitful discussions. Finally, I would like to thank my family for their support.

Table of Contents

Table of Figures

List of Source Listings

Chapter 1

Introduction

Shadows are important for the creation of realistic computer generated imagery. Shadows provide significant cues to the perception of distance between an object and a nearby surface [Hu et al. 2002] as well as providing visual cues to the geometry of complex shadow casters and to the geometry on which shadows fall – the receivers. The effects of static, cast shadows on perceived depth relations between surfaces is phenomenologically clear, and their perceptual salience has also been empirically demonstrated [Mamassian et al. 1998]. There is a tendency to match a floating object with any dark object on the ground [Ni and Braunstein 2004].

Furthermore, the motion of cast shadows in dynamic scenes provides information about the relative motion of objects. Kersten et al. [1996] introduced a dramatic phenomenon called "illusory motion from shadow" in which the displacement of the shadow relative to a stationary square induced a strong perception of the motion of the square in depth. Kersten et al.'s [1997] results indicate that the motion of the shadow is more important than the correspondence between the size of the object and the size of the shadow in determining the perceived position of the object. Experiments conducted by Ni and Braunstein indicate the importance of common motion of an object and shadow for the dominance of the shadow location over other cues, such as optical contact, in determining the perceived position of objects.

Since Leonardo's [1490] early realization of indirect illumination on shadows, it has been well known that shadows can be delineated by two distinct regions. A point is considered in the inner shadow region, the *umbra*, if it is completely hidden from direct light. A point is classified in the outer shadow region, the *penumbra*, if the light source is only partially occluded. For a given shadow, the umbra and penumbra do not necessarily have to exist simultaneously. Point light sources do not produce a penumbra. Likewise, an umbra may not be present in configurations where shadows cast from a small occluder fall on a distant object.

Many factors contribute to the shape of a shadow. The most important factors are the shape of the light source, the shape of the occluder, the shape of the receiver, and the distance relationship between the light, occluder, and receiver. Other factors, such as inter-diffuse reflections, also play a part in determining the shadow's shape, but are currently too costly to generate in real-time.

Fortunately, generating perceptually convincing shadows is computationally feasible provided that shadows exhibit specific appearance characteristics related to the distance between light, occluder, and receiver.

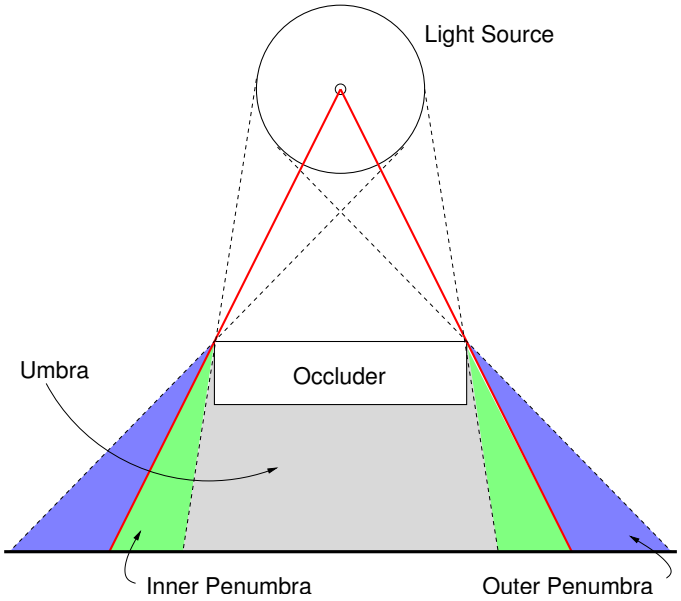

Figure 1.1: Regions of a shadow. A shadow consists of two regions, the umbra (gray) and the penumbra (blue/green). The hard shadow (red), splits the penumbra into two regions: the inner penumbra (green) and the outer penumbra (blue).

The key to producing convincing shadows is the generation of realistic penumbrae. In nature, the penumbra grows as the distance between the occluder and receiver increases or as the area of the light source increases. Conversely, the umbra diminishes as the area of a light source increases, or as the distance from the light source increases. Shadow generation techniques that vary the width of the penumbra according to the ratio of the distances to the light source of the occluder and the receiver are termed *perceptually correct* and are distinguished from more computationally expensive techniques that are able to produce *physically correct* shadows [Hasenfratz et al. 2003].

Early real-time shadow generation techniques [Crow 1977; Williams 1978] focused on casting shadows from point light sources. The results generated by these techniques only produced shadows with an umbra region. These shadows are binary, either a point is in shadow or it is not. This binary decision produces hard, abrupt transitions between shadowed and unshadowed regions. As such, these shadows are labeled *hard shadows*. The abrupt transition from shadow to light is not visually convincing.

In contrast to hard shadows, techniques that generate shadows cast by area light sources, thereby producing a penumbra region, are called *soft shadow* techniques. The term is intuitively appropriate, as the illumination in the penumbra region smoothly transitions from dark to light, producing a soft appearance.

The shadow penumbra can be further divided into two subregions, the inner and outer penumbra. These subregions are contained within the penumbra and are defined by the hard shadow boundary. The hard shadow boundary is produced by sweeping the point light source central to the area light source through the occluders light space silhouette. Figure 1.1 illustrates the shadow regions.

The distinction between the inner and outer penumbra is important for the generation of perceptually correct shadows. It may be especially important for dynamic scenes, since perception of object motion (particularly in depth) is more likely with soft shadows than with hard [Kersten et al. 1996; Kersten et al. 1997]. Unfortunately, many soft shadowing algorithms are unable to generate both subregions in real-time.

Here, we present two new techniques for generating perceptually correct shadows cast from spherical light sources at real-time rates: *penumbra volume mapping* and *penumbra volume mask filtering*. We review previous shadow generation techniques and point out their associated difficulties and limitations. Next, we give an overview of existing work that is relevant to our method. This is followed by a description and discussion of penumbra volume mapping and penumbra volume mask filtering.

Chapter 2

Background

Shadow generation is a difficult task, analogous to computing visibility relationships in 3D [Cohen-Or et al. 2002]. The difficulty is compounded by the requirement of real-time rendering (i.e., $\geq$ 10 frames per second). Many shadow algorithms are able to efficiently produce shadows from point light sources. Unfortunately, light sources in nature are not composed of a single point. Rather they extend over an area, and hence are known as area or extended light sources. The irradiance from an area light source incident on a surface is given by Cohen et al. [1993]:

$$E = \int_{A_{light}} \left[\frac{L \cos\theta_i \cos\theta_l}{r^2} \right] V \, dA,$$

where L is the incident radiance from the light source, θ_i is the incident angle, θ_l is the angle made with the light normal, V is a binary visibility term, and r is the distance from the point being shaded to a point on the light source.

Agrawala et al. [2000] note that for small solid angles the lighting and binary visibility terms can be separated, at which point the shadow attenuation can be computed with:

$$v = \frac{1}{A_{light}} \int_{A_{light}} V \, dA. \tag{2.1}$$

We can then approximate the irradiance with:

$$E = lv, \tag{2.2}$$

where l is the surface irradiance due to a point light source.

In order to generate high quality shadows in real-time, many shadow algorithms solve Equation 2.2 while approximating Equation 2.1. In this respect, our algorithms are no different. In the next section, current shadowing techniques and their limitations are reviewed. Like our algorithms, most of the techniques covered approximate shadows cast from spherical light sources. Some techniques are unable to produce soft shadows while others can produce soft shadows, but are unable to produce both inner and outer penumbra regions. Many of the algorithms are plagued by sampling and banding artifacts. Others produce shadows that resemble soft shadows, but are not perceptually correct. Several of the techniques work only for simple scenes and do

not scale well to multiple light sources and complex scene geometry. Some methods restrict the geometry of a shadow receiver, while other techniques map poorly to graphics hardware. Finally, some shadowing techniques rely on excessive pre-processing and are therefore not suitable for dynamic scenes.

Our goal is to generate perceptually correct soft shadows cast from spherical light sources at real-time rates for dynamic scenes. Our methods map well to modern graphics hardware and are capable of producing both inner and outer penumbra regions. We limit the class of occluder geometry to closed two-manifolds, but allow for arbitrary shadow receivers. The proposed methods are targeted at game engines and previsualization of offline renderings.

Chapter 3

Previous Work

Real-time shadow generation can be categorized by two approaches: image-based and object-based techniques. Most image-based techniques center on *shadow mapping* [Williams 1978], which uses a light space depth image to determine visibility. Conversely, most object-based techniques are based on *shadow volumes* [Crow 1977]. Shadow volumes extrude the light space silhouette of an occluder to infinity. This creates a volume in which points within the volume are considered shadowed. While neither shadow mapping nor shadow volumes inherently produce soft shadows, a great deal of research has been devoted to expanding these techniques to generate soft shadows. We discuss those image- and object-based shadowing techniques that are most relevant to our work.

3.1 Image-Based Methods

Williams' [1978] shadow mapping, the predominant real-time shadow generation technique, is image-based, using a depth buffer constructed from the light's view of the scene to determine the binary visibility of a point with respect to the light. Figure 3.1 illustrates an overview of the shadow mapping algorithm.

Conceptually, shadow mapping is simple. The scene is rendered from the light's point of view, with the resultant depth buffer stored into a texture known as the shadow map. In a second pass, the scene is rendered from the point of view of the camera while the shadow map is projected from the light source. To determine the light visibility of each camera pixel, the depth of the camera pixel is compared to the depth of the projected shadow map texel. If the projected shadow map texel's depth is deemed closer than the camera pixel's depth, the camera pixel is marked as shadowed.

One of Williams' key observations is that a transformation exists such that a point in the camera's view can be mapped into the space of the light's shadow map. Given a point $\mathbf{p}_o$ in object space, $\mathbf{p}_o$ can be transformed into the light's shadow map space with:

$$\mathbf{t}_{sm} = \mathbf{SP}_{light}\mathbf{M}_{light}^{-1}\mathbf{M}\mathbf{p}_o, \tag{3.1}$$

where $\mathbf{M}$ is the matrix that transforms object space to world space, $\mathbf{M}_{light}$ is the transformation from light

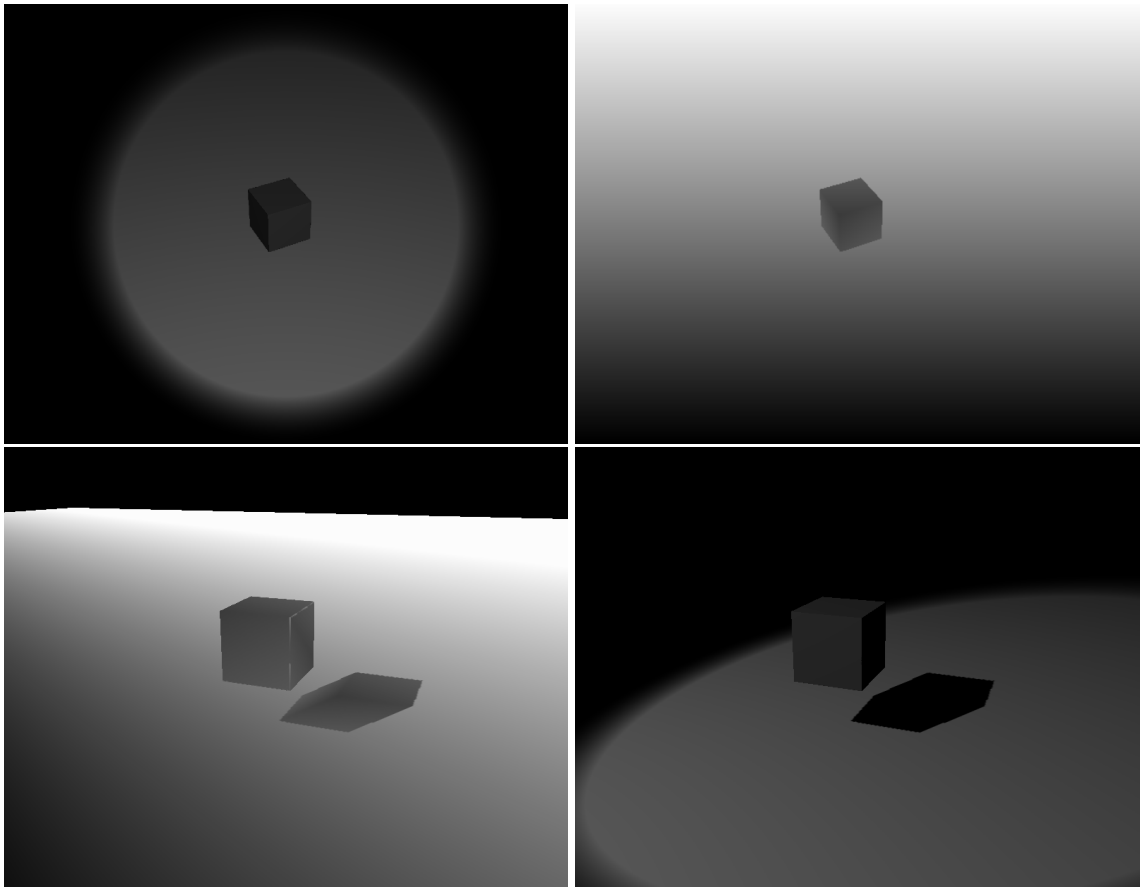

Figure 3.1: Upper Left: The view of the scene from the light source. **Upper Right:** The depth of the scene rendered from the light's viewpoint is saved into a texture called the shadow map. **Lower Left:** The shadow map is projected on the scene. **Lower Right:** For each camera pixel, the depth of the projected shadow map texel is compared to the depth of the pixel. If the shadow map sample is less than the camera pixel's depth, the pixel is in shadow.

space to world space, $\mathbf{P}_{light}$ is the light's projection matrix, and $\mathbf{S}$ is a matrix which transforms points from homogeneous clip space to the shadow map's texture space.

Shadow mapping enjoys several benefits. Unlike previous algorithms, Williams' shadow mapping was one of the first methods to support casting and receiving shadows for non-polygonal objects. This ability is afforded by the algorithm's use of the depth buffer. The depth buffer also allows the cost of shadow mapping to be relatively independent of scene complexity. Since shadows are evaluated in image space, a full description of the scene is not required during the visibility computation.

Segal et al. [1992] provide a hardware accelerated shadow mapping technique exploiting modern hardware's ability to perform perspective correct texture mapping. When using texture hardware is too expensive, Zhang [1998] shows how the use of texturing can largely be avoided using techniques similar to splatting [Westover 1990]. In practice, Zhang's methods are unnecessary as graphics hardware manufactures have incorporated dedicated shadow mapping support into their products, first on the Infinite Reality [Montrym et al. 1997] and today ubiquitously on all modern GPUs [ARB_ shadow].

3.1.1 Shadow Mapping Aliasing

From Equation 3.1 and the image-based nature of the shadow map, it is evident that shadow mapping is view independent. However, the image-based nature of the shadow map gives rise to various aliasing artifacts.

One form of aliasing, *depth aliasing*, occurs due to the quantization of depth values in the shadow map, causing the depth function to be discontinuous. These discontinuities are exposed as the depth function is sampled at a different frequency from the camera's viewpoint. As illustrated in Figure 3.2, depth aliasing causes surfaces to incorrectly self-shadow and produces a visible moiré pattern.

Perspective aliasing [Stamminger and Drettakis 2002] is a result of undersampling the shadow map from the camera's viewpoint and manifests itself in the form of "jaggies". Perspective aliasing is easily seen as the camera zooms into a shadow boundary. Figure 3.3 shows an example of perspective aliasing.

A great deal of research has been devoted to subverting the various forms of aliasing inherent in shadow mapping. Williams proposed that depth aliasing can be removed by adding a bias to depth values either while creating the shadow map or during the shadow test. However, Wang and Molnar [1994] note that depth bias is scene dependent and can be troublesome to compute for dynamic scenes. Additionally, adding a bias results in visible gaps between shadows and their corresponding casters. Wang and Molnar go on to propose *second-depth shadow mapping*. Here, the shadow map stores the depth of the second visible surface with respect to the light's viewpoint. As long as each object in the scene is solid and not intersecting another

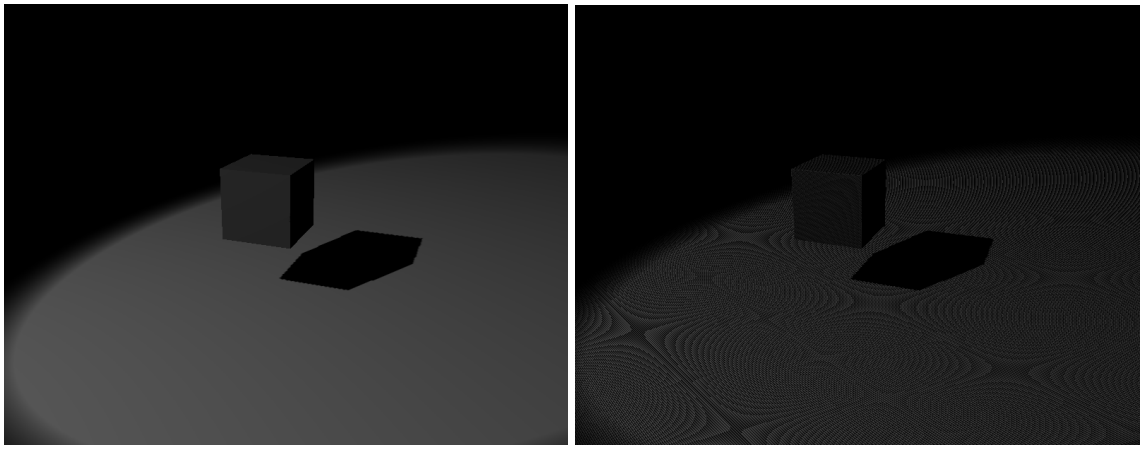

Figure 3.2: Left: Shadows rendered using shadow mapping. **Right:** Depth aliasing causes incorrect self-shadowing resulting in a moiré pattern.

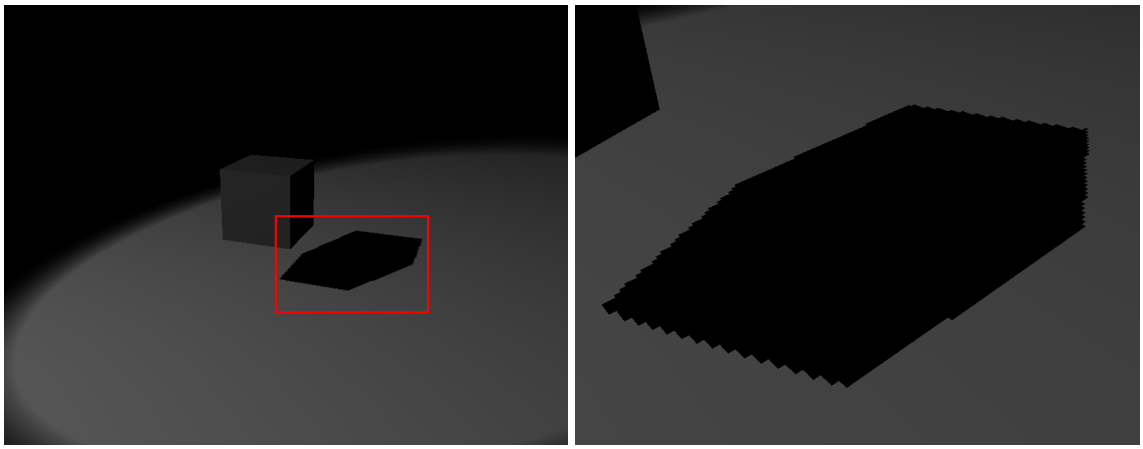

Figure 3.3: Right: As the camera zooms into the region highlighted in the left image, the shadow boundary becomes aliased. This aliasing is due to undersampling of the shadow map.

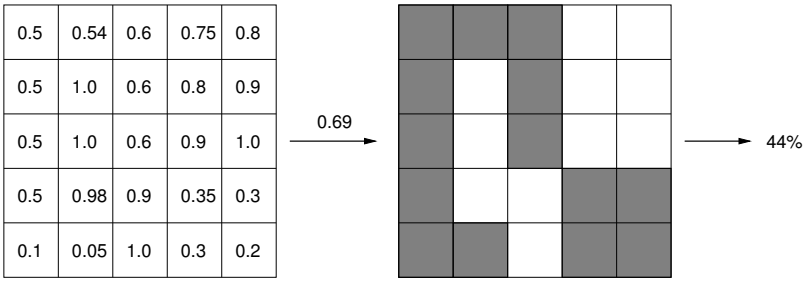

Figure 3.4: A percentage-closer filter samples an area in a shadow map to find the percentage of depth samples that are less than the given fragment's depth threshold. In this example, the threshold, 0.69, is compared to depth values within the shadow map (left). This results in a binary map (right), where each depth sample in the sampling area that is less than the given threshold is deemed in front of the given fragment and therefore shadowing the fragment. In this example 44% of the samples are deemed behind the given fragment's threshold and therefore 44% of the light's total radiance is reaching the fragment.

object, the second-depth surface will be the back surface of an object whose front surface will be in the first-depth shadow map. By performing the shadow test with second-depth surfaces, the authors guarantee that the second-depth surface's corresponding first-depth surface will pass the shadow test, thereby avoiding incorrect self-shadowing. Likewise, using the second-depth surface also guarantees that surfaces beyond the second-depth will be shadowed.

Second-depth shadow mapping does not eliminate depth aliasing, instead it moves depth aliasing from the first-depth surface to the second-depth surface. In practice this is acceptable, as second-depth surfaces point away from the light source and therefore are not illuminated by most illumination models.

Woo [1992] proposed a method similar to Wang and Molnar's to avoid adding a depth bias. By performing the shadow test with the mid-point between the first- and second-depth surfaces, the depth aliasing problem is all but eliminated. Mid-point shadow testing allows for object intersection and arbitrary illumination models at the cost of rendering a second depth map.

Reeves et al.'s [1987] *percentage-closer filtering* attempts to address the aliasing along shadow boundaries, caused by perspective aliasing, by determining the percentage of neighboring pixels around a given pixel that pass the shadow test. The resulting percentage is used to compute the shadow attenuation at the pixel. Figure 3.4 illustrates percentage-closer filtering. Percentage-closer filtering effectively blurs the shadow boundary, producing boundaries that resemble penumbrae. However, these penumbra regions are not physically based. Brabec and Seidel [2001] provide a multi-pass algorithm that maps percentage-closer filtering to graphics hardware. Today, most modern GPUs contain dedicated hardware to fully accelerate percentage-closer filtering [ARB_shadow].

3.1.2 ID Based Shadow Mapping

The shadow test does not necessarily have to be based on depth. Hourcade and Nicolas [1985] store object IDs in the shadow map rather than depth values. During the shadow test, if the shadow map texel's ID does not match the camera pixel's ID, the pixel is deemed shadowed. While this approach avoids depth aliasing, other problems arise. If IDs are stored per object, concave objects cannot correctly self-shadow. To address this problem, Vlachos et al. [2001] suggest assigning IDs per bone or model segment rather than per object. Dietrich [2003] shows how IDs can be stored per triangle on modern graphics hardware and provides a solution to incorrect shadowing caused by rasterization.

3.1.3 Layered and Deep Images

Chen and Williams [1993] propose a simple extension to shadow mapping that simulates soft shadows. Here shadow maps are rendered from a few key sample points on an area light source. Chen and Williams then use view interpolation to create additional shadow maps. For each point to be shaded, the point is tested against each shadow map to compute the average visibility with respect to the area light source. Because sample points are chosen a priori, banding artifacts may occur.

Max [1996] developed a system suited for rendering trees where images are precomputed for twigs and branches at various levels in the hierarchical structure of a tree. During rendering, these images are adaptively combined based on the viewpoint to construct the image of a new tree. Each precomputed image stores surface colors and normals to be used in shading after re-projection, subpixel masks for anti-aliasing, and multiple depth levels to avoid missing detail during reconstruction.

Like Max, Shade et al. [1998] provide a method to reconstruct images that allows for disocclusions and large amounts of parallax as the viewpoint moves. They propose *layered depth images* (LDI). A layered depth image is a two-dimensional array of depth pixels, where each depth pixel is a sorted list of depth values. A layered depth image is constructed by warping multiple depth images into the space of the LDI. For each warped depth value mapped to a depth pixel in the LDI, the depth pixel's list is searched for a value close to the warped depth within a given tolerance. If the value is found, it is averaged with the incoming value. Otherwise, a new layer (element in the list) is added to the depth pixel.

While Max's and Shade et al.'s methods are suited more for reconstructing a scene from multiple images than shadow evaluation, Lokovic and Veach [2000] expand upon the idea of using images that store multiple depth values per pixel, to develop *deep shadow maps* (DSM). Deep shadow maps store the fractional visibility

of a point light source through a pixel for all possible depths. DSMs are able to produce high-quality shadows for high frequency geometry such as hair and fur. Furthermore, deep shadow maps support casting shadows from partially transparent surfaces and volumetric objects. The partial visibility function that a deep shadow map describes is compressed and prefiltered within the DSM to allow for quick sampling. The prefiltered values within the DSM result in shadows that resemble soft shadows, however these simulated penumbra regions are not physically based. Deep shadow maps are expensive to compute and are not suited for real-time rendering of dynamic scenes.

Kim and Neumann [2001] propose *opacity shadow maps*, a method similar to deep shadow maps. Opacity shadow maps use a set of parallel opacity maps oriented perpendicular to the light's direction. Each opacity map stores the integral of densities along a ray through each opacity map pixel. Opacity shadow maps are considerably less expensive to compute than deep shadow maps and are suited for interactive rendering of dynamic scenes.

Agrawala et al. [2000] expand upon Shade et al.'s layered depth images by storing an attenuation factor with each depth sample. Each layer's attenuation can be thought of as a projective soft shadow texture. During display, depth information is used to select the correct attenuation map, which in turn is used to modulate the point being shaded. Computing the layer attenuation image is expensive and not suitable for real-time rendering of dynamic scenes.

Lischinski and Rappoport [1998] use hierarchical raytracing of depth images for computing secondary rays. Keating and Max [1999] note that light leaking is a problem with this technique because each depth sample is treated as an unconnected independent sample. To avoid light leaks, Keating and Max combine adjacent depth samples into discrete depth buckets. This strategy forms large, relatively flat surfaces and, as a possible side effect, may completely change the scene's geometry.

Im et al. [2005] propose a hardware accelerated method to generate soft shadows similar to Keating and Max's, but avoid changing scene geometry. Im et al. use hardware accelerated depth peeling [Everitt 2001] to create layered depth images. When objects are close together, gaps may appear between a shadow and its caster. Similar to Chen and Williams [1993], banding artifacts may occur due to light sample points being chosen *a priori*. The authors report rendering times of several seconds for simple scenes.

St-Amour et al. [2005] present an adaptation of the deep shadow map structure for area light sources, similar to that of Agrawala et al. [2000]. They introduce the *penumbra deep shadow map* (PDSM), which is created from multiple depth images sampled from an area light source. The PDSM stores the attenuation function for every location in the PDSM's field of view. The authors provide a method to store and sample

the PDSM on modern graphics hardware. The PDSM is precomputed and is able to shadow both static and dynamic objects within the scene. However, the PDSM does not encode information about shadows being cast by dynamic objects. As such, the authors recommend using other techniques to cast shadows from dynamic objects since recomputing the PDSM would be too expensive.

3.1.4 Planar Shadows

Heckbert and Herf [1997; 1996] propose a method to cast soft shadows onto planar surfaces using an accumulation buffer [Haeberli and Akeley 1990]. The authors render the hard edged planar projection of the scene's occluders from many area light source sampling points. These images are averaged together to produce a radiance texture (see Heckbert [1990] and Myszkowski and Kunii [1994]) which is used during display to shade surfaces. Heckbert and Herf's method requires many samples in order to avoid banding artifacts and is therefore not suitable for dynamic scenes.

Gooch et al. [1999] show a technique similar to Heckbert and Herf's but suited for technical illustration where the scene's occluders are projected onto a series of stacked planes. The results are averaged to generate soft shadows that approximate shadows cast by a spherical light source. Gooch et al.'s technique has the benefit of requiring fewer samples than Heckbert and Herf's but is still restricted to planar shadow receivers.

3.1.5 Single Sample Soft Shadows

Parker et al. [1998] illustrate a technique to create believable soft shadows in the context of a parallel ray tracer. They propose creating an extended hull for each occluder in the scene. By treating the original object as opaque, and setting a linear falloff from 0.0 at the original object's hull to 1.0 at the extended hull, the contribution of a point light source can be adapted to account for the relative distances of light, occluder, and receiver.

Using Parker et al.'s method, only a single light sample is needed to generate soft shadows. However, the generated soft shadows are not perceptually correct. Because the original object is treated as opaque, the resulting shadow does not have an inner penumbra region, resulting in an umbra that is too large. The authors report that their results look reasonable as long as occluders aren't excessively small compared to the area light source.

Parker et al. note that care must be taken when a shadow ray intersects multiple extended hulls. They state that addition: $s = 1 - ((1 - s_1) + (1 - s_2))$, multiplication: $s = (s_1 s_2)$, and thresholding: $s = \min(s_1, s_2)$ will

all yield continuous intensity transitions. Of the three choices, Parker et al. recommend thresholding since it will result in shadows that are never darker than they should be.

Haines [2001] adapts the work of Parker et al. [1998] to cast soft shadows onto planar surfaces using graphics hardware. Penumbra regions are computed by drawing cones at each light space silhouette vertex. The width of the base of each cone is related to the distance to the ground plane. Cones are connected together with Coons patches. Both the cones and the patches are shaded from dark to light to resemble the shadow's penumbra. The result is a texture that can be projected onto the ground plane. Wyman and Hansen [2003] extend Haines' work by using programmable graphics hardware to allow for arbitrary shadow receivers.

Chan and Durand [2003] introduce *smoothies*, semi-transparent, billboarded quadrilaterals that are attached to the outside of an object's light space silhouette. Smoothies are shaded from 0.0 at the silhouette edge, to 1.0 based on the relative distances from the light, smoothie, and receiver. The result is a smoothie buffer, which is used during display to modulate shaded surfaces.

Inspired by the work of Parker et al. [1998], the techniques of Haines [2001], Wyman and Hansen [2003], and Chan and Durand [2003] all operate at real-time rates but none is capable of producing inner penumbra regions.

Penumbra Quads are primitives utilized in Arvo and Westerholm's [2004] soft shadow technique that is capable of producing both inner and outer penumbra regions. In their technique, inner and outer quadrilaterals are rendered at the light space silhouette of each object. These quadrilaterals are stretched and shaded based on the silhouette vertex's shared face normal and the relative distances between the light, silhouette edge, and the light's far clip plane. This results in two maps, an inner and outer penumbra map. These maps are used in conjunction with the first- and second-depth shadow maps to shadow the scene. The penumbra regions produced are slightly exaggerated and the authors note that artifacts may occur for large light sources. Additionally, visual shadow artifacts may occur if objects overlap in light space.

Brabec and Seidel [2002] generate soft shadows by transforming camera space pixels into the space of the shadow map. They then search the area around each shadow map sample for the nearest occluder boundary. The search radius is based on the distance from the light to the receiver. Brabec and Seidel's search routine may incorrectly select the surface being shadowed as an occluder, which leads to objectionable visual artifacts. To avoid this, the authors assign each object a unique ID. During the search routine, if an occluder is found matching the ID of the shadow receiver, the search is continued. Using object IDs eliminates the incorrect self-shadowing along with correct self-shadowing.

Like Brabec and Seidel, Fernando [2005] also uses a search routine in a single depth map to produce soft shadows. Given a point to be shaded, the depth map is used to compute the nearby average occluder depth. The average occluder depth, the light size, and the distance from the light to receiver, are then used to modify the size of a percentage-closer filter. The filter is then used to sample the depth map, the result of which is used to modulate the point being shaded. Fernando has implemented his technique using modern programmable graphics hardware. To avoid visual artifacts, an excessive number of samples are needed.

3.1.6 Multiple Maps

Heidrich et al. [2000] propose a method to generate soft shadows for linear light sources. Each linear light source is sparsely sampled to generate multiple shadow maps. Depth discontinuities are found using an edge detection filter. The discontinuities are triangulated, warped, and then rendered into a visibility map, which stores the percentage of the light source that is visible from a point in the scene. During display, the visibility map is used to attenuate points being shaded. The authors report that the construction of the visibility map is expensive, taking several seconds to generate for simple scenes. Ying et al. [2002] generalize Heidrich et al.'s method to polygonal light sources.

Kirsch and Döllner [2003] use both a depth map and a shadow width map to generate soft shadows. The shadow width map measures the distance to the closest point that is geometrically illuminated. The shadow width is used in concert with a depth shadow map to generate penumbras within hard shadow regions. As a result, Kirsch and Döllner's method is unable to produce outer penumbra regions and generates artifacts for objects that overlap in light space.

Arvo et al. [2004] compute light visibility values by using an edge detection filter to find hard shadow boundary pixels in the camera's image space. These boundary regions are then dilated by using an eight connected recursive flood-fill mechanism over multiple passes. Visibility is computed by creating a ray from the dilated shadow boundary pixel to the shadow map border pixel and intersecting this ray with the light source. The dilation process requires multiple passes and the algorithm suffers from artifacts when objects overlap in light space. However, since the visibility calculation takes place in the camera's image space, undersampling artifacts are effectively hidden.

3.1.7 Point-Sampling Based on Instant Radiosity

Rendering shadowed scenes from traced reflections of the initial light path can lead to synthetically approximated radiosity, as demonstrated by Keller [1997]. Keller's "instant radiosity" is achieved through a quasi-Monte Carlo integration and a collection of hardware renderings with point light sources. Multiple rendering passes are combined in an accumulation buffer. The result, using Heidmann's [1991] shadow volume variant and displayed for a given viewpoint, exhibits approximated smooth shadows. In a viewpoint independent approach, Van Pernis [2004] uses OpenGL's feedback mechanism instead of the accumulation buffer to accumulate first-order vertex lighting. Employing the depth map for shadow generation, Van Pernis then performed *r-refinement* of surface meshes to alleviate shadow aliasing at mesh-shadow boundaries. Although both approaches led to pleasing soft shadows, neither was particularly fast: Keller reported rendering rates of a few seconds per frame, while Van Pernis generated a 600 frame animation averaging 78 seconds per frame. Approximations to global illumination such as these are approaching interactive frame rates [Kollig and Keller 2004].

3.2 Object-Based Methods

The real-time shadow generation technique competing with the image-based approach is based on shadow volumes, first proposed by Crow [1977] and mapped to graphics hardware by Heidmann [1991] using the stencil buffer. In the first pass, the scene is rendered with ambient light, which in addition to contributing ambient lighting to the scene, also initializes the depth buffer for subsequent passes. In a second pass, semi-infinite quadrilaterals are extruded from possible light space silhouette edges. Front facing shadow volume quadrilaterals which pass the depth test increment the stencil buffer, while back facing quadrilaterals decrement the stencil buffer. This creates a mask in the stencil buffer, where stencil pixels with non-zero values are denoted as in shadow. In a final pass, the scene is rendered with diffuse and specular lighting where the stencil mask is set to zero. Figure 3.5 (left) illustrates the stencil buffer updates for an example scene. While simple, the method described above fails when the camera viewpoint is located inside of a shadow volume. This situation results in an incorrect mask generated in the stencil buffer, leading to surfaces being incorrectly shadowed. Figure 3.5 (right) shows an example of stencil buffer updates when the camera is located within a shadow volume. Bilodeau and Songy [1999] and Carmack [2000] independently discovered a method to solve this problem. They propose decrementing the stencil buffer while rendering front faces and incrementing for back faces. However, stencil buffer updates occur only when the depth test fails, rather than

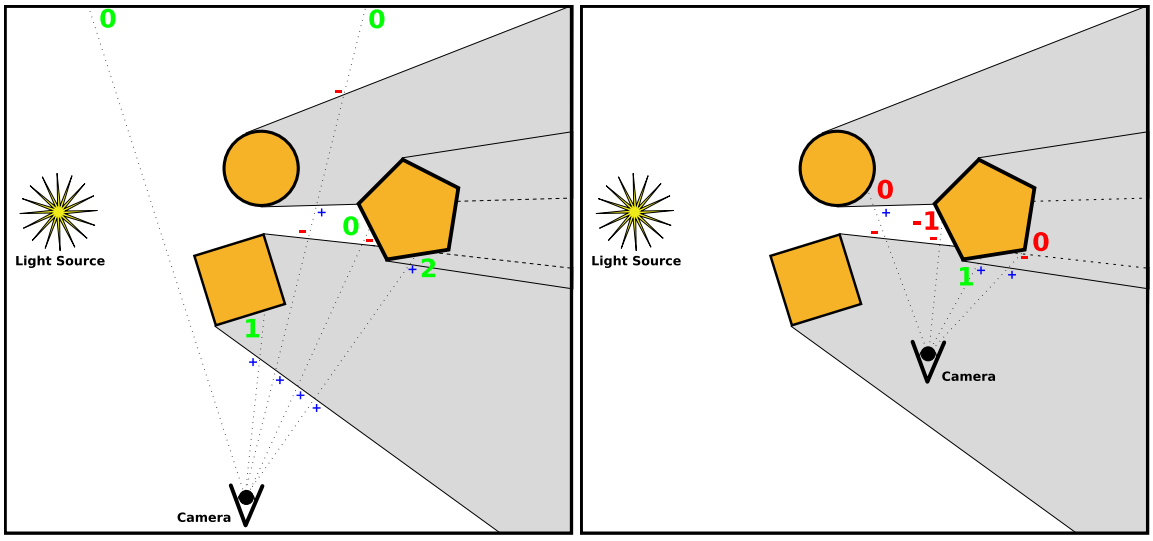

Figure 3.5: Left: Shadow volume stencil buffer updates. As faces of the shadow volumes (gray) generated from occluders (orange) are rendered into the stencil buffer, front faces that pass the depth test increment (blue) the stencil buffer, while back faces decrement (red). This generates a mask (green) in the stencil buffer, where non-zero values indicate the pixel is in shadow. **Right:** An incorrect shadow mask is generated when the camera enters a shadow volume. Numbers in red denote an incorrect mask value.

when it passes. The latter shadow volume method is know as *z-fail* and the former as *z-pass*. Each method is intuitively named for the type of depth test needed to update the stencil buffer.

In order to work properly, *z*-fail shadows require that shadow volumes be capped. This requirement leads to problems, as parts of the shadow volume may be "sliced-open" by the far clip plane. Everitt and Kilgard [2002] propose a solution to this problem by manipulating the camera's projection matrix to move the far clip plane to infinity. The *z*-fail front shadow volume cap also leads to *z*-fighting at the occluder's front faces. This problem can be solved by adding a bias or using the occluder's back faces as a front cap [Zioma 2003]. The latter technique is inspired by Wang and Molnar's [1994] second-depth shadow mapping.

Due to stencil buffer updates, shadow volumes are fill-rate limited. As such, a good deal of research (see McGuire et al. [2003], Lloyd et al. [2004], Laine [2005], and Lengyel [2002]) has focused on limiting the number of stencil buffer updates.

While fill-rate is a limiting factor for shadow volumes on graphics hardware, silhouette detection is a limiting factor on the CPU. To make shadow volumes less CPU bound, Brabec and Seidel [2003] describe a method to both detect and extrude silhouettes on modern programmable graphics hardware. Likewise, McCool [2000] uses an edge detection filter to find discontinuities in a depth shadow map. These discontinuities are then extruded to form shadow volumes.

Shadow volumes as described above produce pixel accurate shadows. However, these shadows are hard shadows. Both Brotman and Badler [1984] and Everitt and Kilgard [2002] extend the shadow volume algorithm to cast soft shadows. They achieve this by rendering multiple shadow volumes from random samples on an area light source. This approach is not practical for real-time rendering. By rendering multiple shadow volumes, the authors are increasing the fill-rate requirements for an already fill-rate limited algorithm. Additionally, many light samples are necessary to avoid banding artifacts.

Brebion [2003] generates soft shadows by rendering an inner and outer shadow volume using z-fail shadow volumes. The inner volume represents the umbra region and is stored in the red channel, while the outer volume represents the penumbra region and is stored in the green channel. Using multi-pass rendering, the red channel is blurred only where the green channel is non-zero. The outer shadow volume is created from an extended hull of the occluder, similar to Parker et al.'s [1998] technique. However, creating the outer volume for an arbitrary object is potentially expensive since it may require CSG operations to remove vertexes that self-penetrate the occluder when extended.

Akenine-Möller and Assarsson [2002] propose the *penumbra wedge* as a primitive to be used to model the penumbra region of a shadow. Penumbra wedges are four quadrilaterals generated from the light space silhouette edge of an occluder. Each wedge shares its side planes with neighboring wedges and empirically models the visibility of the light for each point contained within the wedge. Penumbra wedges are rendered to the camera space image of the scene in a fashion similar to shadow volumes. The authors later improve upon their technique (see Assarsson and Akenine-Möller [2003]) by introducing a new wedge construction algorithm that avoids the creation of degenerate wedges and removes the dependency of each wedge's side plane to be shared by its neighboring wedges. They also improve upon their visibility calculation by projecting the wedge's silhouette edge into a precomputed light space visibility lookup texture. This texture can be animated and can also encode the shape of the light source. Assarsson et al. [2003; 2004] provide further optimizations for the penumbra wedge algorithm.

Penumbra wedges suffer from visual artifacts when objects overlap in light space. The authors also note that the shadows generated by their technique are incorrect for large light sources due to occluders' silhouettes being computed in reference to the center of the light source. However, to overcome this single silhouette artifact, the authors propose breaking up large light sources into multiple lights. Like most shadow volume algorithms, penumbra wedges are fill-rate limited and therefore limit their ability to render dynamic, complex scenes in real-time.

Jakobsen et al. [2004] proposes a method to overcome the single silhouette artifact from which penumbra wedges suffer. They proceed by rendering penumbra and umbra shadow volumes into a light space image called the *D-buffer*. The D-buffer stores several distances, which are later used to compute visibility. A disk light source is assumed, which allows the authors to accurately find silhouette edges with respect to the entire light source, not just the center point. Artifacts occur when objects overlap in light space and also when silhouette corners form acute angles. The authors also point out that their algorithm is unable to render correct shadows for umbra regions that are completely diminished.

The above review summarizes previous work that is most relevant to our own. For a comprehensive overview on shadow generation techniques, see the work of Woo et al. [1990], Möller and Haines [1999], and Hasenfratz et al. [2003].

Chapter 4

Penumbra Volume Maps

To generate perceptually correct soft shadows at real-time rates, we propose *penumbra volume mapping*, or PVM. PVM is a hybrid technique that borrows key approaches from both shadow volumes and shadow maps and combines them in a unique manner. PVM is similar to shadow volumes in that penumbra volumes are extruded from silhouette edges. Unique to PVM, occluder information is stored in each volume represented as a light space image. No stencil buffer updates are used. The light space image representation of penumbra volume maps is inspired by traditional shadow maps, which are also used to determine umbra regions.

Penumbra volume maps are composed in three passes. In the first pass, a light space shadow map of the scene is generated. In the second pass, we compute and then render from the light's viewpoint penumbra volumes into a penumbra volume map. In the final pass, the scene is rendered from the camera's viewpoint, where we use the shadow map and penumbra volume maps to shadow the scene.

In the following sections, we describe the second and third passes, as the first pass is trivial. We conclude by discussing the pros and cons of the method.

4.1 Penumbra Volume Map Creation

At the core of our algorithm is the penumbra volume map. For each point in the light's view frustum, this map stores the light space silhouette edge of an occluder that may be partially occluding the light. In the following section, we decompose the creation of the penumbra volume map into three steps: silhouette detection, volume generation, and volume rendering.

4.1.1 Silhouette Detection

We begin the creation of the penumbra volume map by first computing the light space silhouette of each object in the scene. A modified winged-edge data structure [Baumgart 1975] is used to store the connectivity of each mesh. For each edge in the mesh, this data structure stores each face of which the edge is a member. Additionally, given an edge, we are able to determine the neighboring edges that share either of the given edge's vertexes. We assume that each object in the scene is described by a closed two-manifold mesh. This

leads to a simple silhouette detection algorithm. For each edge, we classify the edge as a silhouette edge if one of the edge's faces is front facing with respect to the light, while the other face is back facing with respect to the light. This case is easily detected by taking each face's geometric face normal and dotting it with a vector to the center of the light. If the signs of the two dot products differ, the edge is a silhouette edge.

The silhouette detection algorithm described above is a brute force algorithm, i.e., each edge in the mesh is tested. However, the silhouette does not need to be recomputed every frame. Rather, the silhouette needs to be recomputed only when the light or object moves in relation to each other (within a tolerance) or the geometry of the object is manipulated. Also note that in many applications the connectivity of meshes does not change across frames. In this case, the winged-edge data structure need only be computed once.

4.1.2 Penumbra Volume Creation

Once light space silhouette edges have been computed, we concern ourselves with extruding the silhouette edges to model the penumbra volume. Given a silhouette vertex $\mathbf{v}_1$, we use silhouette connectivity information to determine two neighboring vertexes, $\mathbf{v}_0$ and $\mathbf{v}_2$, that are also in the light space silhouette. Together, these three vertexes define two silhouette edges, $\mathbf{e}_0$ and $\mathbf{e}_1$. We extrude $\mathbf{v}_1$ by first computing the normal of $\mathbf{e}_1$'s hard shadow plane with:

$$\mathbf{n}_1 = \frac{(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_1 - \mathbf{l})}{\|(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_1 - \mathbf{l})\|},$$

where $\mathbf{l}$ is the object space position of the light source. We then compute two displaced light positions, $\mathbf{d}_1^-$ and $\mathbf{d}_1^+$, the outer and inner light positions, respectively,

$$\mathbf{d}_1^- = \mathbf{l} - \mathbf{n}_1 r, \quad \text{and} \quad \mathbf{d}_1^+ = \mathbf{l} + \mathbf{n}_1 r,$$

where r is the radius of the light source.

The points $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{d}_1^-$ form a plane describing the bounds of $\mathbf{e}_1$'s outer penumbra volume, with normal:

$$\mathbf{n}_1^- = \frac{(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_1 - \mathbf{d}_1^-)}{\|(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_1 - \mathbf{d}_1^-)\|}.$$

Likewise, $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{d}_1^+$ form a plane describing the bounds of $\mathbf{e}_1$'s inner penumbra volume. See Figure 4.1 (left and center).

In a similar fashion, we compute the inner and outer penumbra volume planes for $\mathbf{e}_0$. With these planes, we can then compute the inner and outer extrusion vectors for $\mathbf{v}_1$. To compute $\mathbf{v}_1$'s outer extrusion vector, $\hat{\mathbf{v}}_1^-$,

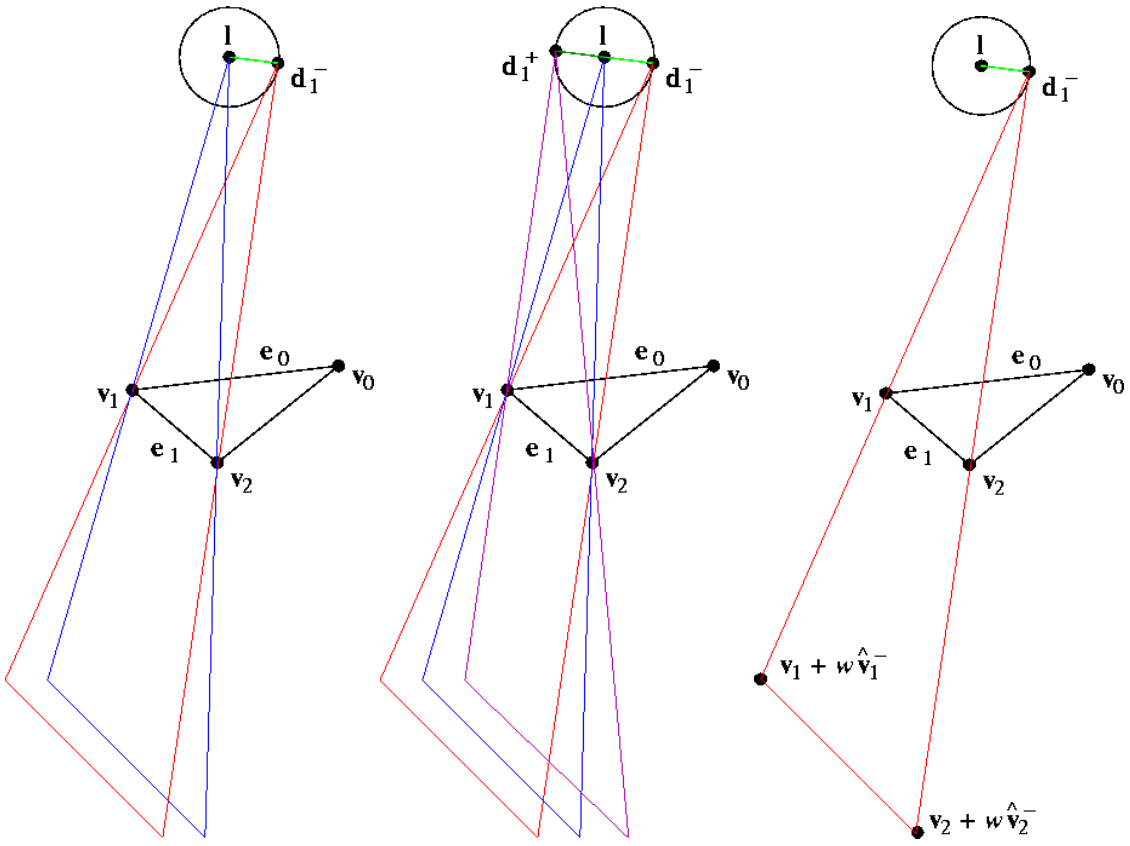

Figure 4.1: Left: Outer penumbra volume plane. **Center:** Inner penumbra volume plane added. **Right:** Outer penumbra volume quadrilateral.

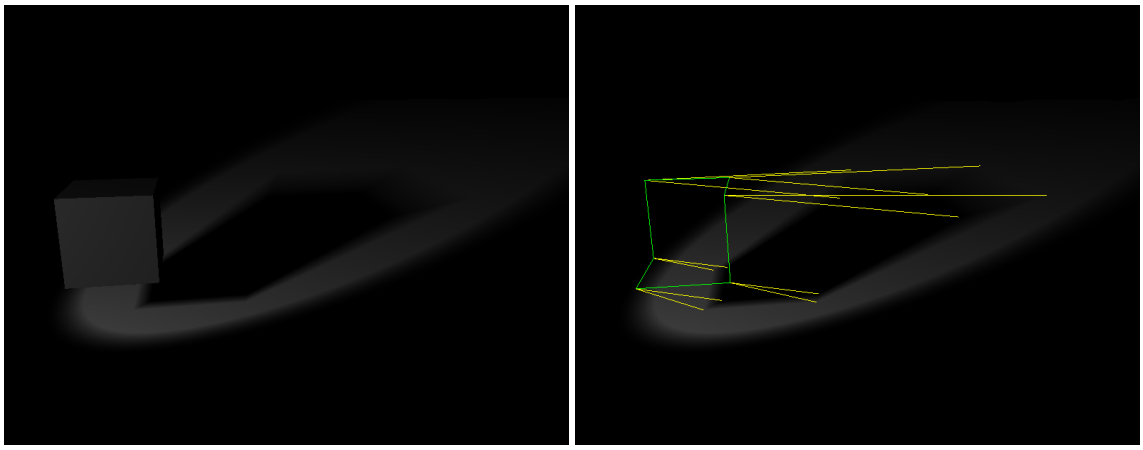

Figure 4.2: Left: A shadowed scene. **Right:** Visualization of the inner and outer extrusion vectors for each silhouette vertex (yellow). The occluder's silhouette is shown in green.

we intersect $\mathbf{e}_0$'s and $\mathbf{e}_1$'s outer penumbra volume planes. Since these two planes share a point, $\mathbf{v}_1$, and with edge $\mathbf{e}_0$'s outer penumbra volume plane defined by $\mathbf{n}_0^-$,

$$\mathbf{n}_0^- = \frac{(\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_0 - \mathbf{d}_0^-)}{\|(\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_0 - \mathbf{d}_0^-)\|},$$

this intersection is computed simply with:

$$\hat{\mathbf{v}}_1^- = \mathbf{n}_1^- \times \mathbf{n}_0^-.$$

The extrusion vector $\hat{\mathbf{v}}_1^-$ must be negated when $\mathbf{v}_1$ is in the negative halfspace of the plane described by $\mathbf{v}_0$, $\mathbf{v}_2$, and $\mathbf{l}$. If $\mathbf{n}_0$ and $\mathbf{n}_1$ are equal, the planes are coincident, in which case $\hat{\mathbf{v}}_1^-$ is given by:

$$\hat{\mathbf{v}}_1^- = \frac{\mathbf{v}_1 - \mathbf{d}_1^-}{\|\mathbf{v}_1 - \mathbf{d}_1^-\|}.$$

Vertex $\mathbf{v}_1$'s inner extrusion vector, $\hat{\mathbf{v}}_1^+$, is computed analogously.

The process above is repeated for each vertex in the silhouette. As a result, two quadrilaterals are produced for each edge, one describing the inner penumbra region, the other describing the outer penumbra region. For example, $\mathbf{e}_1$'s outer volume quadrilateral is defined by $\mathbf{v}_1$, $\mathbf{v}_1 + w\hat{\mathbf{v}}_1^-$, $\mathbf{v}_2 + w\hat{\mathbf{v}}_2^-$, and $\mathbf{v}_2$, where w is a scalar value that will place the extruded point either at or beyond the light's far clipping plane. See Figure 4.1 (right) shows a depiction of the outer volume quadrilateral and Figure 4.2 gives a visualization of the extruded volume vectors.

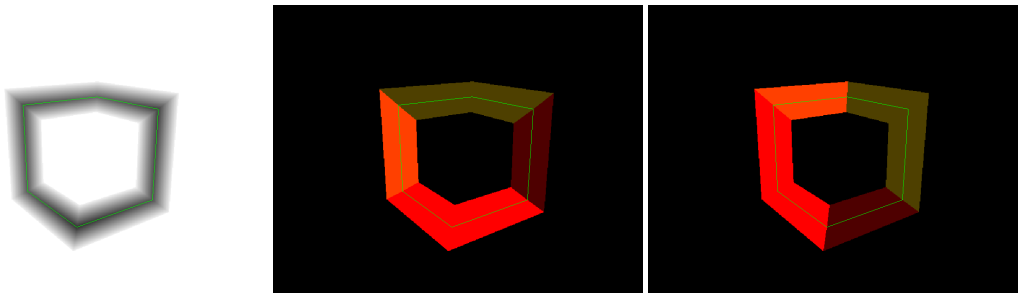

Figure 4.3: Layers of the penumbra volume map. The green lines represent the light space silhouette of the occluder. **Left:** The penumbra volume depth maps encodes the light space depth of each penumbra volume. The PVDM is used to determine if a point is contained within a penumbra volume. **Center/Right:** The penumbra volume edge maps. Each map encodes one of the end-points of a silhouette edge that gives rise to a volume. These points are used during the visibility computation to reconstruct occluder edges.

4.1.3 Penumbra Volume Rendering

Once the penumbra volume geometry has been generated, we render the volumes from the light space view of the scene. During this phase, three maps are constructed. The first map, the *penumbra volume depth map* (PVDM), stores the light space depth of each first-depth visible volume. The second and third maps, the *penumbra volume edge maps* (PVEM), each store one of the silhouette edge points that give rise to the volume. The PVEMs serve to identify the vertexes of the edge casting the penumbra shadow. These vertexes are then used in the third pass to construct the light cutting plane (see below). We treat the composition of all three maps as a multi-layered map called the *penumbra volume map* (PVM). Figure 4.3 illustrates the three maps.

The maps described above can be efficiently generated on modern graphics hardware. In our current implementation, we utilize the GPU's ability to render to multiple buffers simultaneously [ARB_ draw_ buffers] creating the three maps in a single pass. Each PVEM is composed of 16-bit floating point elements [ARB_ texture_ float], while the penumbra volume depth map is a 24-bit depth texture [ARB_ depth_ texture].

Rendering penumbra volumes is the second pass in a multi-pass algorithm. Since the results of this pass will be used in later passes, we make use of modern hardware's efficient render-to-texture capabilities [EXT_ framebuffer_ object].

4.2 Light Visibility

In the final pass we use the shadow map and penumbra volume map generated in the first and second passes to shade the scene from the camera's point of view. Listing 4.1 outlines pseudo-code for the final pass.

```
uniform sampler2D Edge0Map, Edge1Map, PVDM, ShadowMap;

vec4 shade(in vec3 p_pixel, // pixel's world space position
           in vec3 n_pixel, // pixel's world space normal
           in vec3 p_light, // light's world space position
           in vec2 t_proj, // pixel's projected shadow map coord.
           in float d_pixel) // pixel's light space depth

{
  // compute the incoming light from light p_light,
  // do not account for shadows (similar to OpenGL pipeline)
  vec4 incoming = computeIncomingLight(p_pixel,p_normal,p_light);

  // compute the light's visibility using penumbra volumes maps
  // (Edge0Map, Edge1Map, PVDM) and shadow map (ShadowMap)
  float shadowAtten =
    computeLightVisibility(p_pixel,p_light,t_proj,d_pixel);

  // modulate incoming light with light visibility term
  // add ambient lighting
  return = incoming * shadowAtten + ambient;
}
```

Listing 4.1: OpenGL Shading Language pseudo-code for computing irradiance at a camera pixel in the final pass. Lighting is first computed without accounting for light visibility, the result of which is modulated by an attenuation factor computed from the penumbra volume map.

```
uniform sampler2D Edge0Map, Edge1Map, PVDM, ShadowMap;

float computeLightVisibility(
        in vec3 p_pixel, // pixel's world space pos.
        in vec3 p_light, // light's world space pos.
        in vec2 t_proj, // pixel's projected sm coord.
        in float d_pixel) // pixel's light space depth

{
  vec3 shadowAtten = 1.0;

  // sample the volume depth map,
  // if depth test fails, p_pixel is in a penumbra volume
  float vd = texture2D(PVDM,t_proj);
  if( vd < d_pixel ) {
    // p_pixel is in a penumbra volume, estimate light visibility
    shadowAtten = intersectLight(p_pixel,p_light,t_proj);
  } else {
    // p_pixel is not in a penumbra volume, but still may be in an
    // umbra region. sample the shadow map, if the sample is
    // closer than the pixel's depth, the pixel is completely
    // shadowed.
    float sd = texture2D(ShadowMap,t_proj);
    if( sd < d_pixel ) {
      shadowAtten = 0.0;
    }
  }

  return shadowAtten;
}
```

Listing 4.2: OpenGL Shading Language pseudo-code for estimating light visibility at a camera pixel, $\mathbf{p}_{pixel}$. The code first determines if the pixel is in a penumbra volume, if so, the visibility of the light within the volume is estimated. If the pixel is not in a penumbra volume, a shadow map is sampled to determine if the pixel is in an umbra region.

We begin the final pass by rendering all scene geometry from the camera's viewpoint. Each rendered pixel is shaded by the pseudo-code in Listing 4.1. The key function in creating shadows in Listing 4.1 is `computeLightVisibility`. This function estimates the visibility of the light at pixel **p** (see pseudo-code in Listing 4.2).

The function `computeLightVisibility` begins by sampling the penumbra volume depth map. This sample is used to determine if the pixel being shaded is contained within a penumbra volume. If the pixel is deemed outside a penumbra volume, the shadow map generated in the first pass is sampled. If the shadow map sample is less than the pixel's light space depth, the pixel is known to be shadowed. Since we know

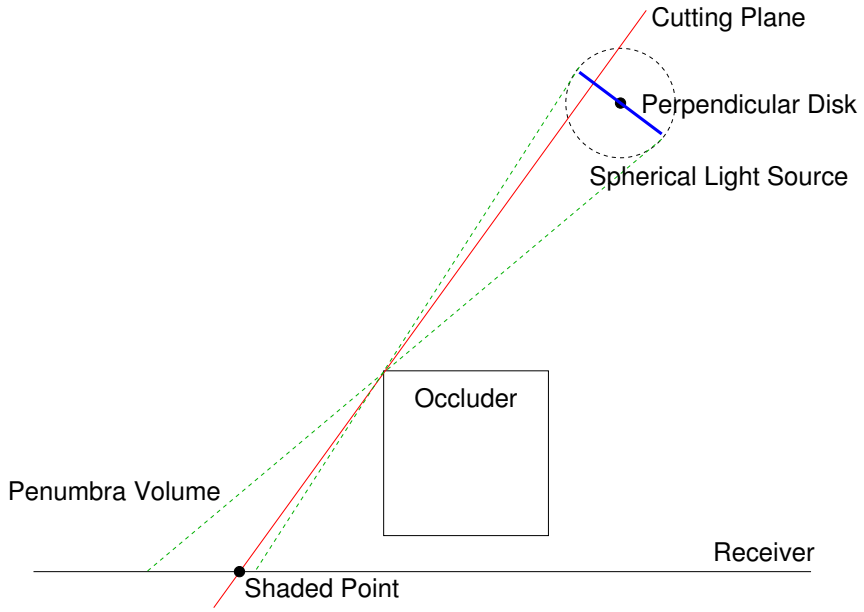

Figure 4.4: The cutting plane (red) is defined by the point being shaded and the occluder edge which generated the penumbra volume (green) in which the point resides. To compute light visibility, a disk (blue) perpendicular to the cutting plane is created.

from the PVDM sample that the pixel is not in a penumbra region, a pixel that is deemed shadowed by the shadow map test must be in the umbra region of a shadow. Accordingly, `computeLightVisibility` returns a shadow attenuation factor of 0.0.

If the initial PVDM test results in the pixel being marked in a penumbra volume, `computeLightVisibility` calls `intersectLight`, which estimates the visibility of the light source at the pixel being shaded. The function `intersectLight` begins by sampling the penumbra volume edge maps, *Edge0Map* and *Edge1Map*. These two samples define the two points, $\mathbf{h}_0$ and $\mathbf{h}_1$, of the silhouette edge that gave rise to the penumbra volume. Points $\mathbf{h}_0$, $\mathbf{h}_1$, and fragment position $\mathbf{p}$ form the *cutting plane*, $\mathbf{n} \cdot \mathbf{p} + d = 0$, where:

$$\mathbf{n} = \frac{(\mathbf{h}_1 - \mathbf{h}_0) \times (\mathbf{p} - \mathbf{h}_0)}{||(\mathbf{h}_1 - \mathbf{h}_0) \times (\mathbf{p} - \mathbf{h}_0)||}, \quad \text{and} \quad d = -\mathbf{n} \cdot \mathbf{h}_0.$$

To further our visibility calculation, we construct a disk that has the same radius as the light source, is centered at the light source's position, and is perpendicular to the cutting plane. We use the percentage of the area of the disk that is in the positive half-space of the cutting plane as an estimate of the visibility of the light. Figure 4.4 illustrates the relationships between the cutting plane, light source, and disk.

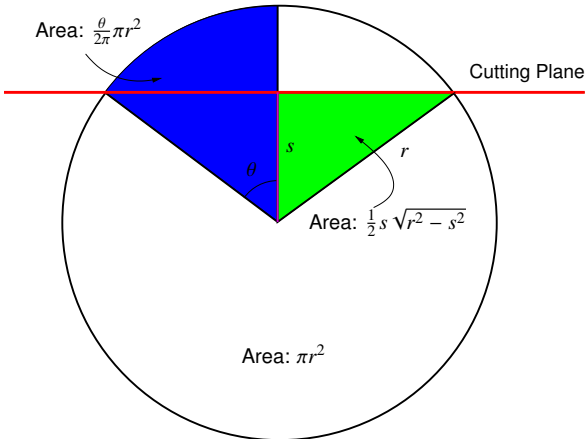

Figure 4.5: The area of the sector (blue) is subtracted from the area of the triangle (green) formed by the cutting plane (red), giving the white upper right area. The difference is then divided by the area of the circle to estimate the visibility of the light.

To determine the area of the disk that is in the positive half-space of the cutting plane, we compute the area of the sector formed by the normal of cutting plane and subtract the area of the triangle formed by the plane's intersection. Figure 4.5 illustrates the sector and triangle formed by the cutting plane.

We continue by computing the shortest distance, s, from the plane to the center of the light source with:

$$s = \mathbf{n} \cdot \mathbf{l} + d.$$

With s and the radius of the disk, r, we can then compute the area, a, of the light source in the positive half-space of the cutting plane with:

$$\begin{aligned} a &= \frac{\theta}{2\pi}\pi r^2 - \frac{1}{2}s\sqrt{r^2 - s^2} \\ &= \frac{\cos^{-1}(\frac{s}{r})}{2}r^2 - \frac{s\sqrt{r^2 - s^2}}{2}. \end{aligned} \tag{4.1}$$

Visibility can then be estimated with:

$$k = \frac{2a}{\pi r^2}. \tag{4.2}$$

Figure 4.6 shows an example result of Equation 4.2.

As we are modeling spherical light sources, Parker et al. [1998] note that the falloff for a diffuse spherical light source and occluders with straight edges is sinusoidal, not linear. We follow their suggestion of modeling

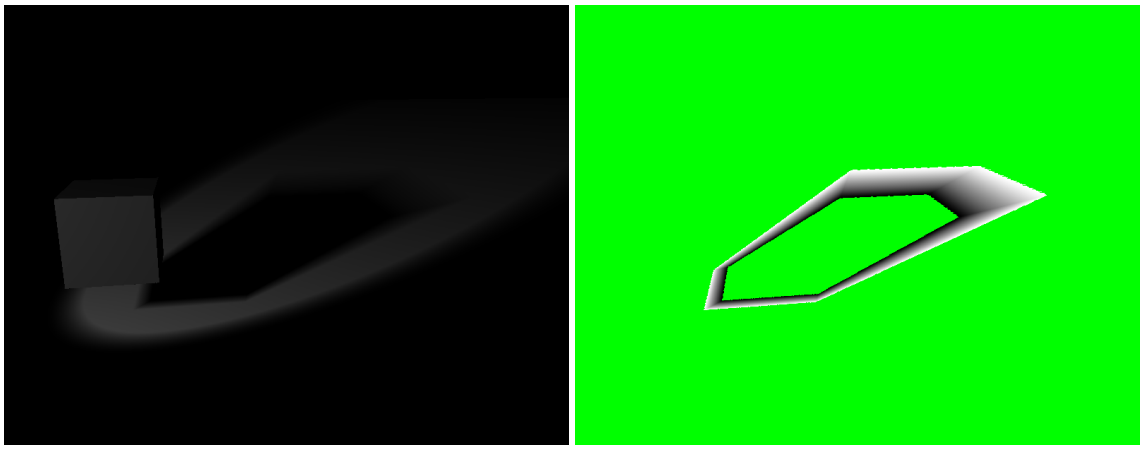

Figure 4.6: Left: A shadowed scene. **Right:** Visualization of the visibility computation executed for each pixel in the penumbra region.

this behavior with the Bernstein polynomial interpolant:

$$v = 3k^2 - 2k^3.$$

4.3 Penumbra Volume Mapping Summary

Penumbra volume maps provide several benefits over existing algorithms:

1. **Perceptually Correct Soft Shadows**. The soft shadows generated are perceptually correct, i.e., they account for the relative distances between light, occluder, and receiver. The penumbra volume edge maps allow us to determine the part of an occluder that is partially obstructing the light from a given point. With the known light position, point to be shaded, and the occluder edge, we are able to account for the distances between these three factors to produce realistic soft shadows.

2. **Mapping to Modern Graphics Hardware**. Real-time frame rates are achieved due in part to our algorithm's mapping to modern graphics hardware. Our current implementation uses newly developed render-to-texture APIs [EXT_ framebuffer_ object] to render multiple maps in a single pass. We also use modern hardware's floating point texture facilities [ARB_ texture_ float] to store occluder edges in penumbra volume edge maps at high precision. Finally, we use programmable graphics hardware [Rost 2004] to perform visibility on the GPU, thereby avoiding expensive GPU-to-CPU read-backs and freeing the CPU for other computations.

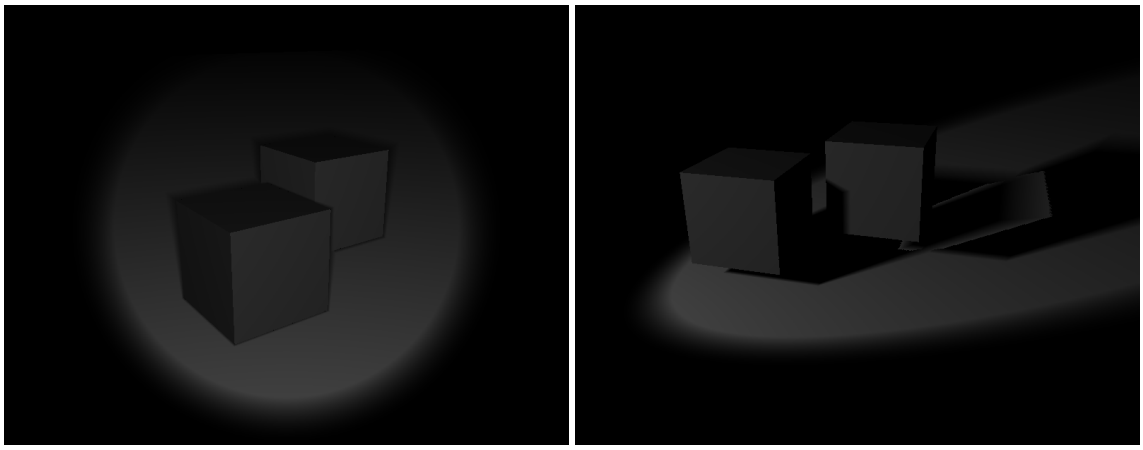

Figure 4.7: Left: Objects overlapping in light space look correct from the light's viewpoint. **Right:** However, as objects overlap in light space, penumbra volumes overwrite each others' edge information, which leads to artifacts when viewed from the camera.

3. **Camera Space Visibility Computation**. Using the light space PVDM, PVEMs, and shadow map, we are able to compute visibility in camera space on a pixel-by-pixel basis. The result is visually smooth shading within each penumbra volume generated from a single edge.

4. **Fast Visibility Computation**. The visibility calculation presented in Section 4.2 maps to few instructions on modern programmable GPUs. We also use recent GPU abilities to perform conditional branching to avoid computing light visibility outside of penumbra volumes.

Unfortunately, penumbra volume maps also suffer from the following limitations:

1. **Penumbra Volume Overlap**. As penumbra volumes overlap in light space, objectionable shadowing artifacts may occur. Figure 4.7 shows an example of one such artifact. These artifacts are a result of penumbra volumes overwriting each others' edge maps in light space, causing the visibility function of neighboring pixels to be evaluated by spatially differing occluder edges.

2. **Visibility Discontinuities**. While the visibility computed from a single edge is visibly smooth, the visibility across neighboring edges may be discontinuous. Figure 4.8 illustrates this discontinuity. These artifacts are quite noticeable, but can be somewhat hidden on textured receivers.

3. **CPU Silhouette Detection/Volume Construction**. Silhouette detection and volume construction take place entirely on the CPU. By performing these calculations on the CPU, we consume precious CPU time while not fully utilizing the GPU. McCool [2000] proposes a method to generate hard shadow volumes by finding depth discontinuities in a depth map. Unfortunately, this requires a read-back of

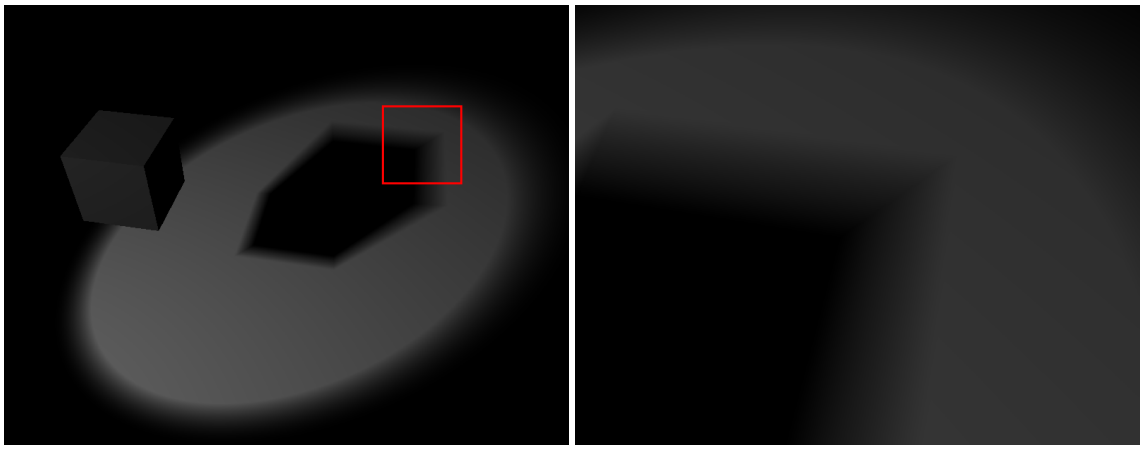

Figure 4.8: Discontinuities in the visibility computation across penumbra volumes produce visual artifacts.

the depth map from the GPU to the CPU, where the shadow volumes are constructed. While the current generation of graphics hardware is able to detect depth discontinuities in a depth map without a CPU read-back, modern GPUs are unable to generate new geometry (i.e., the shadow volumes created from the depth discontinuities) without CPU intervention. Upcoming generations of graphics hardware will support render-to-vertex-array, which will remove this restriction.

4. **Single Silhouette Artifact**. Like most other soft shadow methods, our algorithm may produce incorrect shadows due to computing occluder silhouettes with respect to the center of a light source, rather than accounting for each point on a light source. While this artifact is noticeable if the light source is close to, and is a great deal larger than an occluder, it can be minimized by breaking the light source into several sub-lights [Assarsson et al. 2003].

Chapter 5

Penumbra Volume Mask Filtering

In order to overcome the shortcomings of penumbra volume mapping, we propose a new method to generate perceptually correct soft shadows, *penumbra volume mask filtering*, or PVMF. In this method, light visibility is estimated by varying the size of a percentage-closer filter over a standard depth shadow map. This estimation is then blurred to hide artifacts. To avoid unnecessarily processing any areas of the scene not in shadow, modified penumbra volumes are used as a computation mask.

Penumbra volume mask filtering is a sampling based approach. The quality of the resulting shadows can be controlled simply by varying the number of samples taken. In practice relatively few samples are needed to avoid the visibility discontinuity and object overlap artifacts suffered by penumbra volume maps.

Penumbra volume mask filtering is closely related to the percentage-closer shadows algorithm [Fernando 2005], which also uses a variable sized percentage-closer filter to generate soft shadows. Unlike PCS, we use penumbra volume depth values to mask shadowed areas of the scene. This mask is used to restrict the regions where expensive filtering and sampling operations occur. Much like penumbra volume mapping, we use edge information encoded in each penumbra volume to estimate the size of each penumbra region. We then use this estimate to control the size of a percentage-closer filter. By using the penumbra volume's edge information, we avoid the PCS algorithm's occluder search, which accounts for 30% of the PCS algorithm's cost. Additionally, similarities to PCS are extended by application of jittered sampling and visibility blurring. These operations aid in eliminating many of the visual artifacts suffered by percentage-closer shadows.

Our algorithm is composed of five steps. In the first step, a standard depth shadow map is generated. In the second step, a penumbra volume mask is created and rendered from the light's viewpoint. Light visibility is computed into a visibility buffer in the third step. This buffer is blurred in the fourth step and used to modulate the scene as rendered from the camera in the final step.

In the following sections, we describe in detail the steps necessary to generate perceptually correct soft shadows with penumbra volume mask filtering. As with penumbra volume mapping, we do not discuss how to generate the standard depth shadow map in the first step, as generating this map is trivial. Once we have discussed the steps in the PVMF algorithm, we continue by detailing our implementation of the algorithm. Performance of the algorithm is then covered, followed by a discussion of the pros and cons of penumbra volume mask filtering.

5.1 Penumbra Volume Mask Creation

After creating a standard depth shadow map, we turn our attention to generating a penumbra volume mask. This mask will be used to limit the number of expensive filtering operations in future steps.

We proceed by finding the light space silhouette of all occluders in the scene. These silhouettes are then extruded from the light source to generate volumes which enclose the scene's penumbra regions. These volumes are rendered into a depth map along with the distance from the light source to the silhouette edge that gave rise to the penumbra volume. The penumbra volumes are used during the visibility estimation step as a mask to avoid processing pixels that are not in shadow. The light to silhouette edge distances are used during the visibility estimation step to control the size of the percentage-closer filter. We call the two channel map that stores the penumbra volume light space depth and the distance from each silhouette edge to the light source, the *penumbra volume mask*. In this section, we cover in detail the generation of the penumbra volume mask.

We begin the construction of the penumbra volume mask by first detecting the light space silhouette of each occluder in the scene. The silhouette of each occluder is found by using the algorithm outlined in Section 4.1.1. Much like Section 4.1.2, we then extrude the light space silhouette from the light source to create penumbra volumes. However, we do not use the extrusion algorithm in Section 4.1.2 to generate the volumes. Instead, we use a simplified volume construction algorithm that allows each silhouette edge's volume to be constructed independently of other silhouette edges. The volumes generated by our new algorithm are an overestimation of the volumes generated in Section 4.1.2.

Given two silhouette edge points $\mathbf{e}_0$ and $\mathbf{e}_1$ describing a single edge, the edge's penumbra volume is created by first moving the points an equivalent distance away from the center of the light source:

$$\begin{aligned} \hat{\mathbf{e}}_0 &= \mathbf{e}_0 - \min(\|\mathbf{e}_0 - \mathbf{l}\|, \|\mathbf{e}_1 - \mathbf{l}\|)\frac{\mathbf{e}_0 - \mathbf{l}}{\|\mathbf{e}_0 - \mathbf{l}\|}s_e \\ \hat{\mathbf{e}}_1 &= \mathbf{e}_1 - \min(\|\mathbf{e}_0 - \mathbf{l}\|, \|\mathbf{e}_1 - \mathbf{l}\|)\frac{\mathbf{e}_1 - \mathbf{l}}{\|\mathbf{e}_1 - \mathbf{l}\|}s_e \end{aligned}$$

where $\mathbf{l}$ is the object space position of the light source and s_e is a value in the range $(0, 1]$ used to overestimate the size of the penumbra volume. This overestimation helps us to account for sampling errors during the visibility estimation in Section 5.2.

With the two moved edge points, we can then find the edge's hard shadow plane normal with:

$$\begin{aligned} \mathbf{e} &= \frac{\hat{\mathbf{e}}_1 - \hat{\mathbf{e}}_0}{||\hat{\mathbf{e}}_1 - \hat{\mathbf{e}}_0||} \\ \mathbf{v} &= \frac{\hat{\mathbf{e}}_0 - \mathbf{l}}{||\hat{\mathbf{e}}_0 - \mathbf{l}||} \\ \mathbf{n} &= \mathbf{v} \times \mathbf{e} \end{aligned}$$

We can then create four extrusion vectors with:

$$\begin{aligned} \mathbf{v}_0^- &= \frac{\hat{\mathbf{e}}_0 - \mathbf{l} + \mathbf{e} - \mathbf{n}}{||\hat{\mathbf{e}}_0 - \mathbf{l} + \mathbf{e} - \mathbf{n}||} \\ \mathbf{v}_1^- &= \frac{\hat{\mathbf{e}}_1 - \mathbf{l} - \mathbf{e} - \mathbf{n}}{||\hat{\mathbf{e}}_1 - \mathbf{l} - \mathbf{e} - \mathbf{n}||} \\ \mathbf{v}_0^+ &= \frac{\hat{\mathbf{e}}_0 - \mathbf{l} + \mathbf{e} + \mathbf{n}}{||\hat{\mathbf{e}}_0 - \mathbf{l} + \mathbf{e} + \mathbf{n}||} \\ \mathbf{v}_1^+ &= \frac{\hat{\mathbf{e}}_1 - \mathbf{l} - \mathbf{e} + \mathbf{n}}{||\hat{\mathbf{e}}_1 - \mathbf{l} - \mathbf{e} + \mathbf{n}||} \end{aligned}$$

These vectors define both the outer penumbra volume quadrilateral: $\hat{\mathbf{e}}_0$, $\hat{\mathbf{e}}_0 + w\hat{\mathbf{v}}_0^-$, $\hat{\mathbf{e}}_1 + w\hat{\mathbf{v}}_1^-$, $\hat{\mathbf{e}}_1$ and the inner penumbra volume quadrilateral: $\hat{\mathbf{e}}_1$, $\hat{\mathbf{e}}_1 + w\hat{\mathbf{v}}_1^+$, $\hat{\mathbf{e}}_0 + w\hat{\mathbf{v}}_0^+$, $\hat{\mathbf{e}}_0$ where w is a scalar value that will place the extruded point either at or beyond the light's far clipping plane. Figure 5.1 illustrates the volume quadrilaterals created by the construction algorithm.

As a final step in the penumbra volume mask generation process, these quadrilaterals are rendered to a depth texture (see Figure 5.2). Along with each volume, the distance from the light source to the edge that gave rise to the volume:

$$d_e = \min(||\mathbf{e}_0 - \mathbf{l}||, ||\mathbf{e}_1 - \mathbf{l}||)$$

is also stored in a separate floating point channel.

This volume construction algorithm differs from the construction algorithm in 4.1.2. When constructing a volume from a silhouette edge, neighboring edge information is not needed. The PVM volume construction algorithm requires that neighboring edges' volume planes be clipped against each other. This clipping is to avoid volumes from the same silhouette loop from overwriting each other's edge information in light space,

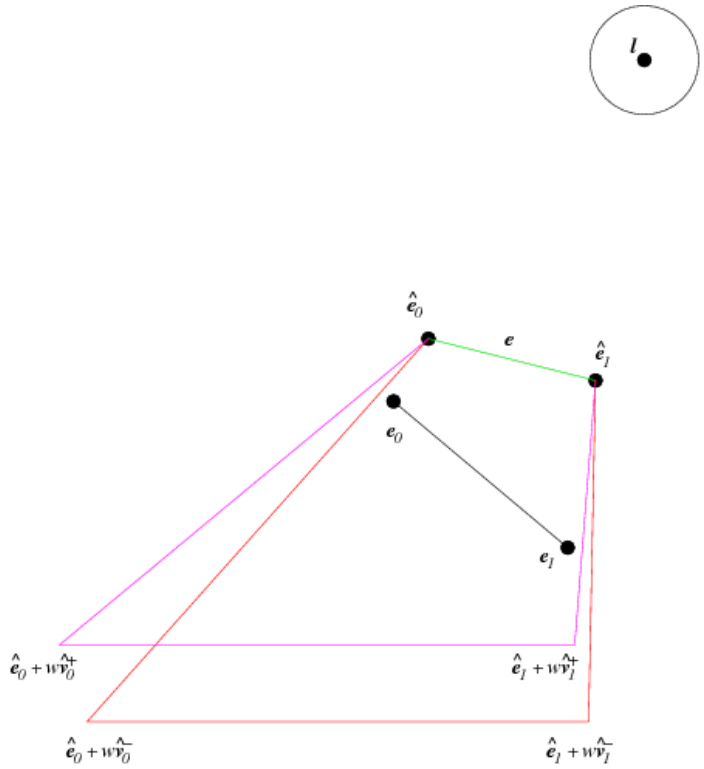

Figure 5.1: Overview of penumbra volume mask construction. The silhouette edge points, $\mathbf{e}_0$ and $\mathbf{e}_1$, are moved an equivalent distance away from the center of the light source, $\mathbf{l}$. These new points, $\hat{\mathbf{e}}_0$ and $\hat{\mathbf{e}}_1$ are extruded away from the light source to form the volume's quadrilaterals.

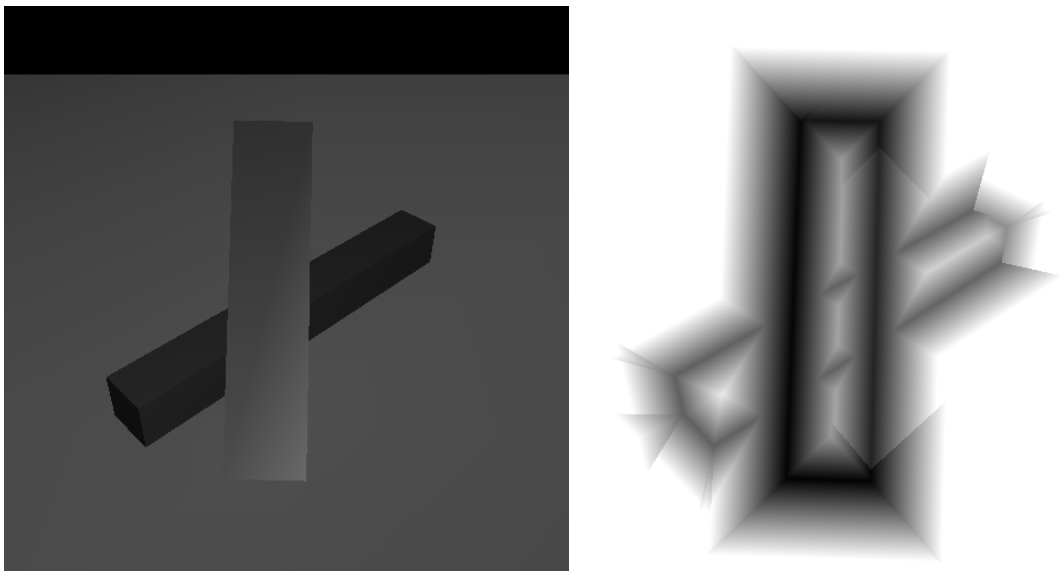

Figure 5.2: Left: The scene from the light's point of view. **Right:** The penumbra volume mask's depth value as seen from the light's viewpoint.

exacerbating object overlap artifacts. In contrast, overwriting penumbra volumes in the penumbra volume mask is not a problem, as the volumes are used to mask regions of the scene that may be in shadow.

5.2 Visibility Estimation

In this section, we discuss how the standard depth shadow map created in step 1 and the penumbra volume mask created in step 2, are used to estimate the visibility of the light source for each camera visible pixel into a floating point visibility buffer. We begin the visibility computation by first setting the size of a percentage-closer filter. The size of the filter is based on the relative distances between light, occluder, and receiver. This variable sized filter is used to sample the shadow map, the result of which is the visibility estimate. Using a mask to control filtering is similar to Duchowski et al.'s [2004] use of degradation masks to control sampling into pre-filtered textures to simulate human visual system deficiencies.

We begin our visibility estimate discussion by detailing how to set the size of the percentage closer filter. We finish by covering the sampling scheme used when applying the filter to the depth shadow map.

The strategy behind our visibility estimate is to vary the size of a percentage closer filter based on the relative distances between light, occluder, and receiver. Note that the filtered result of a percentage closer filter tends to increase as the size of the filter increases. With this in mind, we set the size of the percentage closer filter based on the projection of the estimated size of the penumbra at the pixel being shaded into the light's shadow map:

$$w_{pcf} = s_p \frac{(d_r - d_o)r}{2d_o d_r \tan\frac{\alpha}{2}}$$

s_p is a user defined scaling factor used to control the size of the penumbra. d_r is the light space depth of the shadow receiver and r is the radius of the light source. α is the light's horizontal field of view angle. d_o is the average light space depth of occluder edges that may have generated penumbra volumes that enclose the receiver point. d_o is computed by averaging samples in a region around the receiver point's projection, contained in the silhouette edge distance channel of the penumbra volume mask.

With the size of the filter set, we then proceed to apply the percentage-closer filter on all camera space pixels that are either enclosed within a penumbra volume (determined by sampling the penumbra volume mask's depth channel), or fail the depth test with respect to the standard depth shadow map generated in the first step. When applying the percentage closer filter, we break up the filter into an $n_{pcf} \times n_{pcf}$ cell grid where

Figure 5.3: Left: Banding artifacts occur when sampling the shadow map without jittered sampling. **Right:** Banding artifacts are effectively hidden with jittered sampling. While we reduce banding artifacts, we have artificially introduced noise.

each cell is $\frac{w_{pcf}}{n_{pcf}}$ square. n_{pcf} is a user defined variable which controls the number of percentage-closer filter samples. When sampling each cell, we randomly jitter the sample. As figure 5.3 illustrates, this jittering effectively reduces banding artifacts at the cost of introducing noise.

5.3 Visibility Estimate Noise Reduction

The visibility buffer resulting from the light visibility estimate purposely trades off banding artifacts in favor of noise. In the fourth step of our algorithm, we reduce noise in the visibility buffer by blurring the buffer. Our blur filter is described by:

$$v = \frac{1}{b_s} \sum_{i=-\frac{n_{blur}}{2}}^{\frac{n_{blur}}{2}} \sum_{j=-\frac{n_{blur}}{2}}^{\frac{n_{blur}}{2}} \mathrm{S}(u+i, v+j) \;\; \mathrm{B}(u, v, u+i, v+j)$$

Here n_{blur} is the number of samples to be taken by the blur filter. u, v is the texel coordinate of the current visibility buffer texel being blurred. *S(s,t)* is a function which samples the visibility texture at s, t. B is a

Figure 5.4: Left: Visibility buffer before blurring. **Right:** By blurring the visibility buffer we can effectively hide noise artifacts.

binary function which maps to:

$$\mathrm{B}(u, v, s, t) = \begin{cases} 0 & \text{if } \mathrm{ID}(u, v) \neq \mathrm{ID}(s, t) \\ 1 & \text{if } \mathrm{ID}(u, v) = \mathrm{ID}(s, t) \end{cases}$$

where *ID(s, t)* returns an unique integer assigned to each object in the scene. Finally, b_s is the number of samples where B returns true.

B is critical to correctly blurring the visibility buffer. In B's absence, the buffer is uniformly blurred, which results in shadows "bleeding" onto fully lit objects. To avoid this, B is used to restrict blur samples to texels which lie on the same object as the texel at u, v. For complex objects, assigning a single ID to each object may still lead to incorrect "bleeding" within the object. To avoid incorrect inter-object bleeding, a unique ID can be assigned per bone, an ID assignment technique similar in spirit to Vlachos et al. [2001].

The result of the blur step is a new visibility buffer that is relatively noise free. Figure 5.4 illustrates the visibility buffer before and after the blur step.

Figure 5.5: Left: Illumination from a point light source. **Center:** The visibility buffer. **Right:** By modulating the visibility buffer with illumination from a point light source, we produce the final shaded scene.

5.4 Final Compositing

In the final step of our algorithm, we render the scene from the camera's viewpoint. In this step, we use the center of the area light source as a point light source. For each pixel rendered from the camera, the pixel's corresponding visibility buffer texel, v, is sampled and multiplied by the irradiance due the point light source with Equation 2.2. Figure 5.5 illustrates the result of modulating the visibility buffer with irradiance from the point light source.

5.5 Results

In this section, we discuss our implementation of the penumbra volume mask filtering algorithm and provide results for samples scenes. We implemented and tested the algorithm on an AMD Athlon 3200+ CPU with a NVidia GeForce 6800 Ultra GPU utilizing NVidia's 76.76 GNU/Linux OpenGL 2.0 drivers.

Our implementation of the algorithm is composed of six passes per light. Figure 5.6 illustrates the passes. In our first pass, we store the scene's depth from the light's viewpoint. During this pass we enable front face culling to simulate second-depth shadow mapping. In the second pass, we generate and render penumbra volumes, storing their depth and the distance from the volumes' silhouette edges to the center of the light source.

In the third pass, we render from the camera. For each camera pixel we estimate the light's visibility using the shadow map, penumbra volume depth map, and penumbra volume silhouette edge distance map generated in the first two passes. This visibility is stored in the red channel of the visibility buffer. In the green channel of the visibility buffer, we store a unique ID for each object rendered. This ID is passed as

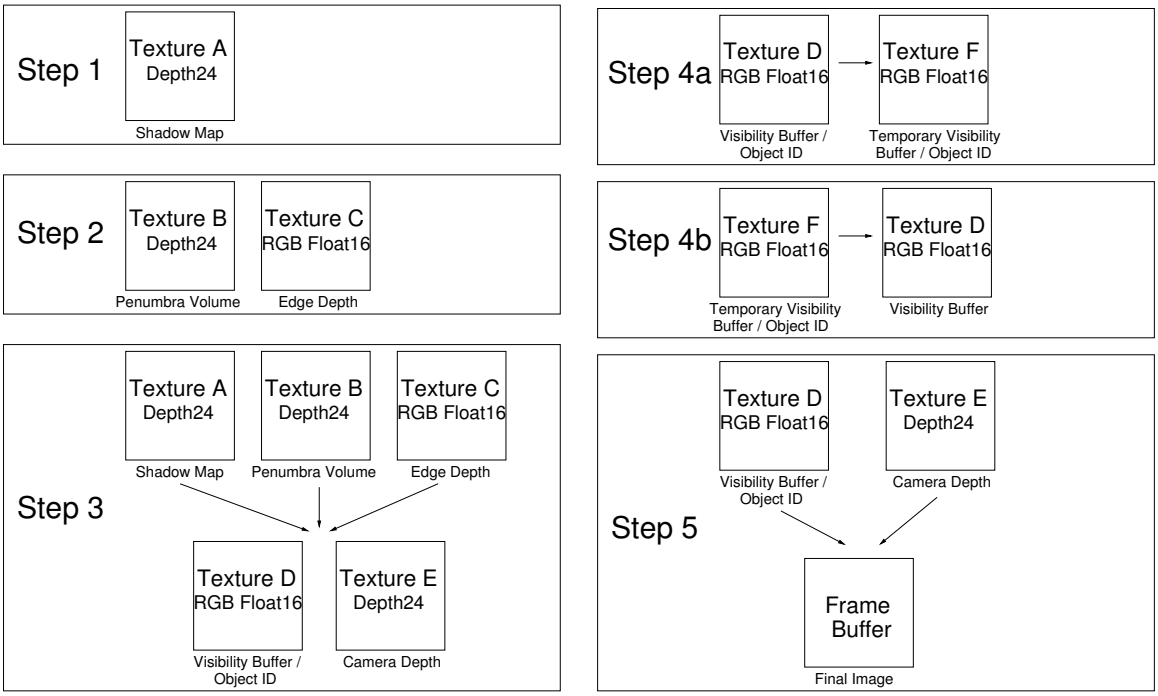

Figure 5.6: An illustration of the render textures used during the penumbra volume mask filtering algorithm. Step 1 produces a standard depth shadow map (Texture A) while step 2 renders penumbra volumes to a depth texture (Texture B) and silhouette edge depths to a floating point texture (Texture C). Step 3 uses the textures created in the first 2 passes to estimate light visibility for camera visible pixels and stores the result into the visibility buffer (Texture D). The fourth step first horizontally blurs the visibility buffer (Texture D), storing the result in a temporary floating texture (Texture F) and then blurs this temporary texture vertically, storing the result back into the visibility buffer (Texture D). In the fifth step the scene is rendered from the camera into the framebuffer, where the visibility buffer is used to modulate the scene.

uniform data to the shader for each object. In the blue channel, we store a mask derived from the penumbra volume mask and depth shadow map denoting whether or not the visibility buffer pixel should be blurred. Also in the third pass, we render the depth of the scene from the camera's viewpoint. This depth texture can be reused across lights for fast z-culling [Morein 2000].

When sampling from the depth shadow map, we enable NVidia's hardware accelerated percentage-closer filtering. Enabling hardware accelerated filtering improves overall picture quality.

The fourth and fifth passes (step 4 in Section 5.3) blur the visibility buffer to remove noise. In the fourth pass we blur the visibility buffer horizontally, storing the result in a temporary floating point texture. We finish the blur in the fifth pass by blurring this temporary buffer vertically, storing the result back into the visibility buffer. We break the blurring step into multiple passes to better take advantage of the GPU's texture cache.

In the final pass, we render the scene from the camera's viewpoint, accounting for illumination from the center of the light source. As each camera fragment is rendered, we modulate the fragment with its corresponding sample from the visibility buffer. The result is the scene rendered with shadows.

Figure 5.7 illustrates a scene rendered with penumbra volume mask filtering. All texture maps are 512×512 pixels with $n_{blur} = 12$ and $n_{pcf} = 7$. The scene renders between 27 and 150 frames per second. This large disparity is due to the use of the penumbra volume masks. If the camera's viewport contains many fragments which are masked by the penumbra volume mask, the frame rate decreases, since more fragments need to be filtered. As fewer fragments are masked by the penumbra volume mask (such as when the camera zooms out), the frame rate increases since fewer fragments require shadow processing.

Due to current hardware/driver restriction, texture memory usage in our implementation is inefficient. The silhouette edge depth texture (Texture C) need only be composed of a single floating point channel, not three. Likewise, the visibility buffers (Textures D and F) only need a single floating point channel. Object ID's can be stored in a single 8-bit channel while blur masks can be stored in a 1 bit stencil texture. Unfortunately, the graphics driver used in our implementation does not support rendering to fully featured 1 and 2 channel textures. Likewise, the current driver does not support rendering to 1-bit stencil textures. We expect these issues will be addressed in future drivers.

Artifacts occur in our implementation due to our use of second-depth shadow mapping [Wang and Molnar 1994] . Second-depth shadow mapping removes depth aliasing artifacts by moving the artifacts from the first-depth surface to the second-depth surface. Generally, these second-depth artifacts are hidden by the illumination model. However, artifacts may still occur at the intersection of first- and second-depth surfaces.

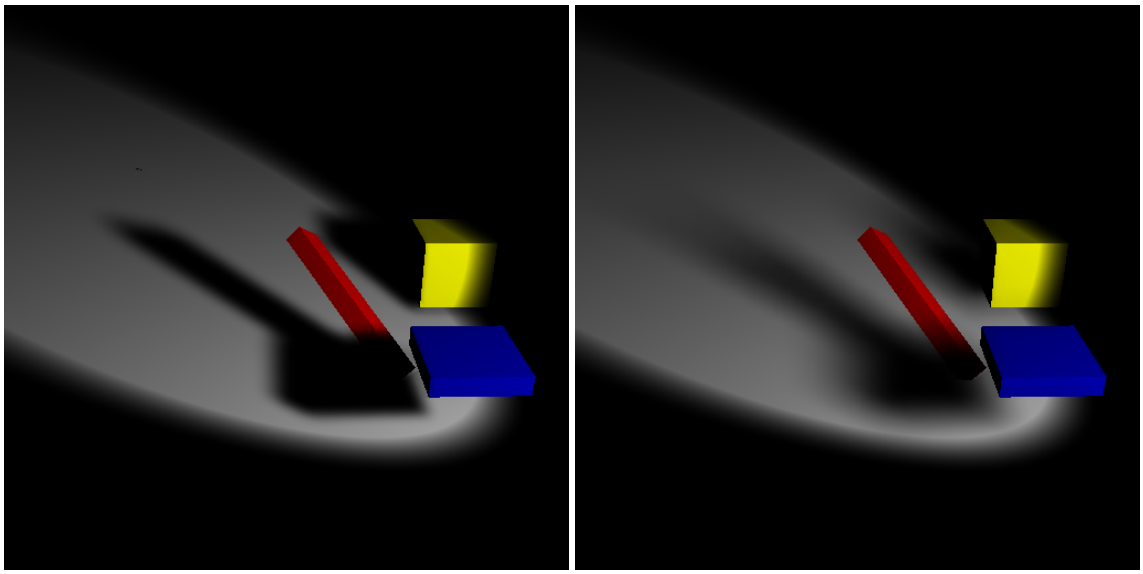

Figure 5.7: Left: A sample scene with the radius of the light source set to 0.3. **Right:** The same scene with the radius of the light source set to 1.0. As the relative distances between the occluder, receiver, and light source increases, the penumbra region grows and the umbra region diminishes in a perceptually correct manner.

These intersection artifacts are magnified by our implementation of penumbra volume mask filtering, due to sampling from an area of the shadow map, rather than a single texel.

In the future, we would like to explore using mid-point depth shadows [Woo 1992]. Unfortunately, our current hardware is not capable of efficiently implementing mid-point shadows with hardware accelerated depth buffer filtering.

Like many other image-based shadowing techniques, our algorithm suffers from perspective aliasing. While the percentage-closer filter and blur step aid in hiding this artifact, perspective aliasing can manifest itself in the form of blocky, although soft, shadows. Any of the many techniques (for example, see Fernando et al. [2001], Stamminger and Drettakis [2002], Aila and Laine [2004], Martin and Tan [2004], Wimmer et al. [2004], and Arvo [2004]) designed to combat perspective aliasing can be combined with our technique to reduce perspective aliasing.

5.6 Penumbra Volume Mask Filtering Summary

Like penumbra volume mapping, penumbra volume mask filtering produces perceptually correct shadows at real-time frame rates by utilizing modern graphics hardware. Unlike PVM, penumbra volume mask filtering does not suffer from visibility discontinuity and object overlap artifacts. However, single silhouette artifacts

(as described in 4.3) can occur. Additionally, penumbra volume construction occurs on the CPU. In the future, we plan to explore using the GPU's vertex processing engine to construct penumbra volumes "on-the-fly" on the GPU.

Chapter 6

Conclusions

We have described two new methods to generate soft shadows, *penumbra volume mapping* and *penumbra volume mask filtering*. The shadows generated by both methods are perceptually correct and can be evaluated and displayed at real-time rates. With penumbra volume mapping, we introduced a soft shadow algorithm that maps to few shader instructions. To overcome artifacts suffered by penumbra volume mapping, we introduced penumbra volume mask filtering. We have also provided an overview of related existing soft shadow algorithms. Likewise, we discussed the benefits and shortcomings of our proposed methods and outlined points for continued research.

Bibliography

AGRAWALA, M., RAMAMOORTHI, R., HEIRICH, A., AND MOLL, L. 2000. Efficient image-based methods for rendering soft shadows. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 375–384.

AILA, T. AND LAINE, S. 2004. Alias-free shadow maps. In *Proceedings of the EG Symposium on Rendering*. Springer Computer Science. Eurographics, Eurographics Association.

AKENINE-MÖLLER, T. AND ASSARSSON, U. 2002. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Rendering Techniques '02 (Proceedings of the 13th EG Workshop on Rendering)*. Springer Computer Science. Eurographics, Eurographics Association, 297–306.

ARB_DEPTH_TEXTURE. 2004. OpenGL Architecture Review Board. In *OpenGL Extension Registry*. SGI. URL: http://oss.sgi.com/projects/ogl-sample/registry/ARB/depth_texture.txt.

ARB_DRAW_BUFFERS. 2004. OpenGL Architecture Review Board. In *OpenGL Extension Registry*. SGI. URL: http://oss.sgi.com/projects/ogl-sample/registry/ARB/draw_buffers.txt.

ARB_SHADOW. 2002. OpenGL Architecture Review Board. In *OpenGL Extension Registry*. SGI. URL: http://oss.sgi.com/projects/ogl-sample/registry/ARB/shadow.txt.

ARB_TEXTURE_FLOAT. 2004. OpenGL Architecture Review Board. In *OpenGL Extension Registry*. SGI. URL: http://oss.sgi.com/projects/ogl-sample/registry/ARB/texture_float.txt.

ARVO, J. 2004. Tiled shadow maps. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*. IEEE Computer Society, Washington, DC, USA, 240–247.

ARVO, J., HIRVIKORPI, M., AND TYYSTJÄRVI, J. 2004. Approximate soft shadows using image-space flood-fill algorithm. *Computer Graphics Forum 23,* 3, 271–280.

ARVO, J. AND WESTERHOLM, J. 2004. Hardware accelerated soft shadows using penumbra quads. *Journal of WSCG 12,* 1, 11–18.

ASSARSSON, U. AND AKENINE-MÖLLER, T. 2003. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*.

ASSARSSON, U. AND AKENINE-MÖLLER, T. 2004. Occlusion culling and z-fail for soft shadow volume algorithms. *The Visual Computer 20,* 8-9, 601–612.

ASSARSSON, U., DOUGHERTY, M., MOUNIER, M., AND AKENINE-MÖLLER, T. 2003. An optimized soft shadow volume algorithm with real-time performance. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*. ACM Press.

BAUMGART, B. G. 1975. A polyhedron representation for computer vision. In *Proceedings of the AFIPS National Conference*. Vol. 44. 589–596.

BILODEAU, B. AND SONGY, M. 1999. Real time shadows. In *Creativity 1999, Creative Labs Inc. sponsored game developer conferences*. Los Angleles, California and Surrey, England.

BRABEC, S. AND SEIDEL, H.-P. 2001. Hardware-accelerated rendering of antialiased shadows with shadow maps. In *CGI '01: Computer Graphics International 2001*. IEEE Computer Society, Washington, DC, USA, 209–214.

BRABEC, S. AND SEIDEL, H.-P. 2002. Single sample soft shadows using depth maps. In *Proceedings of Graphics Interface*.

Brabec, S. and Seidel, H.-P. 2003. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Proceedings of Eurographics '03) 25,* 3.

Brebion, F. 2003. ShaderX2: Shader programming tips & tricks with directx 9. Wordware Publishing, Chapter Soft Shadows, 559–579.

Brotman, L. S. and Badler, N. 1984. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications 4,* 10 (October), 5–24.

Carmack, J. 2000. E-mail to private list. URL: http://developer.nvidia.com/attach/6832.

Chan, E. and Durand, F. 2003. Rendering fake soft shadows with smoothies. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 208–218.

Chen, S. E. and Williams, L. 1993. View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 279–288.

Cohen, M. F., Wallace, J., and Hanrahan, P. 1993. *Radiosity and realistic image synthesis*. Academic Press Professional, Inc., San Diego, CA, USA.

Cohen-Or, D., Chrysanthou, Y., Silva, C. T., and Durand, F. 2002. A survey of visibility for walkthrough applications. *IEEE Transaction on Visualization and Computer Graphics*.

Crow, F. C. 1977. Shadow algorithms for computer graphics. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 242–248.

Dietrich, S. 2003. ShaderX2: Shader programming tips & tricks with directx 9. Wordware Publishing, Chapter Robust Object ID Shadows, 580–586.

Duchowski, A. T., Cournia, N., and Murphy, H. 2004. Gaze-contingent displays: A review. *CyberPsychology & Behavior 7,* 6, 621–634.

Everitt, C. 2001. Interactive order-independant transparency. Tech. rep., NVIDIA Corporation. Available at: http://developer.nvidia.com/.

Everitt, C. and Kilgard, M. 2002. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Tech. rep., NVIDIA Corporation. URL: http://developer.nvidia.com/.

EXT_framebuffer_object. 2005. OpenGL Architecture Review Board. In *OpenGL Extension Registry*. SGI. URL: http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt.

Fernando, R. 2005. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Conference, Sketches and Applications*. ACM Press, New York, NY, USA.

Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. P. 2001. Adaptive shadow maps. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 387–390.

Gooch, B., Sloan, P.-P. J., Gooch, A., Shirley, P., and Riesenfeld, R. 1999. Interactive technical illustration. In *SI3D '99: Proceedings of the 1999 Symposium on Interactive 3D Graphics*. ACM Press, New York, NY, USA, 31–38.

Haeberli, P. and Akeley, K. 1990. The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 309–318.

HAINES, E. 2001. Soft planar shadows using plateaus. *Journal Graphics Tools 6,* 1, 19–27.

HASENFRATZ, J.-M., LAPIERRE, M., HOLZSCHUCH, N., AND SILLION, F. 2003. A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22,* 4 (December), 753–774. State-of-the-Art Reviews.

HECKBERT, P. S. 1990. Adaptive radiosity textures for bidirectional ray tracing. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 145–154.

HECKBERT, P. S. AND HERF, M. 1997. Simulating soft shadows with graphics hardware. Tech. Rep. CMU-CS-97-104, Carnegie Mellon University. January.

HEIDMANN, T. 1991. Real shadows, real time. *Iris Universe 18*, 28–31. Silicon Graphics, Inc.

HEIDRICH, W., BRABEC, S., AND SEIDEL, H.-P. 2000. Soft shadow maps for linear lights. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*. Springer-Verlag, London, UK, 269–280.

HERF, M. AND HECKBERT, P. S. 1996. Fast soft shadows. In *SIGGRAPH '96: ACM SIGGRAPH 96 Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH '96*. ACM Press, New York, NY, USA, 145.

HOURCADE, J.-C. AND NICOLAS, A. 1985. Algorithms for antialiased cast shadows. *Computer and Graphics*, 259–265.

HU, H. H., GOOCH, A. A., CREEM-REGEHR, S. H., AND THOMPSON, W. B. 2002. Visual Cues for Perceiving Distances from Objects to Surfaces. *Presence: Teleoperators and Virtual Environments 11,* 6, 652–664.

IM, Y.-H., HAN, C.-Y., AND KIM, L.-S. 2005. A method to generate soft shadows using a layered depth image and warping. *IEEE Transactions on Visualization and Computer Graphics 11,* 3, 265–272.

JAKOBSEN, B., CHRISTENSEN, N. J., LARSEN, B. D., AND PETERSEN, K. S. 2004. Boundary correct real-time soft shadows. In *Computer Graphics International*. 232–239.

KEATING, B. AND MAX, N. 1999. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques '99 (Proceedings of the 10th EG Workshop on Rendering)*. Springer Computer Science. Eurographics, Eurographics Association, 205–220.

KELLER, A. 1997. Instant Radiosity. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 49–56.

KERSTEN, D., KNILL, D., MAMASSIAN, P., AND BÜLTHOFF, I. 1996. Illusory motion from shadows. *Nature 379,* 6560, 31.

KERSTEN, D., MAMASSIAN, P., AND KNILL, D. C. 1997. Moving cast shadows induce apparent motion in depth. *Perception 26*, 171–192.

KIM, T.-Y. AND NEUMANN, U. 2001. Opacity shadow maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. Springer-Verlag, London, UK, 177–182.

KIRSCH, F. AND DÖLLNER, J. 2003. Real-time soft shadows using a single light sample. *11,* 2, 255–262.

KOLLIG, T. AND KELLER, A. 2004. Illumination in the presence of weak singularities. In *Monte Carlo and Quasi-Monte Carlo Methods*, D. Talay and H. Niederreiter, Eds. Springer-Verlag, Berlin.

LAINE, S. 2005. Split-plane shadow volumes. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM Press, New York, NY, USA, 23–32.

LENGYEL, E. 2002. The mechanics of robust stencil shadows. *Gamasutra*. URL: http://www.gamasutra.com/features/20021011/lengyel_01.htm.

LEONARDO. 1490. *Codex Urbinas*.

LISCHINSKI, D. AND RAPPOPORT, A. 1998. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98, Proceedings of the Eurographics Workshop on Rendering*. 301–314.

LLOYD, B., WENDT, J., GOVINDARAJU, N. K., AND MANOCHA, D. 2004. CC shadow volumes. In *Proceedings of the 2nd EG Symposium on Rendering*. Springer Computer Science. Eurographics, Eurographics Association.

LOKOVIC, T. AND VEACH, E. 2000. Deep shadow maps. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 385–392.

MAMASSIAN, P., KNILL, D. C., AND KERSTEN, D. 1998. The perception of cast shadows. *Trends in Cognitive Sciences 2,* 8, 288–295.

MARTIN, T. AND TAN, T.-S. 2004. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the EG Symposium on Rendering*. Springer Computer Science. Eurographics, Eurographics Association.

MAX, N. 1996. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Proceedings of the eurographics workshop on Rendering techniques '96*. Springer-Verlag, London, UK, 165–174.

MCCOOL, M. D. 2000. Shadow volume reconstruction from depth maps. *ACM Trans. Graph. 19,* 1, 1–26.

MCGUIRE, M., HUGHES, J. F., EGAN, K., KILGARD, M., AND EVERITT, C. 2003. Fast, practical and robust shadows. Tech. rep., NVIDIA Corporation, Austin, TX. November. Available at: http://developer.nvidia.com/.

MÖLLER, T. AND HAINES, E. 1999. *Real-time rendering*. A. K. Peters, Ltd., Natick, MA, USA.

MONTRYM, J. S., BAUM, D. R., DIGNAM, D. L., AND MIGDAL, C. J. 1997. InfiniteReality: a real-time graphics system. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 293–302.

MOREIN, S. 2000. ATI radeon – HyperZ technology. In *SIGGRAPH/Eurographics Graphics Hardware Workshop 2000, Hot3D Proceedings*.

MYSZKOWSKI, K. AND KUNII, T. L. 1994. Texture mapping as an alternative for meshing during walkthrough animation. In *Fifth Eurographics Workshop on Rendering*. 375–388.

NI, R. AND BRAUNSTEIN, M. L. 2004. Perception of scene layout from optical contact, shadows and motion. *Perception 33,* 11, 1305–1318.

PARKER, S., SHIRLEY, P., AND SMITS, B. 1998. Single sample soft shadows. Tech. Rep. UUCS-98-019, University of Utah. Oct.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 283–291.

ROST, R. J. 2004. *OpenGL(R) Shading Language*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J., AND HAEBERLI, P. 1992. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 249–252.

SHADE, J., GORTLER, S., HE, L., AND SZELISKI, R. 1998. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 231–242.

ST-AMOUR, J.-F., PAQUETTE, E., AND POULIN, P. 2005. Soft shadows from extended light sources with penumbra deep shadow maps. In *Graphics Interface 2005*. 105–112.

STAMMINGER, M. AND DRETTAKIS, G. 2002. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 557–562.

VAN PERNIS, A. 2004. Global Diffuse Illumination for Image Sequences. Ph.D. thesis, Clemson Univeristy, Department of Computer Science, Clemson, SC 29634.

VLACHOS, A., GOSSELIN, D., AND MITCHELL, J. L. 2001. Game Programming Gems 2. Charles River Media, Chapter Self-Shadowing Characters, 421–423.

WANG, Y. AND MOLNAR, S. 1994. Second-depth shadow mapping. Tech. Rep. TR94-019, University of North Caronlina.

WESTOVER, L. 1990. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 367–376.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, USA, 270–274.

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Proceedings of the EG Symposium on Rendering*. Springer Computer Science. Eurographics, Eurographics Association.

WOO, A. 1992. Graphics gems III. Academic Press Professional, Inc., San Diego, CA, USA, Chapter VII.1: The shadow depth map revisited, 338–342.

WOO, A., POULIN, P., AND FOURNIER, A. 1990. A survey of shadow algorithms. *IEEE Comput. Graph. Appl. 10,* 6, 13–32.

WYMAN, C. AND HANSEN, C. 2003. Penumbra maps: Approximate soft shadows in real-time. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 202–207.

YING, Z., TANG, M., AND DONG, J. 2002. Soft shadow maps for area light by area approximation. In *Pacific Conference on Computer Graphics and Applications*. 442–443.

ZHANG, H. 1998. Forward shadow mapping. In *Rendering Techniques '98 (Proceedings of the 9th EG Workshop on Rendering)*. Springer Computer Science. Eurographics, Eurographics Association, 131–138.

ZIOMA, R. 2003. ShaderX2: Shader programming tips & tricks with directx 9. Wordware Publishing, Chapter Reverse Extruded Shadow Volumes, 587–593.