Shadow Rendering Techniques for Real-Time Applications

BY
Sean Shypula
Bachelor of Science, Computer Engineering
University of Florida May 2003

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the graduate studies program
of DigiPen Institute Of Technology
Redmond, Washington
United States of America

Fall
2008

Thesis Advisor: Xin Li

DIGIPEN INSTITUTE OF TECHNOLOGY

GRADUATE STUDY PROGRAM

DEFENSE OF THESIS

THE UNDERSIGNED VERIFY THAT THE FINAL ORAL DEFENSE OF THE

MASTER OF SCIENCE THESIS OF SEAN SHYPULA

HAS BEEN SUCCESSFULLY COMPLETED ON 12/05/2008

TITLE OF THESIS: SHADOW RENDERING TECHNIQUES FOR REAL-TIME APPLICATIONS

MAJOR FILED OF STUDY: COMPUTER SCIENCE.

COMMITTEE:

Xin Li, Chair	Gary Herron
Samir Abou-Samra	Antonie Boerkoel

APPROVED :

Xin Li date Graduate Program Director	Xin Li date Dean of Arts and Sciences
Samir Abou-Samra date Department of Computer Science	Claude Comair date President

INSTITUTE OF DIGIPEN INSTITUTE OF TECHNOLOGY

PROGRAM OF MASTER'S DEGREE

THESIS APPROVAL

DATE: 12/05/2008

BASED ON THE CANDIDATE'S SUCCESSFUL ORAL DEFENSE, IT IS RECOMMENDED THAT THE THESIS PREPARED BY

SEAN SHYPULA

ENTITLED

SHADOW RENDERING TECHNIQUES FOR REAL-TIME APPLICATIONS

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE FROM THE PROGRAM OF MASTER'S DEGREE AT DIGIPEN INSTITUTE OF TECHNOLOGY.

Dr. Xin Li
Thesis Advisory Committee Chair

Dr. Xin Li
Director of Graduate Study Program

Dr. Xin Li
Dean of Arts and Sciences

Table of Contents

Chapter 1: **Abstract**

Shadows, subtle though they may be, provide a number of visual cues as to the layout and orientation of a scene and its light sources that would otherwise be difficult or impossible to convey. Shadows, which play a vital role in computer graphics, have been a highly active area of research from the publication of the first seminal papers in the late 1970's all the way to the current day.

This research provides an overview of the techniques that have been developed for rendering shadows in real-time applications. The two most widely used techniques for rendering hard-edged shadows are presented and discussed in detail. Additionally, several approaches to rendering soft-edged and physically based shadows are discussed, and an implementation of several of these techniques is presented in order to compare their performance and behavior in real-world applications.

Chapter 2: **Introduction**

The role played by shadows in computer graphics is an often subtle but very important one. Shadows provide visual clues to the viewer as to the makeup and layout of a scene, and provide information about the geometry of objects casting the shadows and their location and distance from other objects in the scene. Creating realistic shadows is essentially a process of determining if each pixel has a view of each light source that is not blocked by any object in the scene, which can be a complicated and extremely time consuming process. As a result, shadows were generally left entirely out of earlier games, and more recently they have only been represented by crude approximations. Thanks to new research and fast, highly programmable graphics hardware, rendering physically accurate shadows in real-time applications may soon be a reality.

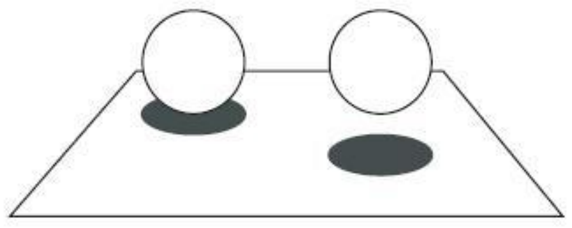
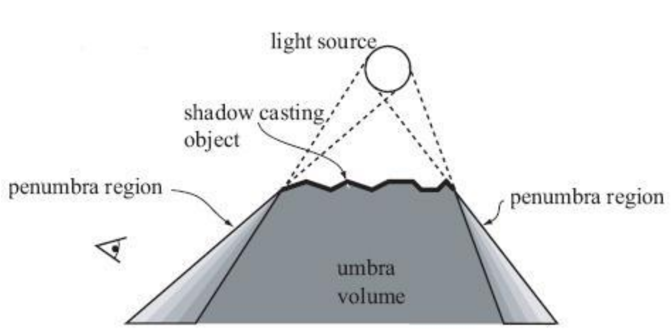
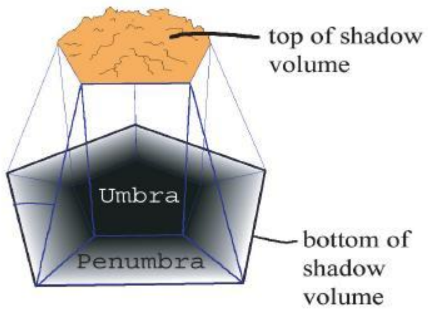


Figure 1: Shadows convey spatial information about objects in a scene. Without shadows, the viewer would be unable to determine that the sphere on the right was at a further distance from the plane than the sphere on the left.

2.1 Hard vs. Soft Shadows

The types of shadows that are typically seen in games are usually referred to as "hard" shadows, due to their hard and well defined edges. In a system which uses only hard shadows, a given pixel is either being shadowed or is fully lit, with no degrees of gradation in between. In addition, light sources are represented as a single point, rather than as an object with a certain size. This is, of course, not a very accurate model of how shadows behave in real life. Real shadows all have some degree of "softness," where the edges have a certain degree of fuzziness based on the size of the light source and the distance between the object casting the shadow (the *occluder)* and the object or surface onto which the shadow is cast (the *receiver)*. This soft edge is known as the *penumbra* region of the shadow, and the inner, fully darkened region is called the *umbra*. The shadowing in the penumbra is due to the fact that, for a real light source that has a certain size, certain parts of the receiver will have a view of the light source that is only partially blocked by the occluder, versus the umbra which is completely blocked and receives no light from the source at all.





Figures 2 and 3: (Left) Umbra and Penumbra regions in 2D, and (Right) in 3D.

Chapter3: **Hard Shadowing Techniques**

3.1 Planar Shadows

Planar shadow rendering is a quick and simple technique first introduced by Jim Blinn [Bli88]. As the name implies, the method is used to cast a shadow onto a plane (generally the ground plane), and cannot handle arbitrary receiver surfaces or self-shadowing. The technique computes a matrix that is used to project the occluder's vertices onto a plane, and the resulting geometry is then generally rendered with a dark shadow texture. The projection matrix is computed as follows [Li05]:

For a receiver plane defined by $N \cdot P + D = 0$ and $Ax + By + Cz + D = 0$, where $N = (A, B, C)$ is the normal and $P = (x, y, z)$ is a point on the plane, and a light source positioned at $l = (l_x, l_y, l_z)$: Let $V = v - l = (p, q, r)$ be a vector defined by the light's position *l* and a vertex *v* from the occluder's mesh. The projected point is defined as

$$P = tV + l \tag{3.1.1}$$

Plugging this into the plane equation and solving for t,

$$N \cdot (tV + l) + D = 0 \tag{3.1.2}$$

$$t = {}^{-(N \cdot l + D)}\!\big/_{(N \cdot V)} \tag{3.1.3}$$

And plugging equation (2.1.3) back into equation (2.1.1),

$$\begin{aligned} P &= l - (N \cdot l + D) \cdot V/(N \cdot V) \\ &= [l\,(N \cdot V) - (N \cdot l + D) \cdot V]/(N \cdot V) \end{aligned} \tag{3.1.4}$$

Separating out each of the x, y, and z components,

$$\begin{aligned} x &= [l_x(N \cdot (v-l)) - (N \cdot l + D)(v_x - l_x)]/(N \cdot (v-l)) \\ &= [l_x(Av_x + Bv_y + Cv_z - N \cdot l) - (N \cdot l + D)(v_x - l_x)]/(Av_x + Bv_y + Cv_z - N \cdot l) \\ &= [l_xA - N \cdot l - D)v_x + l_xBv_y + l_xCv_z + l_xD]/(Av_x + Bv_y + Cv_z - N \cdot l) \end{aligned} \tag{3.1.5}$$

$$\begin{aligned} y &= [l_y(N \cdot (v-l)) - (N \cdot l + D)(v_x - l_x)]/(N \cdot (v-l)) \\ &= [l_y(Av_x + Bv_y + Cv_z - N \cdot l) - (N \cdot l + D)(v_y - l_y)]/(Av_x + Bv_y + Cv_z - N \cdot l) \\ &= [l_yAv_x + (l_yB - N \cdot l - D)v_y + l_yCv_z + l_yD]/(Av_x + Bv_y + Cv_z - N \cdot l) \end{aligned} \tag{3.1.6}$$

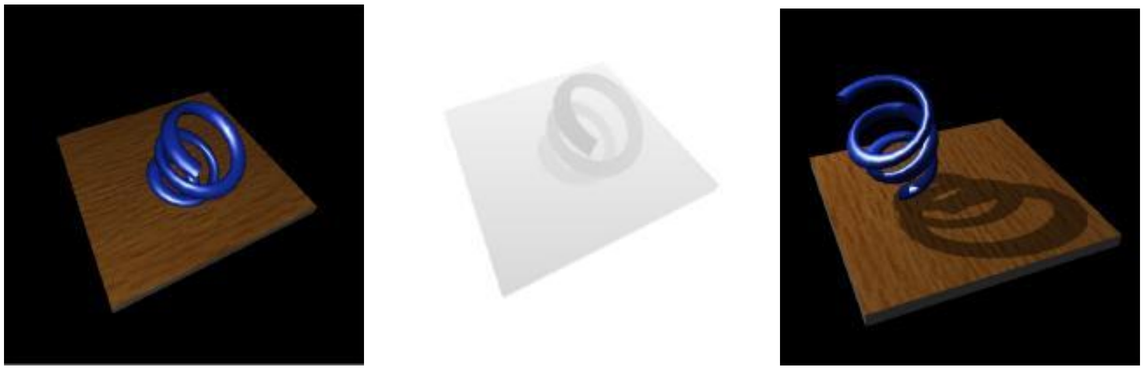
$$\begin{aligned} z &= [l_z(N \cdot (v-l)) - (N \cdot l + D)(v_x - l_x)]/(N \cdot (v-l)) \\ &= [l_z(Av_x + Bv_y + Cv_z - N \cdot l) - (N \cdot l + D)(v_z - l_z)]/(Av_x + Bv_y + Cv_z - N \cdot l) \\ &= [l_zAv_x + l_zBv_y + (l_zC - N \cdot l - D)v_z + l_zD]/(Av_x + Bv_y + Cv_z - N \cdot l) \end{aligned} \tag{3.1.7}$$

The transformation matrix is therefore

$$M = \begin{bmatrix} l_xA - N \cdot l - D & l_xB & l_xC & l_xD \\ l_yA & l_yB - N \cdot l - D & l_yC & l_yD \\ l_zA & l_xB & l_yC - N \cdot l - D & l_zD \\ A & B & C & -N \cdot l \end{bmatrix} \tag{3.1.8}$$

3.2 Shadow Mapping

Shadow mapping is an image-space technique that was introduced by Lance Williams at Siggraph ’78 [Wil78]. In an initial pass, the scene, or some subset of the scene’s occluder objects, is rendered into a depth buffer (the shadow map) using the light’s position and direction as the camera’s position and direction. This shadow map then contains the distances of each pixel to the light in the light’s coordinate system, and can be thought of as a representation of the light’s visibility of the scene. In the following rendering pass, when rendering the scene from the camera’s point of view, each pixel is transformed into the light’s coordinate system and its light-space depth at its light-space xy position is compared to the depth stored in the shadow map at the corresponding location. If the pixel's depth is less than or equal to the depth stored in the shadow map at that location, then the pixel is closer to the light than any of the occluders in the scene, and is therefore being fully lit. If the pixel's depth is greater, then it lies behind some occluder and can therefore not be receiving any light from that light source (meaning that it is being shadowed).

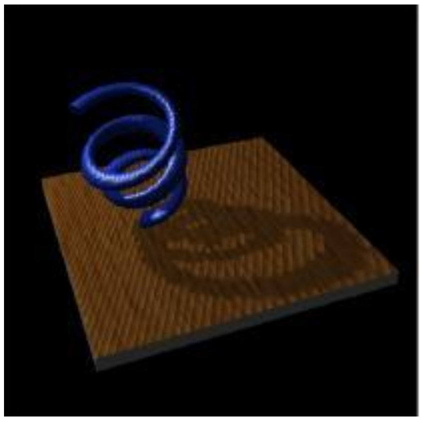
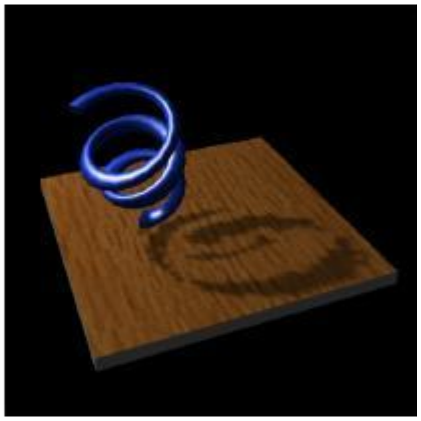

Figures 4-6: (Left) The scene as viewed from the light's point of view. (Center) A visualization of the shadow map, with darker pixels representing smaller depth values. (Right) The final image rendered from the camera's point of view.

In practice, the light-space depth values only need to be computed per-vertex, and that depth value can be interpolated between vertices per-pixel. For directional light sources, the world to light-space transformation is a simple orthographic projection. For spot light sources, the transformation is a projective transformation with the field of view either determined by the angle of the light cone or set wide enough to include all occluder geometry. Omnidirectional, or point, light sources require special handling due to the fact that they have a 360° field of view. Omnidirectional lights are typically treated as a cluster of six spot light sources each with a 90° field of view and facing directions that are 90° or 180°

from each other [ref]. Each of these light sources renders into a separate face of a cube map, and the cube map is used for depth lookups. It is also possible to use a parabolic projection and to represent the omnidirectional light's view of the scene as a pair of back-to-back parabolic shadow maps [BAS02] [Ann02].

3.2.1 Shadow map aliasing artifacts

One potential problem with rendering shadows using shadow maps is that floating point precision problems often cause speckled shadow artifacts on receiver surfaces that are unshadowed. The problem is sometimes referred to as "shadow acne" [Ura05], and is similar to z-fighting. One way of dealing with this problem is to add a small offset bias to the depth values stored in the shadow map. However, this can lead to "light-leakage," where the area directly underneath an occluder is incorrectly illuminated, if the offset is too large. Picking an appropriate offset value is generally a trial and error process, as this offset can vary from scene to scene. Another option is to only render the back facing polygons into the shadow map. This hides any artifacts that might be present, since back facing geometry should all be shadowed anyway. Woo presented a solution that stores the average depth for each object's front and back facing geometry, which is effectively the depth running through the center of the object [Woo92].



Figures 7 and 8: (Left) Shadow acne resulting from using an offset that is too small. (Right) Light leakage resulting from an offset that is too large.

Another major issue with shadow maps is aliasing at the shadow's border, which results in shadows with jagged or pixilated edges and which is due to the discrete resolution of the shadow map. There are two cases where this occurs: Perspective aliasing occurs when areas close to the camera correspond to a relatively small area of the shadow map (figure 10), and projection aliasing occurs when a surface is nearly parallel to the light direction, which causes an area of the shadow map to be stretched over a larger area than other surfaces in the scene (figure 11). Ideally, each pixel in screen space would map to exactly one pixel in the shadow map, which would result in zero aliasing of this type. However, for standard shadow mapping in non-trivial scenes this will never be the case. Over the years a number of techniques have been developed to deal with this problem. Percentage closer filtering hides this problem by taking several samples of the shadow map at slightly offset positions and averages the visibility results [RSC87]. The resulting shadow has a blurred edge, which makes aliased borders less apparent. NVIDIA graphics chipsets currently perform four-sample percentage closer filtering of depth buffers automatically in hardware [Ura05]. Fernando introduced a quadtree based data structure called an adaptive shadow map (ASM) which stores sections of the shadow map at varying resolution at each of its nodes [Fer02]. During the process of progressively refining the ASM, a new node is created for cells of the shadow map that are at the shadow boundary and that occupy a large screen-space area. Deep shadow maps store visibility information at each pixel for a number of depths, and work well for scenes containing semi-transparent objects (such as clouds) or objects with very fine and layered geometry (such as hair or fur) [LV00]. However, deep shadow maps were designed to be used with high quality offline rendering systems and are not applicable to real time application using current generation hardware. Several techniques have been developed which warp the shadow map in such a way as to distribute the shadow map pixels so that they map more closely to screen pixels, or to distribute the pixels more densely closer to the shadow borders or in areas with higher frequency. Perspective shadow maps avoid aliasing by computing the shadow map in the camera's post-perspective space (normalized device coordinate space) [SD02]. This is done by transforming the scene and the light's position by the camera's projection matrix and then rendering the shadow map from the light's point of view. The resulting shadow map has a view of the scene in which objects close to the camera appear larger, and so aliasing is avoided because the objects closest to the camera (which are the objects most likely to have their shadows affected by aliasing problems) have more pixels devoted to them in the shadow map. However, shadow acne can become a problem with perspective shadow mapping because the objects are scaled non-uniformly and therefore using a constant bias can produce incorrect results.

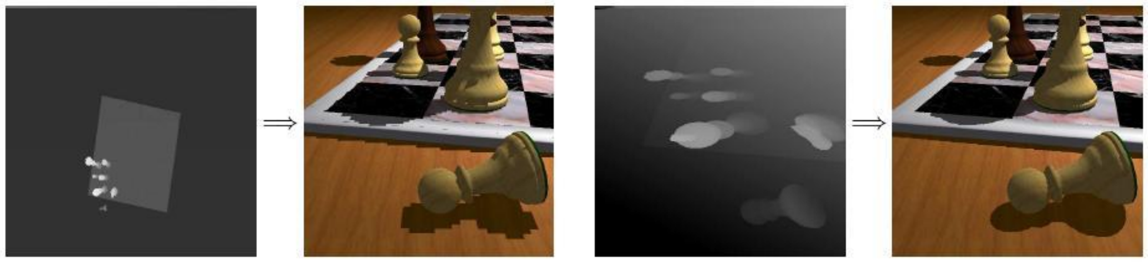


Figure 9: (Left) Traditional shadow mapping showing shadow aliasing artifacts. (Right) Perspective shadow map.

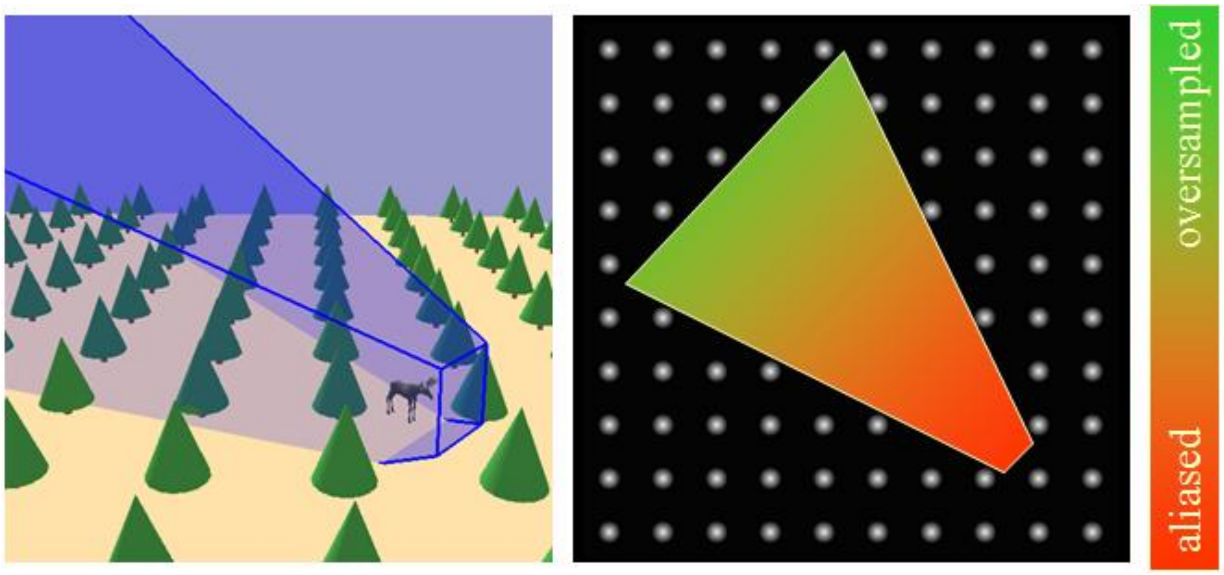


Figure 10: (Left) A scene with rows of pine trees and a moose model. The blue region shows the camera frustum. (Right) A view of the shadow map, overlaid with the region sampled when rendered in the blue camera frustum at the left. The red area shows where perspective aliasing will occur due to shadow map under-sampling.

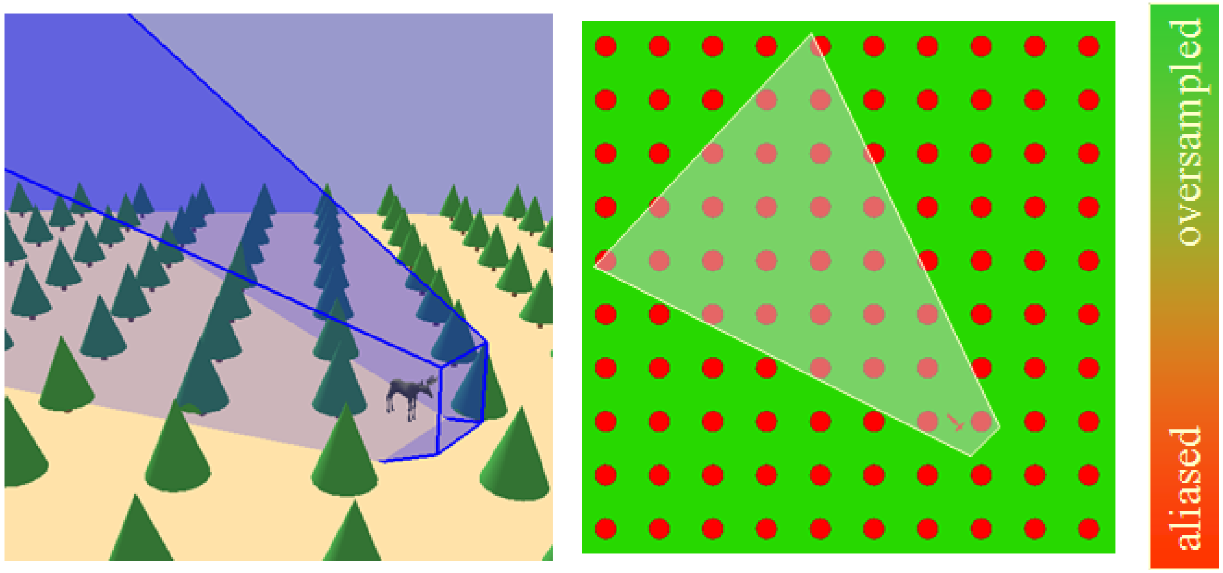


Figure 11: (Right) Projection aliasing caused by surfaces (the green cones) which are nearly parallel to the incoming light direction.

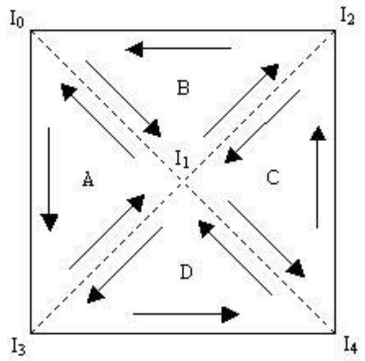
3.3 Shadow Volumes

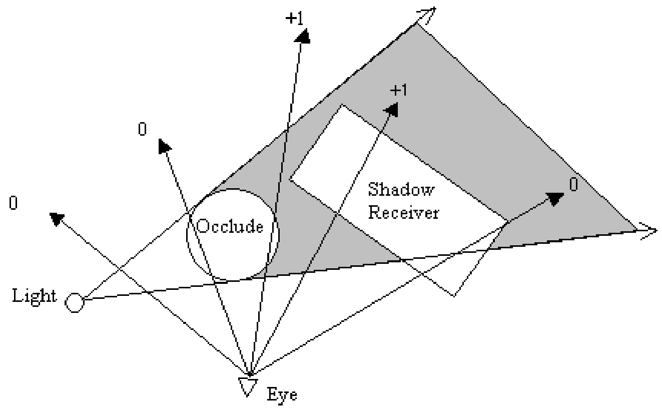
The shadow volume algorithm is a geometry based technique introduced by Franklin Crow in 1977 [Cro77]. The basic idea is that a shadow volume is defined to be a volume within which every point is in an occluder's shadow. In practice, the shadow volume technique makes use of the stencil buffer to mask off pixels that can be rendered with or without contribution from a particular light source [Hei91].

To construct the shadow volume, it is first necessary to determine the occluder object's silhouette as seen from the light source. One way to achieve this is to loop through all adjacent pairs of triangles, and if the dot products between each of the triangle's normals and the light source are of opposite sign (meaning that one triangle is front facing and the other is back facing), then their shared edge is a silhouette edge and can be stored in a list. Another approach is to cull away all back facing geometry and build a list of edges that are only present in one triangle [Kwo02]. This produces a correct silhouette because all interior edges will be shared by exactly two triangles (assuming no t-junctions), and any silhouette edge will have had the other (back facing) triangle which shared that edge culled away. Once the silhouette is determined, a vertex buffer is created that contains two instances of each edge vertex. One set of each pair of edge vertices is then extruded outwards away from the light source, and the resulting quad defined by the four vertices comprises one face of the shadow volume. The resulting geometry is then rasterized into the stencil buffer, where the value is incremented for front facing quads if the depth test passes and decremented for back facing quads if the depth test passes. When rendering the scene, any pixel with a stencil buffer value that is greater than zero lies within a shadow volume. Intuitively, a value greater than zero indicates that a ray cast from the camera to a certain pixel has entered more shadow volumes than it has exited, and is therefore within a shadow volume. On the other hand, a stencil buffer value of zero indicates that the ray has either not entered or exited any shadow volumes, or that it has exited every shadow volume that it has entered. This is known as z-pass, and it works fine as long as the camera is not within any shadow volume. If the camera is within a shadow volume, then the count of shadow volume entry and exit will be off because the ray described above will not have entered the shadow volume, and in this type of situation the shadows are the opposite of what they should be. The solution to this problem is to use a technique called z-fail, or Carmack's Reverse, which was officially documented by Everitt and Kilgard in 2002 [EK02], but which was known by John Carmack and others in the game development industry as early as 1999 [AM04b]. Using z-fail, the stencil buffer value is incremented for back facing quads that fail the depth test and decremented for front facing quads that fail the depth test. The effect is that the stencil buffer value represents a count of the number of entries and exits for a ray cast from infinity to each rendered pixel, and is therefore independent of the camera's position. This provides an elegant solution, but adds some complexity due to the fact that the shadow volumes cannot be open and must be capped with polygons at each end.

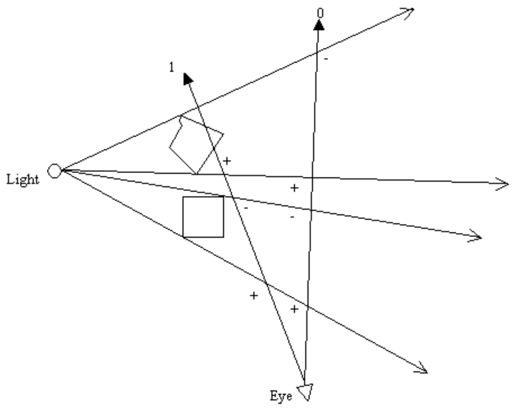
3.3.1 Problems with shadow volumes

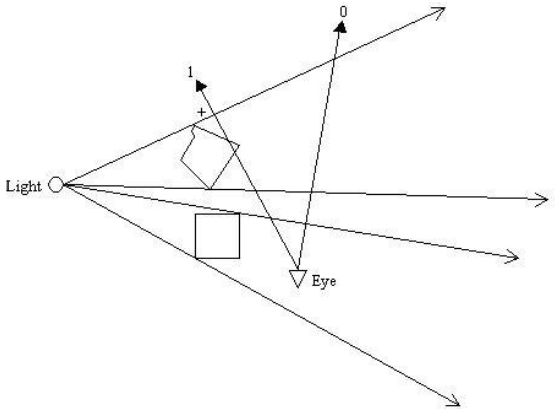
The shadow volume technique produces accurate hard-edged shadows without any of the aliasing problems associated with shadow maps, but it tends to be very processor intensive and does not scale well with scene complexity. The main bottleneck associated with shadow volumes is that they tend to consume a large amount of the graphics card's fill rate. Fill rate is defined as the number of pixels that can be written into video memory over a given period of time. Because every single piece of shadow volume geometry must be rasterized into the stencil buffer and cannot be culled away as in normal rendering, the fill rate becomes the limiting factor. Additionally, calculating the object's silhouette every frame can use up a large number of CPU cycles. The technique also introduces certain inefficiencies due to the fact that the object's vertex and index buffers must be accessible to the CPU for silhouette determination, instead of the more desirable situation of having those buffers only being stored and transformed on the graphics card.





Figures 10 and 11: (Left) Object silhouette determination: Notice that interior edge I0I1 is shared by triangles A and B, and that silhouette edge I0I3 is only present in triangle A. (Right) A simplified representation of a shadow volume.





Figures 12 and 13: (Left) Z-Pass. (Right) Z-Fail.

Chapter 4: **Offline and Pre-Computed Soft Shadowing Techniques**

4.1 Distributed Ray Tracing

Physically accurate soft shadows are relatively simple and straightforward in a ray tracer. For each rendered pixel, cast a number of rays from that pixel to different sample points on the area light source, determine if these rays intersect with any other objects in the scene, and average the visibility test results to determine the percentage of the light source is visible to that pixel and therefore its amount of shadowing. In practice this requires a fairly large number of “shadow feeler” rays to be cast per-pixel in order to prevent banding artifacts in the penumbra. Such an approach would not map well to a real-time renderer, although similar approaches have been attempted.

4.2 Discontinuity Meshing

Another method sometimes used by ray tracers is known as discontinuity meshing [Hec92], [LTG92]. Discontinuity meshing involves projecting an occluder’s geometry onto the receiver, and then using that projected geometry to construct the umbra and penumbra regions. The projection is similar to that of planar shadows, except generalized to arbitrary receiver geometry. While such an approach is faster than sampling the light sources with multiple shadow feelers, it is still much too costly for real-time applications.

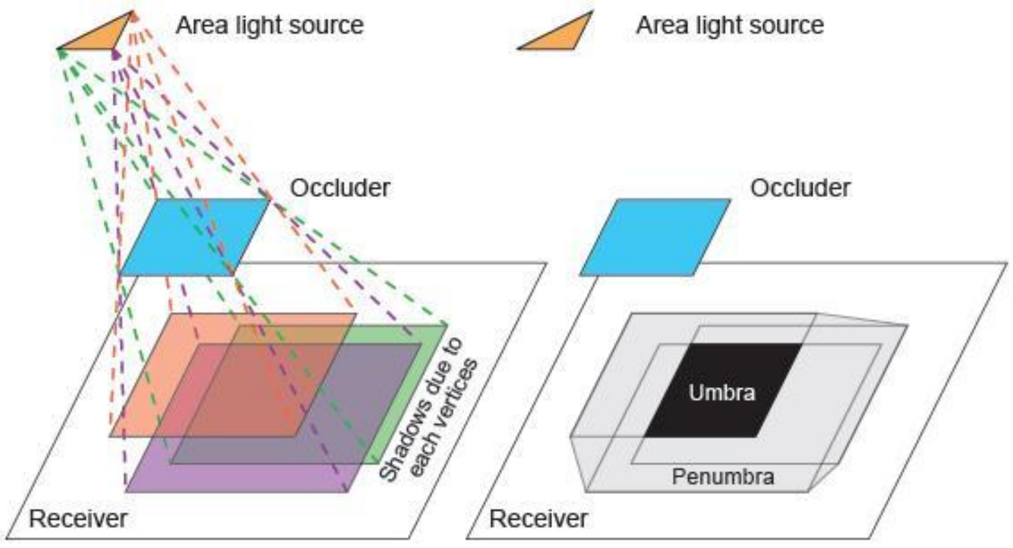
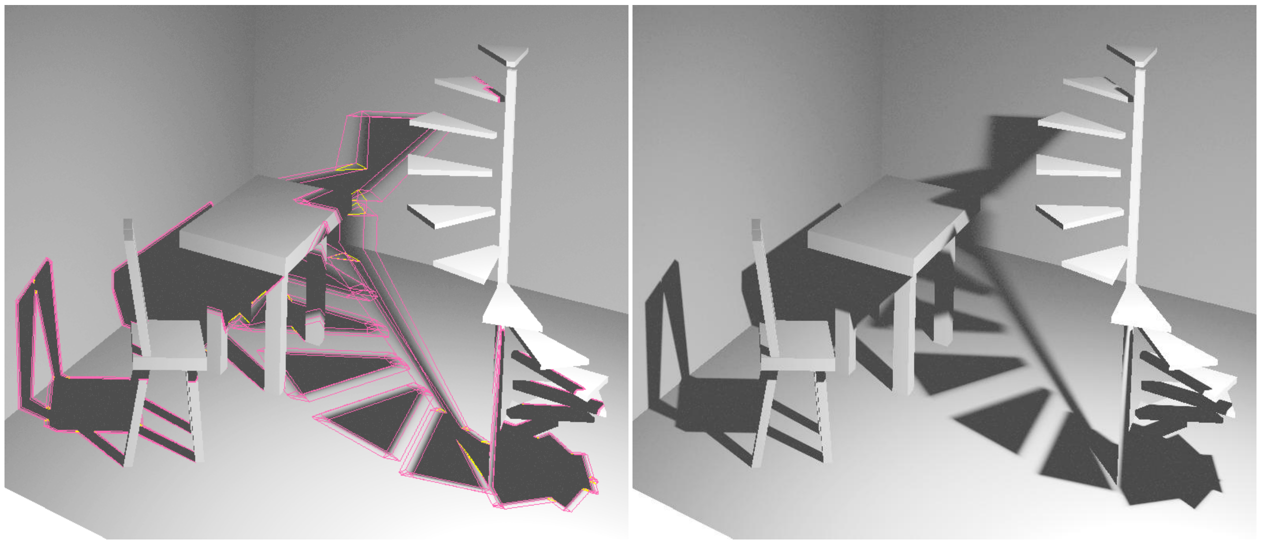


Figure 14: Discontinuity meshing.



Figures 15 and 16: (Left) Pink lines represent the discontinuity mesh. (Right) The final rendered scene.

Chapter 5: **Image Space Soft Shadowing Techniques**

5.1 Combined Shadow Maps

One simple method of generating soft shadows involves generating a set of shadow maps as seen from various points distributed across an area light source and to use this group of shadow maps to determine the percentage of visibility for each pixel in the scene [HH97], [Her97]. Each of these shadow maps is used to render a light/dark visibility map which represents the hard-edged shadow as seen from the camera. These hard-edged visibility maps are then blended together to form a soft-edged visibility map which is used to modulate the intensity of the lit pixels in the final rendered image. This approach is similar to casting multiple shadow feelers in ray tracing, and it produces shadows that approach physical correctness for large numbers of combined visibility maps. While this approach can take advantage of the graphics hardware and can therefore be more efficient than ray tracing, generating many shadow maps is still very computationally expensive and can consume a large amount of video memory. Additionally, the shadows rendered from a small number of shadow maps typically have noticeable banding artifacts and areas where the shadows are not evenly blended or smoothed. Due to these limitations, this technique is not very practical for real-time applications.

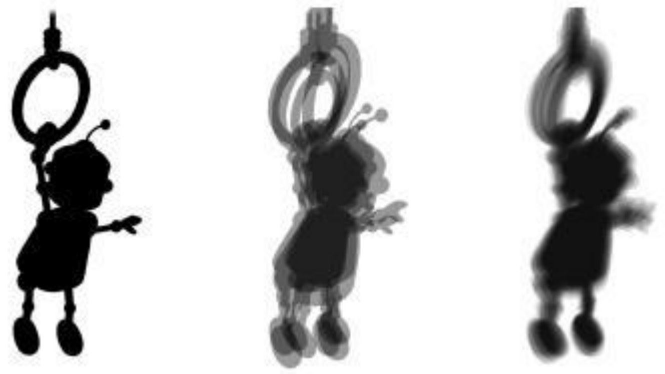


Figure 17: (Left) A single visibility map. (Center) Four visibility maps. (Right) 64 visibility maps.

5.2 Blurred Visibility Maps

Another simple technique is to generate a single visibility map that is evenly blurred by a specified amount before being used to modulate the intensity of the rendered pixels in the rendering pass. The resulting shadow is smooth and visually appealing, but is not physically accurate and can only be used in situations where a rough approximation is adequate. The drawbacks of using this technique can be seen in situations where a tall object is standing on a roughly planar floor, one common example of which would be a human character model standing on flat ground. In such a situation, the shadow close to the character's feet is just as blurry as the part of the shadow cast by his head, where in reality the shadow would have a harder edge near the feet and legs and would gradually become softer with increased distance from the character's base. Regardless of this glaring inaccuracy, this technique is quite common in current generation games.

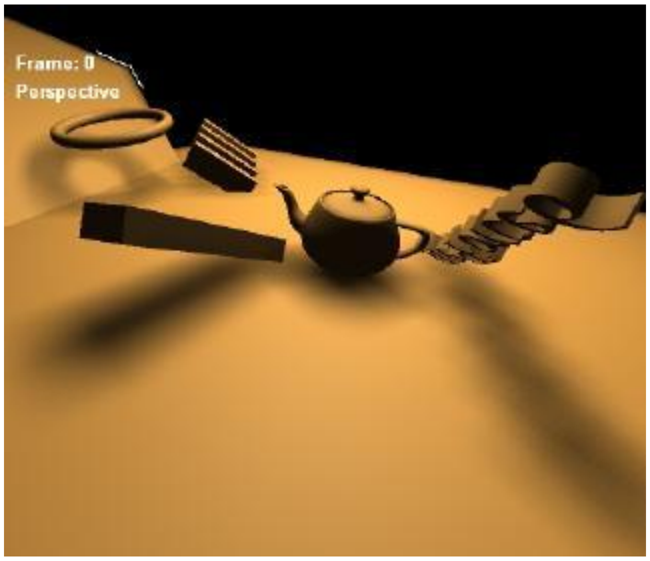


Figure 18: An evenly blurred shadow.

5.3 Percentage Closer Filtering

Another approach is to take several samples of the shadow map at points close to the area being shaded, compute the depth test for each of these samples, and then average the results to determine the percentage of light received by the point being rendered. The penumbra width and therefore the overall softness of the rendered shadow is determined by the area of the shadow map that is sampled, which is the width of the filter kernel being applied. This technique is known as Percentage Closer Filtering [RCS87], and the name indicates that the filtered shading result is the percentage of samples that are closer to the light source than the point being shaded. The results of this technique are similar to those of evenly blurred visibility maps, but are closer to being physically accurate. However, this approach suffers from banding artifacts with larger penumbra widths. These artifacts can be reduced by increasing the number of samples per pixel, but this can incur a very high runtime cost.

5.3.1 Variable Kernel Width Percentage Closer Filtering

Fernando [Fer05] introduced Percentage Closer Filtering using a filter width that is variable depending upon the distance of the occluder and receiver from the light source. Figure 19 shows the relationship between the occluder, the receiver, the width of the light source, and the width of the penumbra. Using similar triangles and assuming the occluder, receiver, and light source are parallel, the width of the penumbra is given by the following equation.

$$w_{penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}} \tag{5.3.1.1}$$

The light source's width is constant and the receiver depth is the depth of the point being shaded. For non-planar objects, the occluder's depth can be estimated by averaging the depth over a given area. This can be achieved by sampling the shadow map and averaging the depth results. These extra samples can be reused when the shadow map is sampled again using the computed filter width. In practice a relatively small number of samples are required to estimate the average occluder depth.

Like fixed-width Percentage Closer Filtering, this technique also suffers from banding when the shadow map is under-sampled. This problem is made worse by the fact that, in dynamic scenes, the occluder can be brought arbitrarily close to the light source, which in turn can produce an arbitrarily large penumbra. While this technique does produce soft shadows that harden on contact, the shadows are still not physically accurate. This physical inaccuracy is more apparent for large area light sources, mostly due to the fact that, in physically accurate lighting, different ends of a light source "see" different parts of an occluder's geometry, and these different geometries are ultimately combined in the occluder's shadow. Additional inaccuracies are introduced when there is a large amount of depth complexity in the scene. For example, in a scene with many objects at various depths, the initial occluder depth estimate could potentially include several different objects and would lead to an incorrect estimate of the penumbra's width.

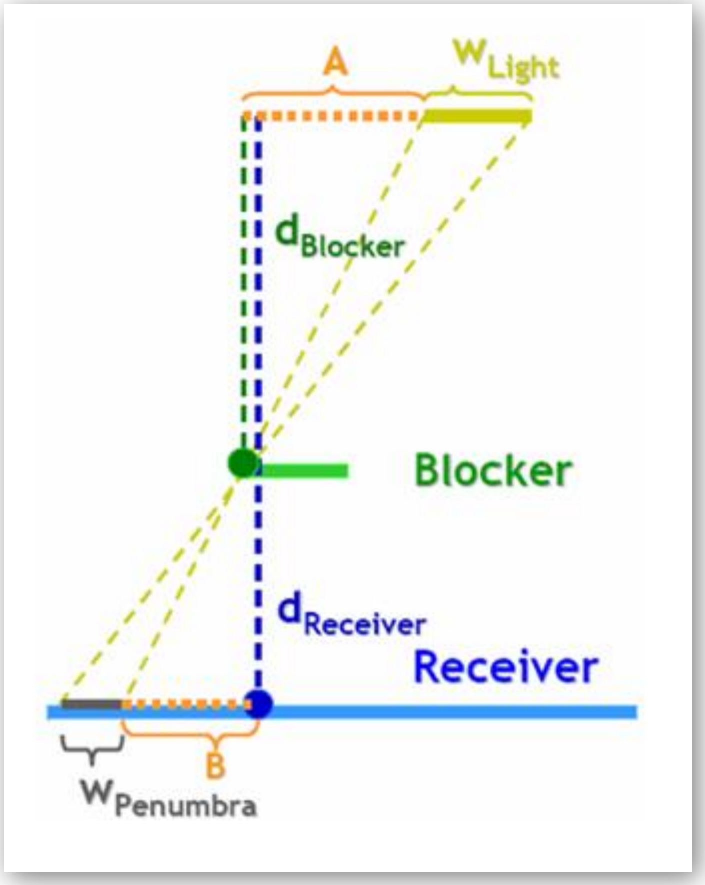


Figure 19: The relationship between the occluder, receiver, light source width and penumbra width.

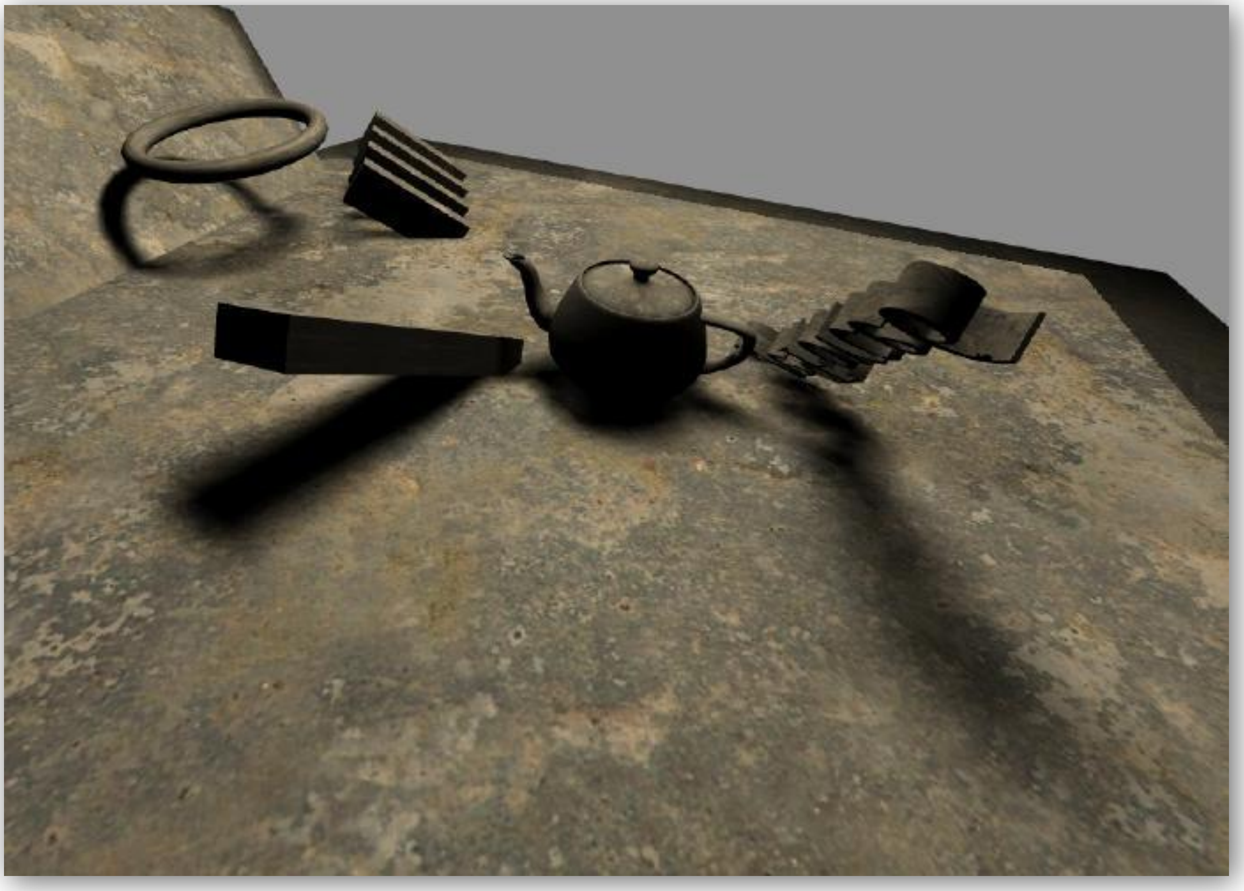


Figure 20: Percentage closer soft shadows

5.4 Pre-Filtered Shadow Maps

Techniques such as Percentage Closer Filtering that involve sampling the shadow map at points around the pixel to be rendered and averaging the results of the depth tests are effective and fairly straightforward to implement, but they suffer from one main flaw; they require many high-latency texture fetches into the shadow map for each rendered pixel, which can be very time consuming for larger penumbra areas or in cases where a very high quality soft shadow is desired. A performance boost could be realized if there was a method of pre-filtering the data stored in the shadow map so that only a single shadow map sample per screen pixel would be required at run time. Unfortunately, simply blurring the shadow map itself is not sufficient. This is because the results of this type of filtering would give the average depth over an area of the shadow map, which in the context of a traditional shadow mapping technique doesn't really make sense. Additionally, using the average depth would not produce soft-edged shadows since the depth test for each rendered pixel would still give a result of either fully lit or fully shadowed.

5.4.1 Variance Shadow Maps

Variance Shadow Maps [DL06] are able to take advantage of such a pre-filtering step through the use of probability theory and by storing slightly different data in each shadow map texel. The key insight of Variance Shadow Maps is that the data stored in the shadow map represents the *distribution* of depths at each texel, and to approximate this distribution the shadow map stores the first two moments of the depth function, which are the mean depth and the mean squared depth. These two moments are used to compute a bound on the fraction of the distribution that is more distant from the light source than the pixel being rendered, and this percentage corresponds to the percentage of light the pixel receives.

The first two moments over a given filter region are defined as follows:

$$M_1 = E(z) = \int_{-\infty}^{\infty} zp(z)dz \qquad (5.4.1.1)$$

$$M_2 = E(z^2) = \int_{-\infty}^{\infty} z^2p(z)dz \qquad (5.4.1.2)$$

And from these moments we can compute the mean μ and variance σ^2 as follows:

$$\mu = E(z) = M_1 \tag{5.4.1.3}$$

$$\sigma^2 = E(z^2) - E(z)^2 = M_2 - M_1^2 \tag{5.4.1.4}$$

The variance can be interpreted as being a measure of the width of a distribution, and describes how far from the mean a set of values can be distributed. The upper bound on the distribution is stated in Chebychev's inequality:

$$P(z \geq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} \tag{5.4.1.5}$$

$P(z \geq t)$ is the upper bound on the probability that z is greater than or equal to t. In the context of shadow mapping, z is the depth of a pixel stored in the shadow map, t is the depth of a pixel being shaded, and $P(z \geq t)$ is the probability that the pixel is being shadowed. If z is the average depth over an area, then the result of this function is the upper bound on the probability that the shadow map texels in that area are shadowing the rendered pixel, which can be thought of as a measure of the amount of visibility at that point, and therefore $p_{max}(t)$ can be used directly as the amount of shading to apply to each pixel. It should be noted that even though equation 5.4.1.5 provides an upper bound on the visibility for arbitrary scenes, the inequality reduces to an equality for planar and parallel occluders and receivers. Many scenes nearly satisfy this case over small areas (specifically over the area that is being averaged), and as a result the equation works well enough in practice.

At runtime, the filtering pass can be performed in a number of different ways. Hardware based anti-aliasing and mip-mapping can be utilized for performance, or a custom filter can be used in a pixel shader for more control over the results of the filter pass. Alternatively, a summed area table can be constructed from the depth and squared depth values and used instead of the filtered textures. If a summed area table is used, the average depth is found by looking up the values in the four corners of the filter area. This introduces additional computational costs and increases the number of required texture lookups, but it has the advantage of allowing for arbitrarily sized filters. As a result, this approach can produce shadows that harden on contact, since the filter size can be made larger or smaller depending upon the relationship between the occluder, the receiver, and the light source as described in section 5.3.1.

While Variance Shadow Maps can produce very convincing soft shadows, there are a few situations that can lead to artifacts. The most noticeable artifact is referred to as light bleeding, which occurs when a shadow is cast onto several different receivers at varying depths. The result of light bleeding is that the penumbra area that falls on a closer object is also incorrectly cast onto a shadowed object further away, which is shown in figures 20 and 21. Light bleeding is caused by the fact that near an object's silhouette there can be a large amount of depth variation, and this variation leads to a larger difference between the computed upper bound on visibility and the actual visibility. In other words, the depth variation at the silhouettes breaks the assumption that a scene is roughly planar over small areas. Light bleeding can be reduced or eliminated by slicing the scene into layers and using a separate Variance Shadow Map for each layer [LM08]. By dividing the scene into layers, this approach avoids the problem by reducing the number of instances of shadows that are cast onto multiple objects at different depths.

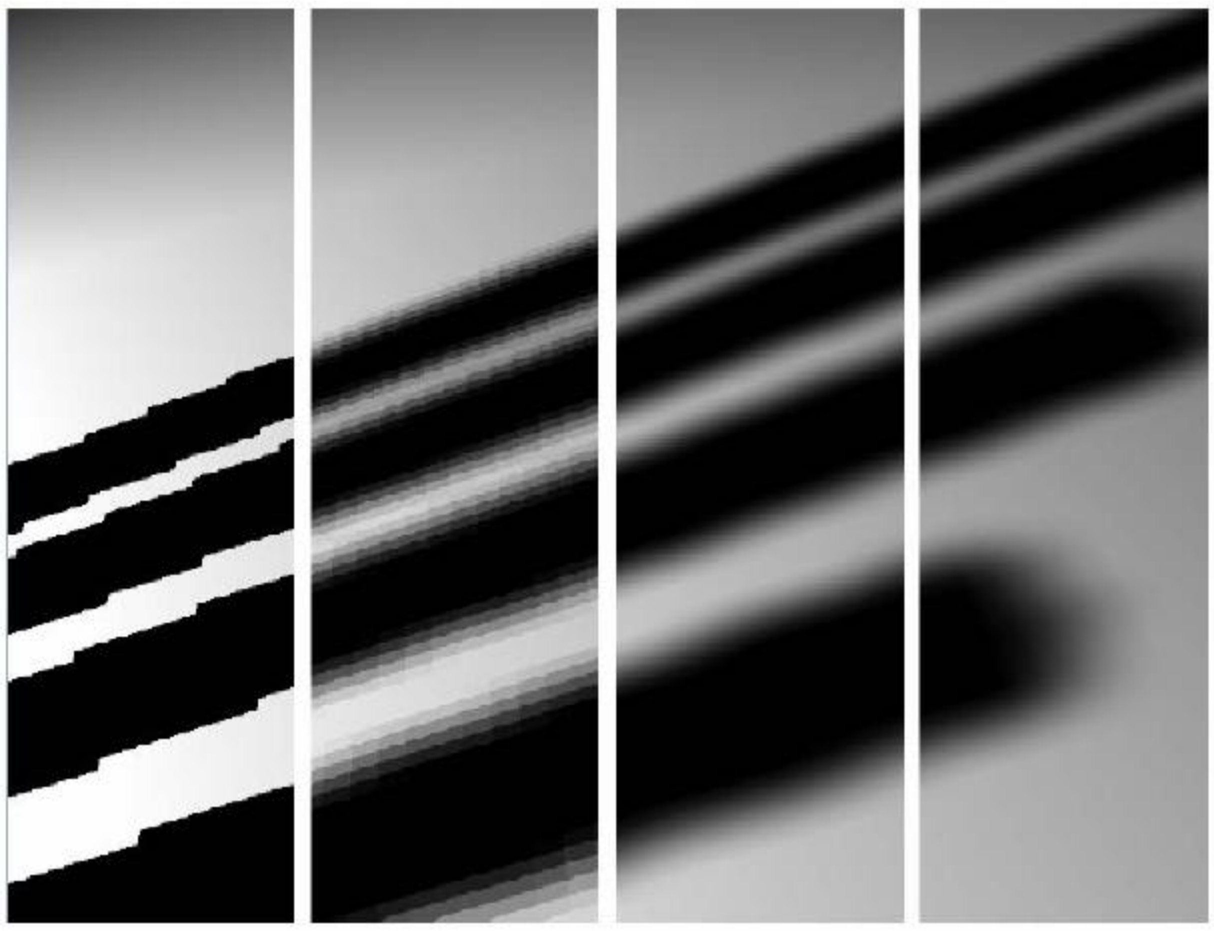


Figure 19: (From left to right) Standard shadow map, 5x5 percentage closer filtering, bilinear 5x5 percentage closer filtering, and variance shadow maps.

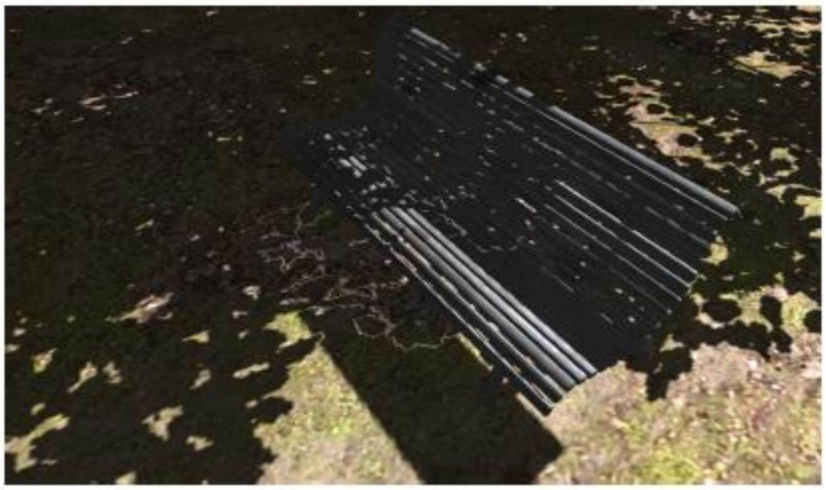


Figure 20: The penumbra areas from the tree leaves are incorrectly bleeding through to the shadow below the bench.

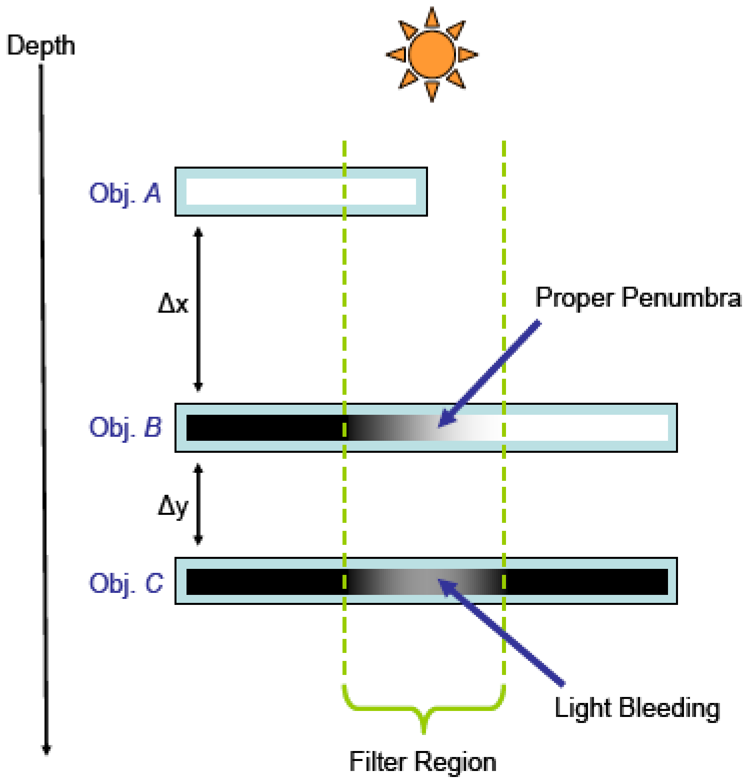


Figure 21: Receiver orientation that leads to light bleeding

4.4.2 Pre-filtering using other methods

The concept behind Variance Shadow Maps has been extended upon in several new techniques that allow pre-filtering by storing different representations of the scene's depth. In Convolution Shadow Maps [AMBSK07] and Exponential Shadow Maps [AMSFK08], the shadow test function, which determines whether or not a pixel is in shadow, is represented as a binary, or step, function with the pixel depth and shadow map depth as inputs. Since the shadow test is not linear with respect to its input arguments, it cannot be pre-filtered. However, approximating the shadow test as a sum of separable terms can turn it into a linear function that can be filtered beforehand. In Convolution Shadow Maps, the step function is represented as a Fourier series expansion. The scene depth is stored as a series of textures that store the sine and cosine terms that are filtered and used during rendering to reconstruct the values used in the shadow test function. Unfortunately the method requires many textures for the appropriate level of accuracy (16 textures were used in the paper), which leads to a high memory cost. Additionally, Convolution Shadow Maps suffer from ringing artifacts which are inherent in Fourier series expansions. Exponential Shadow Maps avoid the high memory and computational costs associated with Convolution Shadow Maps by approximating the step function with an exponential function. The resulting shadows, however, tend to look too faint and somewhat washed-out. Figure 22 shows a comparison between Variance Shadow Maps, Convolution Shadow Maps, and Exponential Shadow Maps.

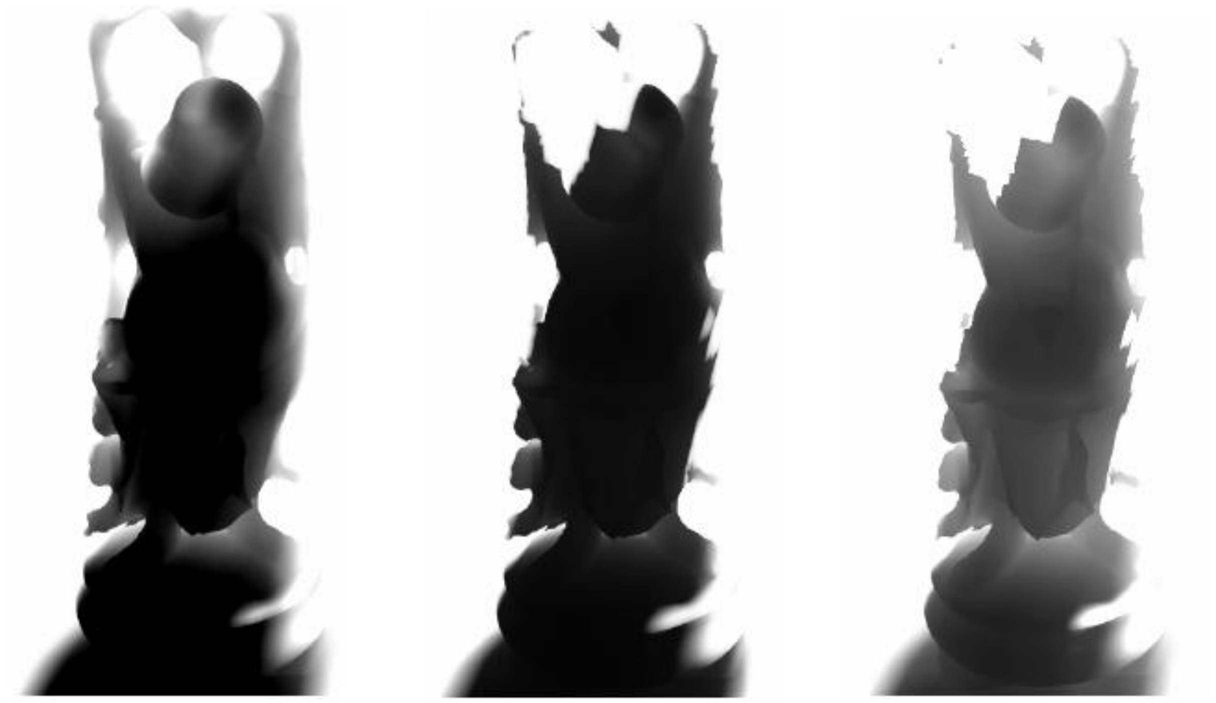


Figure 22: (From left to right) Convolution Shadow Maps, Variance Shadow Maps, and Exponential Shadow Maps. Note the large amount of light bleeding demonstrated in the images rendered using Convolution Shadow Maps and Exponential Shadow Maps.

Chapter 6: **Hybrid Image / Object Space Techniques**

6.1 Plateaus

The plateau technique is a simple extension of the planar shadow technique, where penumbra regions are added to hard edged planar shadows [Hai01]. These penumbra regions are given a smooth shading that fades from full shadow at the border between umbra and penumbra to fully lit at the penumbra's outer edge. Additionally, the width of these penumbra regions is greater for larger distances between occluder and receiver. The technique is fairly straightforward to implement, and the results are reasonably convincing. However, the technique is based on the planar shadow technique and therefore shares the same weaknesses, most notably being the inability to have shadows that are cast onto arbitrary objects and surfaces. Additionally, because these shadows are simply hard shadows with extra penumbra regions tacked on, they are larger than the correct shadow and are inadequate for situations where a shadow has sections consisting of only penumbra.

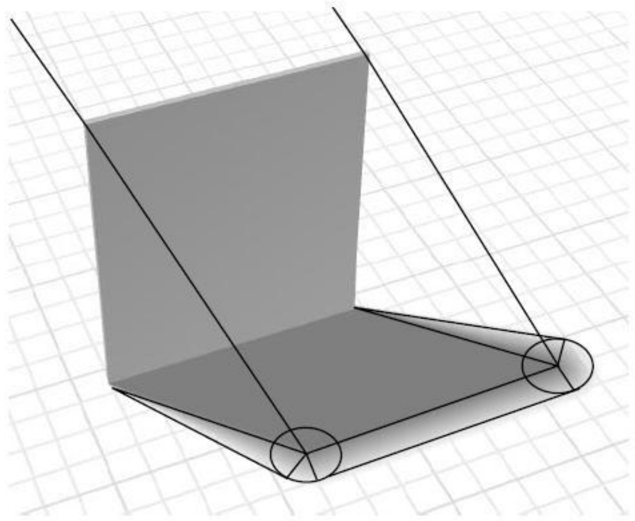


Figure 21: Plateaus.

6.2 "Smoothies" and Penumbra Maps

These two methods were developed independently and at the same time by two separate researchers, and are both essentially the same [CD03], [WH03]. They are an object/image space hybrid, where a normal shadow map is generated and extra penumbra regions are added and given widths that are based on the depth values stored in the z-buffer. These methods are similar to the plateau approach, but can be used with arbitrary receivers due to their being extensions of shadow mapping techniques. However, they also suffer from problems with physical inaccuracy because, like the plateau approach, the rendered shadows are still simply hard shadows with additional smooth penumbra regions.

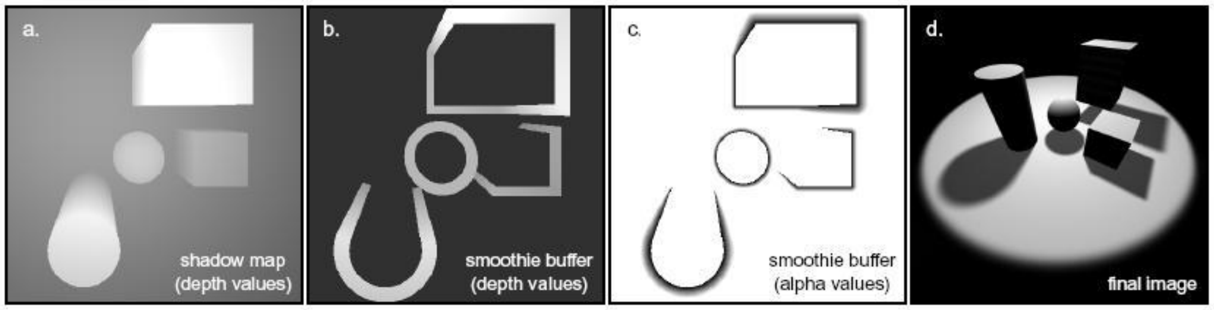


Figure 22: Images showing rendering with smoothies.

Chapter 7: **Object Space Techniques**

7.1 Penumbra Wedges

This technique, developed recently by Ulf Assarsson [AM02], [ADMA03], [AM03], is an extension of the shadow volume technique. In this technique, each of the vertices of the occluder's silhouette is extruded outward in a cone, instead of just in a straight line as in a typical shadow volume approach. This region, defined by the area between these extruded cones, forms the shadow's penumbra, and the area inside the penumbra forms the shadow's umbra. In practice, these two regions form two separate shadow volumes that are processed independently. Each are rendered in the same fashion as a typical shadow volume, with the exception being that pixels found to lie within a penumbra volume are given an amount of shadowing that varies depending upon its position relative to the penumbra region's front and back facing planes. The visibility calculation is achieved in hardware by finding the intersection points of the occluder's silhouette edge and a square representing the light source, and then using these two intersection points to perform a lookup on a 4D texture with precomputed visibility data. This approach is very physically accurate and produces the best visual results of any of the previously mentioned real-time techniques. However, the fill-rate problem typically associated with shadow volumes is more than doubled by the fact that two separate sets of shadow rendering passes are required, and what might otherwise be a minor slowdown can actually become a major bottleneck.

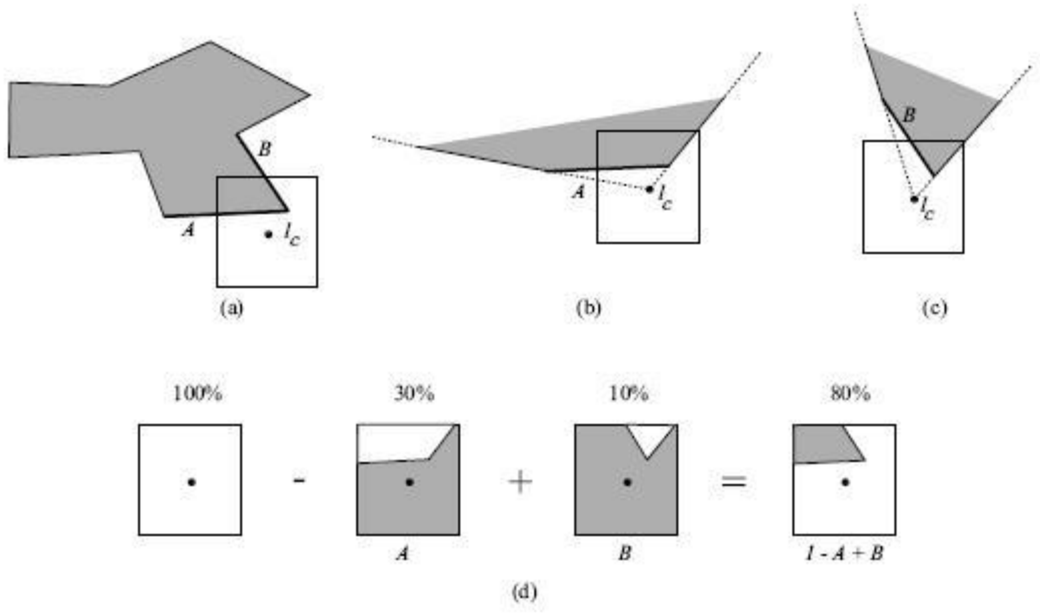
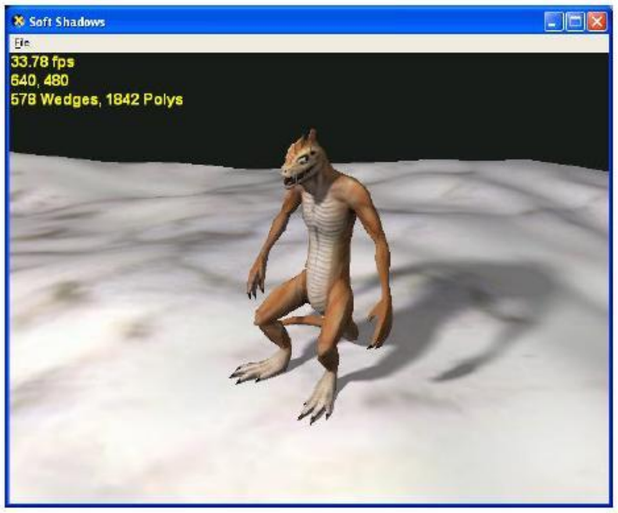
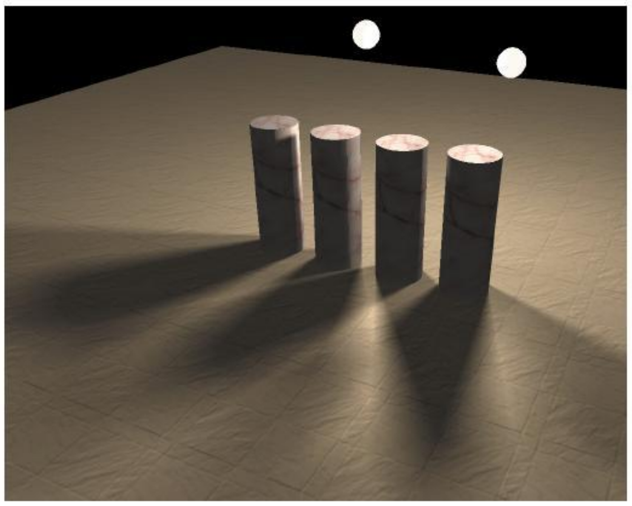


Figure 23: Computing a pixel's visibility.



```
33.78 fps
640, 480
578 Wedges, 1842 Polys
```

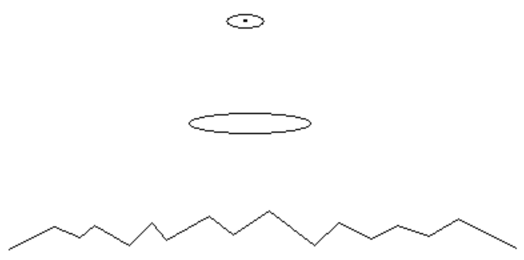


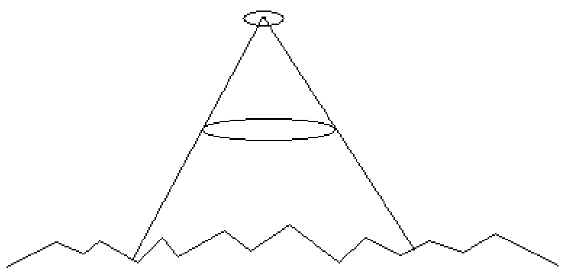
Figures 24 and 25: Two images rendered using penumbra maps.

Chapter 8: **Future Work**

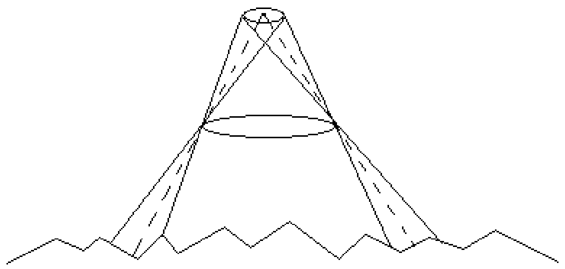
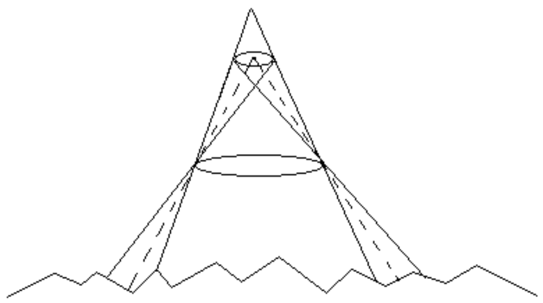
During the course of this research, a new technique based on shadow mapping was proposed. The idea behind the proposed method is similar to rendering multiple shadow maps from different points on an area light source, which approaches physical correctness for large numbers of shadow maps. The proposed technique, however, will only require two shadow maps. Figure 25 shows an area light source (top), an occluder (middle), and a receiver (bottom). Figure 26 gives a simplified image of the light's view frustum in a typical shadow mapping algorithm, and figure 27 shows two extents of roughly how the light in a physically accurate simulation would behave. In figure 28, the inner extent is extended above the light source, and two separate frustums are formed. The proposed technique uses the two points marked in red in figure 29 as two separate camera locations for rendering two separate shadow maps. The two shadow maps define the borders of the umbra and penumbra regions. A given pixel is completely shadowed if it is obscured by both shadow maps, it is completely illuminated if it is not obscured by either of the shadow maps, and it is partly shadowed if it is obscured by one shadow map and not the other. This approach would efficiently identify pixels located within the penumbra area, and would only incur twice the cost of a standard shadow map render and lookup.

Once the penumbra area is determined, there are several possibilities for determining the amount of shading to apply to each pixel. One option would be to project each edge back onto the surface of the light source in order to determine visibility, in a method similar to the method used in the penumbra wedge technique. Another option would be to compute two sets of silhouettes, one from each virtual light source, connect geometry between each edge pair, and render the geometry with a gradient texture from inner to outer silhouette edge as projected onto a plane some distance from the light sources. This rendered texture would serve as the “penumbra buffer,” which would store the amount of shading for each pixel within the penumbra, and would be sampled during the final rendering pass. A third option would be to perform a search of the neighboring texels in each shadow map in order to find the closest shadow map texel belonging to a surface that might potentially occlude the rendered pixel. From the relationship between the pixel found, the pixel being rendered, and their depths, it could be possible to determine enough information about the scene in order to accurately approximate the pixel’s visibility.





Figures 26 and 27.

Figures 28 and 29.

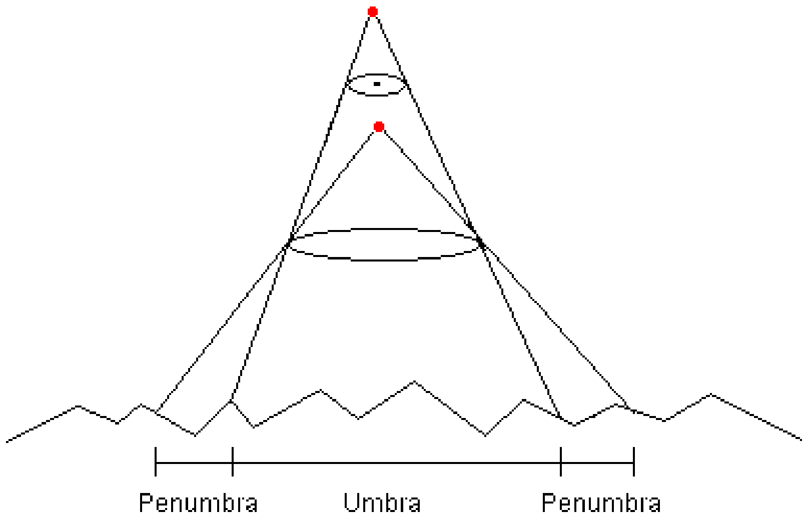


Figure 30

Chapter 9: **Implementation Details**

9.1 Introduction

Several of the hard and soft shadowing algorithms described in this paper were implemented in a windows-based application in order to compare their relative performance and quality. The application renders a user-configurable scene consisting of static and animated models and a single light source, and the camera, light, and models are all movable at run time. The default scene renders at 800 x 600 in windowed mode. The application was developed on a Windows XP machine with a 2.8 GHz Pentium D processor, 2 GB RAM, and an NVidia 8800 GT graphics card with 512 MB of video RAM, and all performance measurements and screenshots were taken using this hardware. All performance measurements were taken from a fixed camera position with texturing disabled (lighting only). The source code was built using the November 2008 DirectX 9 SDK under Visual Studio 2008.

9.2 Shadow Map

The shadow map implementation is a standard implementation using a shadow map resolution of 2048 x 2048 and a 32-bit single component floating point render target. Using a single component render target allows for the entire range of bits to be used in a single value without having to split the depth into separate red, green, blue, and alpha components in the pixel shader. Also, the floating point format allows for more precision and flexibility when compared with fixed-point formats. The shadow map was fitted to exactly match the light's frustum, and no further optimizations based on the layout of the scene were performed.

Additionally, only the back facing geometry was rendered into the shadow map in order to avoid shadow acne, as described in section 3.2.1.

The benchmark scene rendered at an average of 517 frames per second, which is significantly faster than any other technique in this implementation. By comparison, the default unshadowed scene rendered at 739 frames per second.

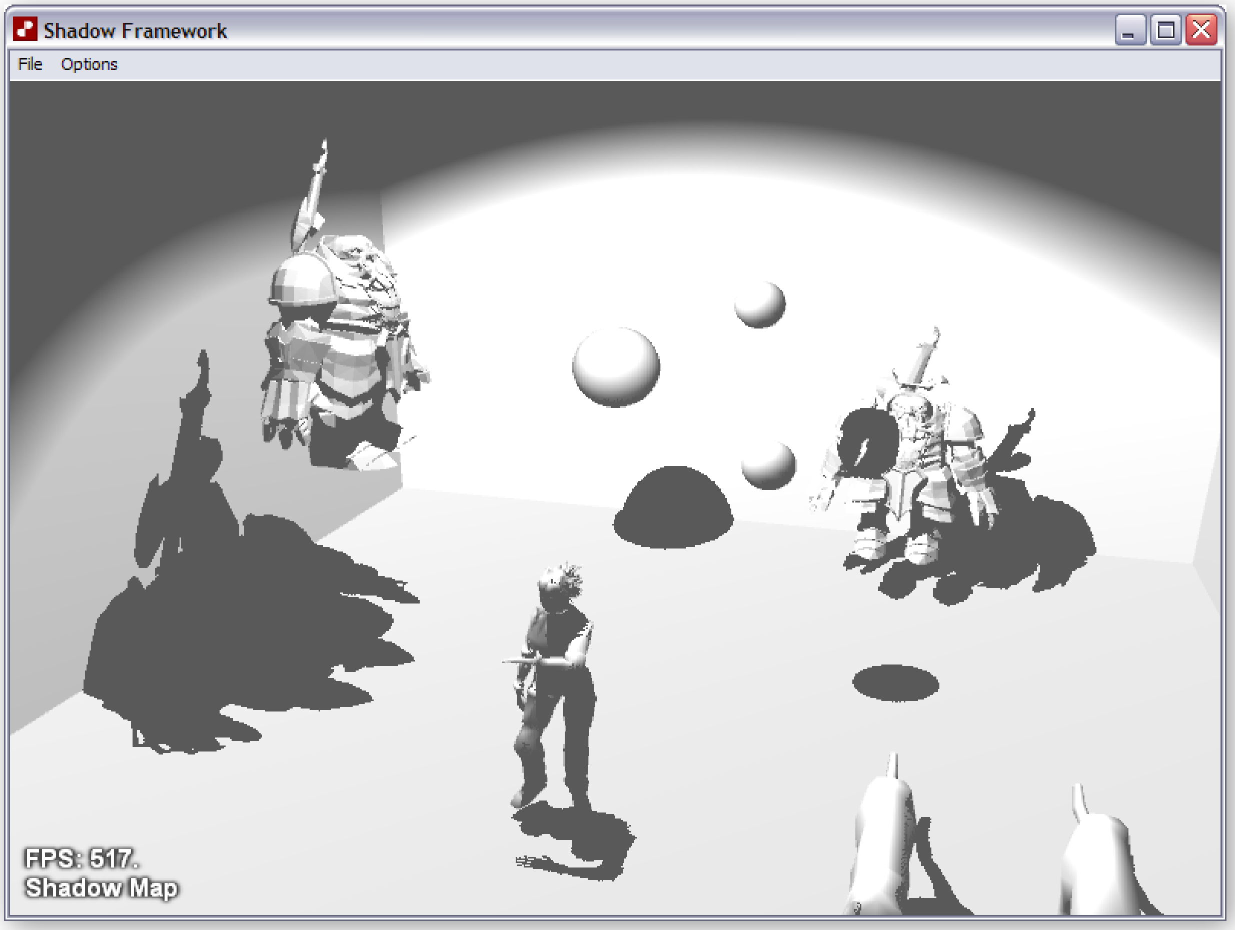


Figure 9.2.1: Standard shadow mapping technique

9.3 Shadow Volumes

In this shadow volume implementation, all shadow volume extrusion and construction was performed on the GPU. This required a special version of each model's mesh which is constructed once at application startup as follows:

1) Ensure that the mesh is manifold (in particular, each edge must belong to two faces, i.e., the model cannot contain any "cracks"). Patch any open edges if possible.
2) Build tables that contain adjacency information between triangles and edges
3) For each triangle in the original mesh
    a. Add new instances of each of the triangle's vertices to the shadow mesh's vertex buffer
    b. Create a new triangle in the shadow mesh's index buffer
4) For each edge between two triangles in the original mesh
    a. Add three new vertices for each of the original edge's two vertices and add all six new vertices to the shadow mesh's vertex buffer. Ensure that the duplicated vertices use two of the first triangle's face normals and one of the second triangle's face normals at one end of the edge, and one of the first triangle's face normals and two of the second triangle's normals at the other end of the edge.
    b. Add two new triangles to the shadow mesh's index buffer such that a degenerate quad is formed by these new vertices. The degenerate quad will

lie exactly along the edge. Ensure that the winding order is such that the normal points in the same direction as the normals for the edge’s adjacent triangles. Additionally, the normals from the vertices duplicated in the previous step will ensure that each of these two new triangles will have two vertex normals from one adjacent triangle and one from the other.

The resulting mesh contains a triangle list of all the original mesh’s triangles plus two degenerate triangles with zero area that lie along each edge. None of the vertices in this mesh are shared between triangles, they are all unique. This mesh is then sent down to the graphics card for use at runtime.

The shadow volume is extruded from this specially constructed mesh just before being rendered into the stencil buffer as follows: In the vertex shader, find the dot product between the vertex normal and the light direction. A positive dot product means that the geometry is facing towards the light source, and the vertices can be transformed as normal. A negative dot product, however, means that the geometry is facing away from the light source, and these vertices are first translated by a fixed amount along a vector from the vertex to the light source. This translation effectively pushes all back facing geometry away from the light source, while leaving all front facing geometry in place. As a result, the degenerate triangles along the edges between front and back facing geometry are then stretched out to form the side sections of the shadow volume. In other words, the mesh is split in half along the object’s silhouette, and the formerly degenerate triangles are extended to fill in the gaps.

The downside to this approach is that there is a large amount of geometry that is duplicated and taking up extra space in video RAM. Additionally, these meshes take extra time to compute and to push down to the graphics card, but fortunately this performance hit only happens once. The alternative, which is to compute the object’s silhouette and construct the shadow volume geometry on the CPU before pushing it down to the GPU per frame, is much, much more costly.

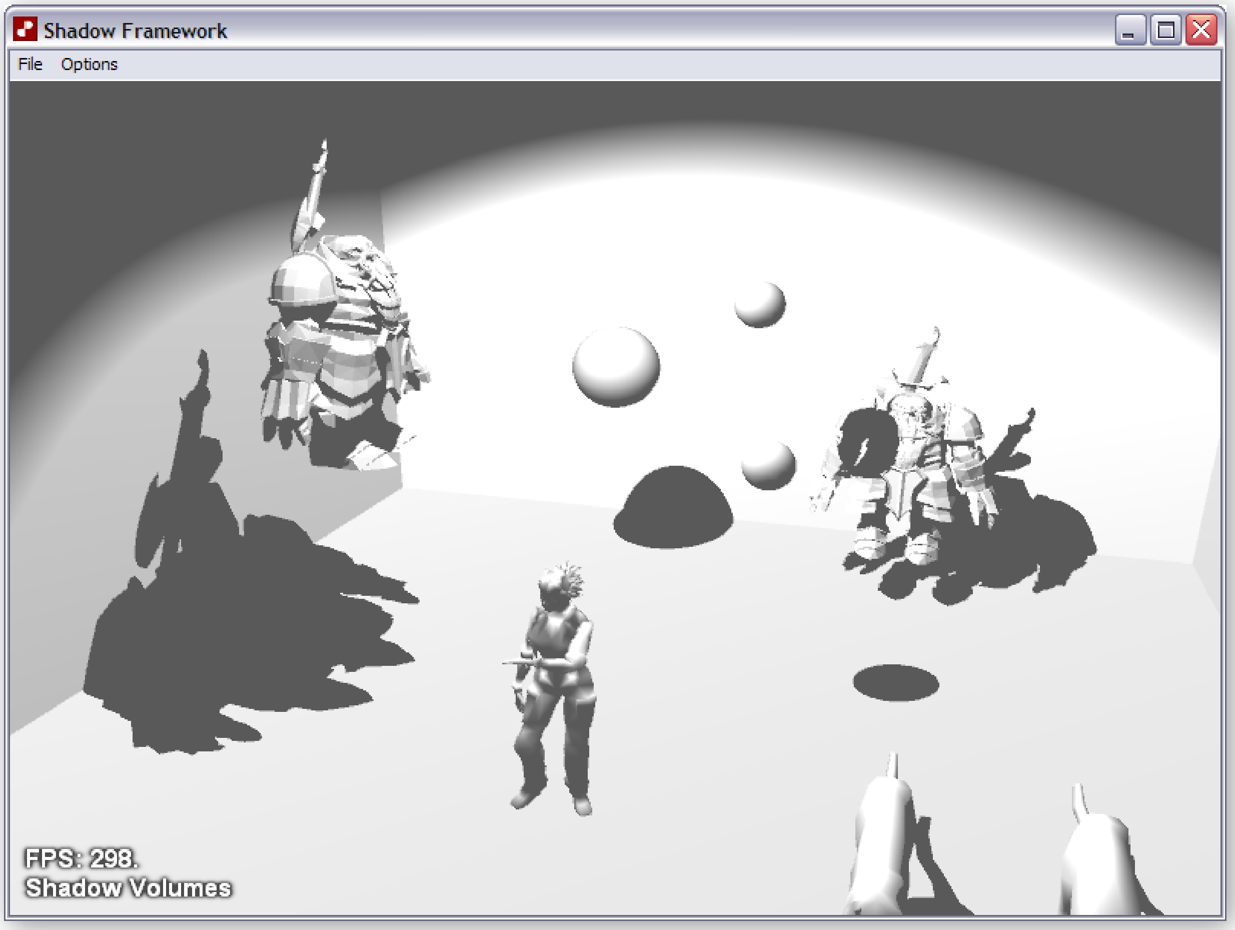


Figure 9.3.1: Shadow Volumes

9.4 Box Filtered Percentage Closer Filtering

All of the following percentage closer filtering implementations are based on the standard shadow mapping implementation described above. The implementations in this section are based on 9x9, 13x13, and 19x19 box filters. For each rendered pixel, the shadow map is sampled at 81 points (for a 9x9 filter), 169 points (for a 13x13 filter), and 361 points (for 19x19 filters) surrounding the corresponding texel in shadow map space. These samples form a square grid of texels centered at the texel in shadow map space that is being rendered. A standard shadow map test is performed using the pixel's depth and the depth of each of the samples, and the results are averaged together. Note that 9x9, 13x13, and 19x19 filters can produce a maximum of 82, 170, and 362 distinct values, respectively, for each shaded pixel. As a result, using fewer samples will result in a penumbra area with more banding artifacts where the transition between the levels of shading becomes sharper and more obvious, and more samples will produce smoother transitions and therefore softer penumbra. It should also be noted that larger filter kernels lead to a huge increase in the number of times the shadow map is sampled per frame. Considering the fact that texture fetches have a high latency, filtering techniques that sample the shadow map heavily can incur a very high runtime costs. In this implementation, the percentage closer filtering technique using a 19x19 box filter ran at only 21.5 frames per second, which put it at the bottom of the list in terms of performance.

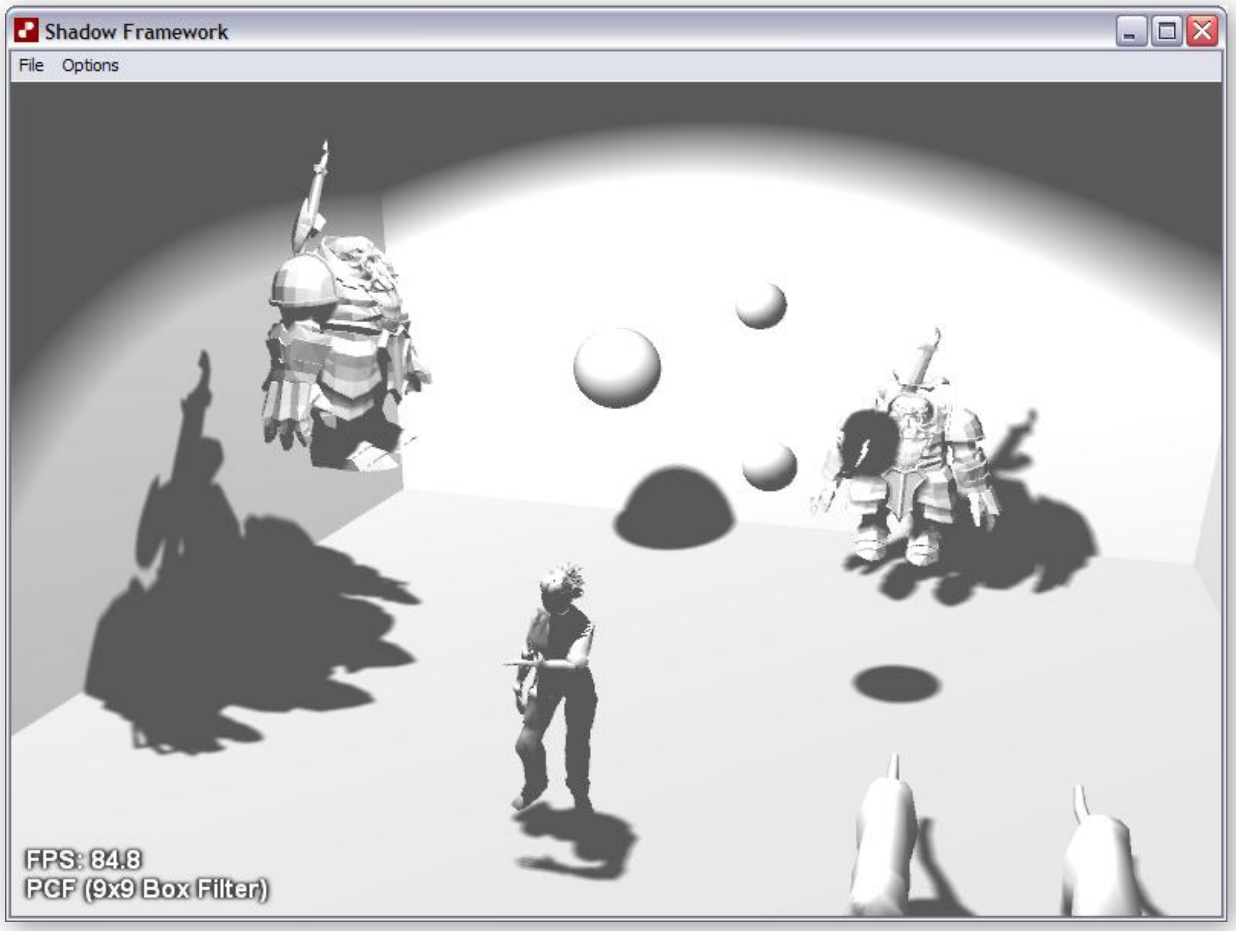
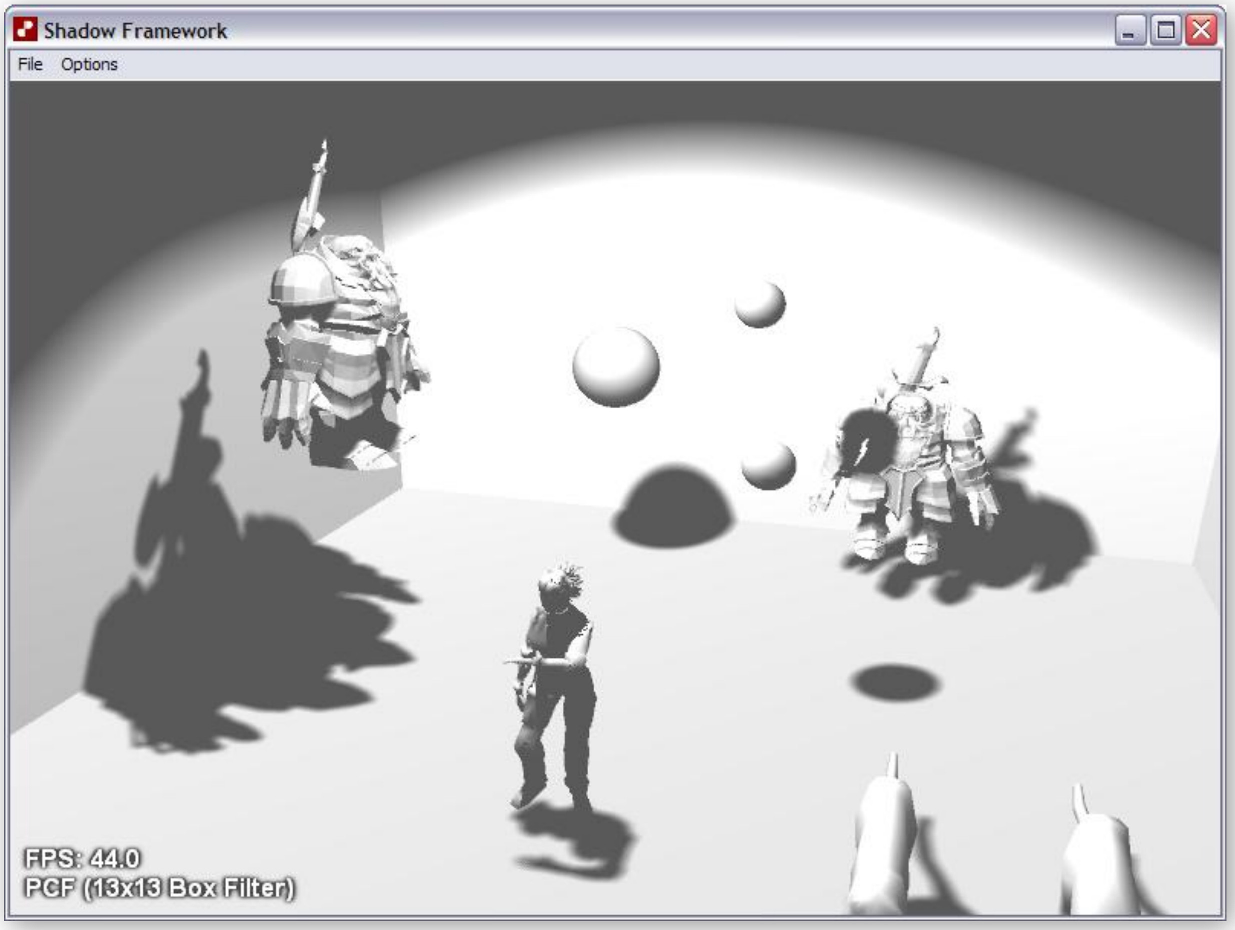
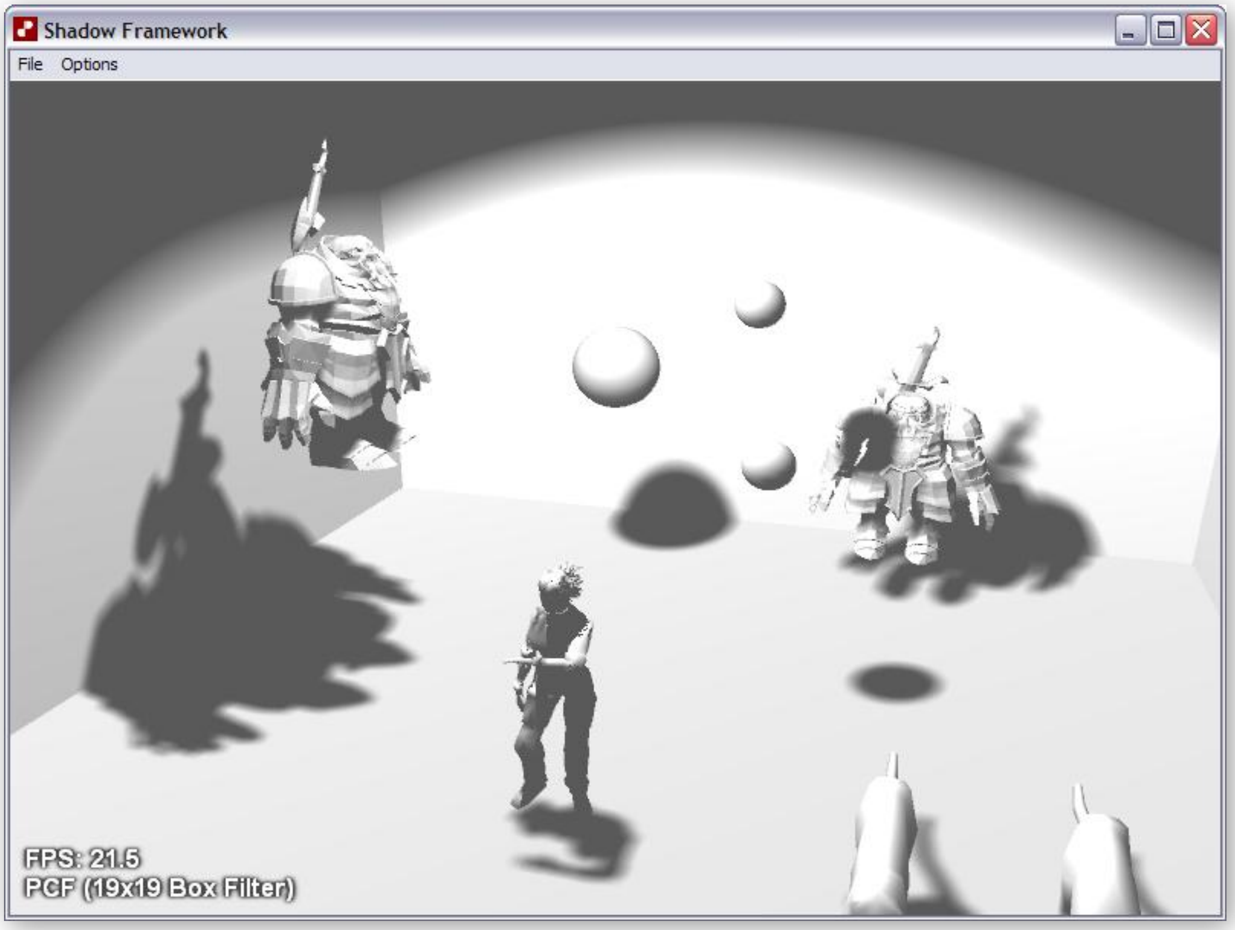


Figure 9.4.1: Percentage closer shadow map filtering, using a 9x9 box filter.



FPS: 44.0
PCF (13x13 Box Filter)

Figure 9.4.2: Percentage closer shadow map filtering, using a 13x13 box filter.



FPS: 21.5
PCF (19x19 Box Filter)

Figure 9.4.3: Percentage closer shadow map filtering, using a 19x19 box filter.

9.5 Gaussian Filtered Percentage Closer Filtering

Gaussian percentage closer filtering is much like percentage closer filtering using a box filter, in that the shadow map is sampled in a grid surrounding the corresponding shadow map texel and the sampled depths are used with the pixel depth in a depth test. However, instead of taking an average of all the results over the entire filter area, a Gaussian filter weights the results before averaging them such that points closer to the center of the filter kernel have more of an influence than points further away towards the edges. The Gaussian distribution can be visualized as a bell-shaped curve in two dimensions. Visually, the filtered result is much "rounder" than that of a box filter, and is typically more aesthetically pleasing.

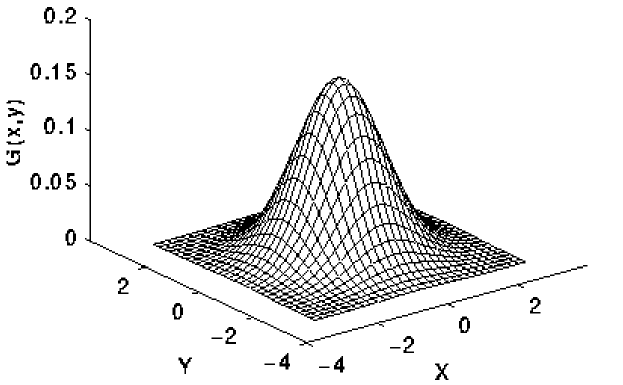
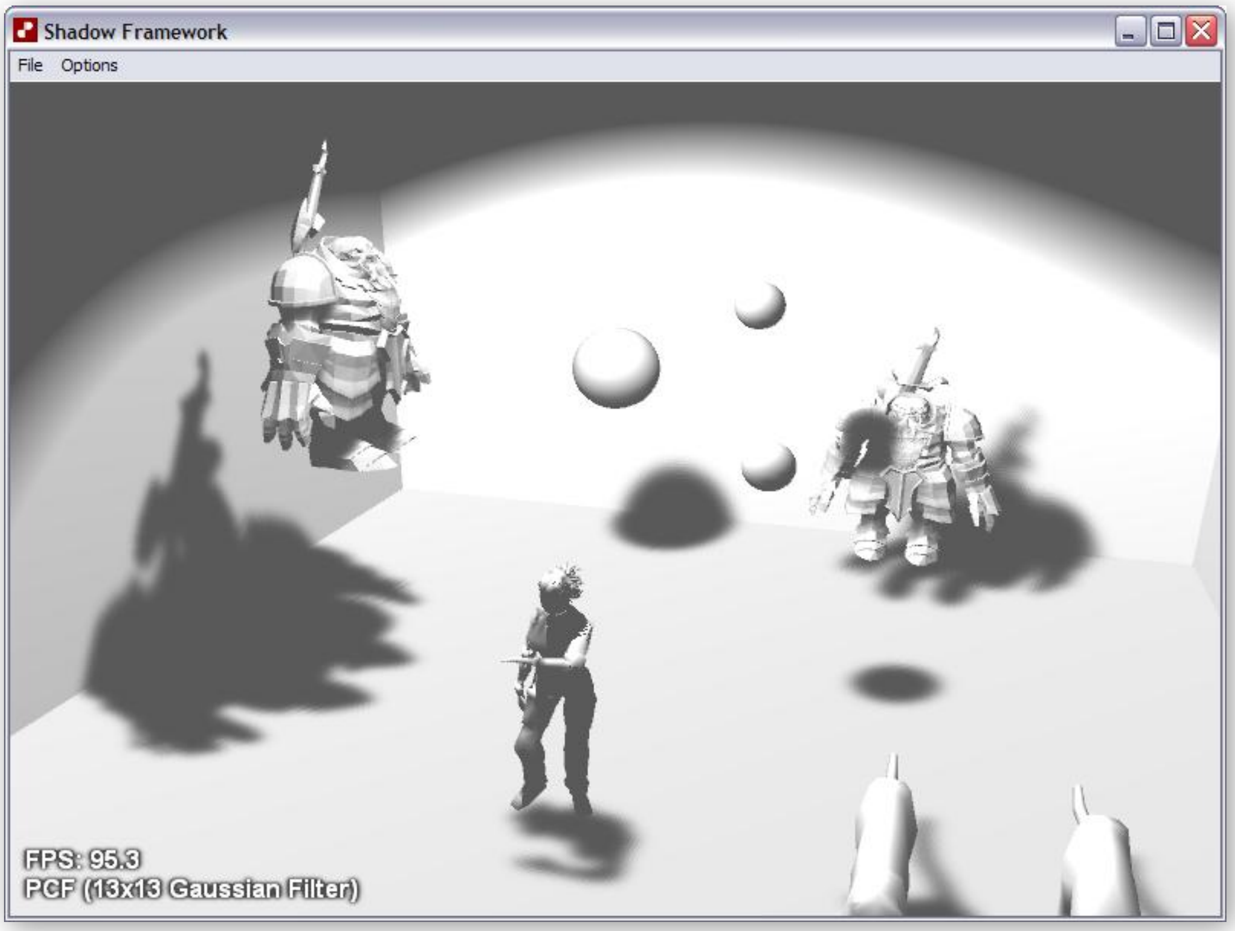


Figure 9.5.1: A Gaussian distribution in three dimensions

In this implementation, the Gaussian weights for each of the filter sizes were pre-computed offline in a separate application and stored in look-up tables which were available to each of the pixel shaders. The rendered shadows were more appealing than the shadows that had been filtered using box filters, which appeared somewhat more angular and "boxy." However, the shadows rendered using a Gaussian filter displayed penumbra that were not as wide as the penumbra rendered using a box filter of the same size, which is the result of the lower influence of the samples taken near the edges of the filter kernel. A wider Gaussian filter using more samples was therefore required to match the shape and penumbra width of a shadow rendered using a box filter of a smaller size.



FPS: 95.3
PCF (13x13 Gaussian Filter)

Figure 9.4.3: Percentage closer shadow map filtering, using a 13x13 Gaussian filter.

9.6 Variable Width Percentage Closer Filtering

The following technique is an implementation of the variable width percentage closer filtering technique as described in section 5.3.1. In this implementation, the initial blocker search is performed in a 3x3 filter kernel which is 64 texels wide. The final amount of shadowing per pixel is determined by a 5x5 box filter with a width given by equation 5.3.1.1. An optimization to the technique is that the initial blocker search also determines whether the pixel is fully lit, fully shadowed, or within the penumbra, and then only performs further sampling and filtering on the pixels within the penumbra area. This optimization can introduce artifacts in the form of fully lit pixels in the outer penumbra and fully shadowed pixels in the inner penumbra when the initial blocker search is incorrect. However, it can provide a significant performance boost when there are a relatively small number of penumbra pixels on screen, since the heavy shadow map sampling is restricted to a relatively small area.

It should be noted that this technique is the only technique implemented that displays correct shadow contact hardening. However, the disadvantage of this feature is that it becomes possible to orient the scene such that shadows are cast with very wide penumbra, and wide penumbra make the inevitable banding artifacts much more apparent.

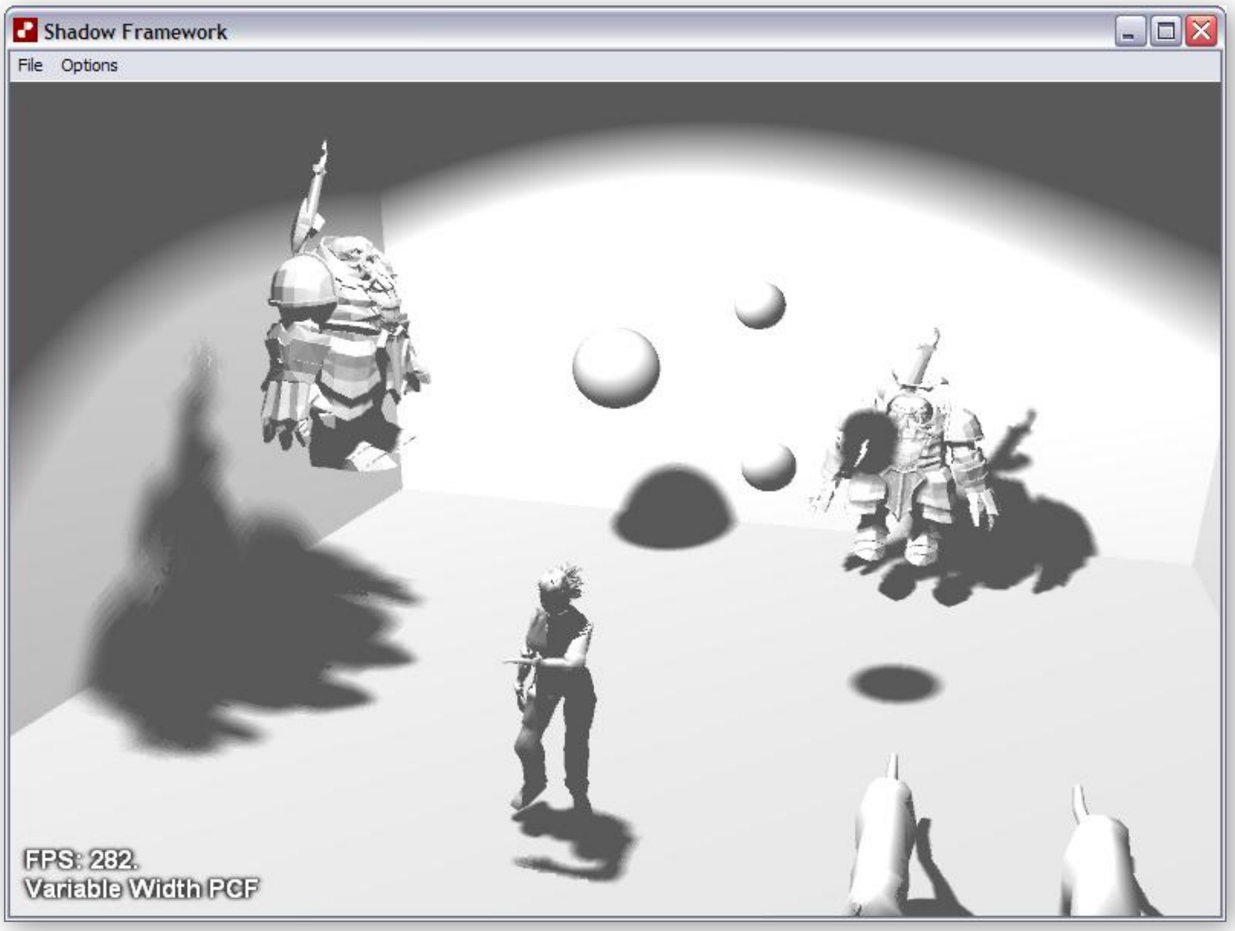


Figure 9.6.1: Percentage closer shadow map filtering, using a variable width filter kernel.

9.7 Jittered Percentage Closer Filtering

In the following technique, percentage closer filtering is performed using samples into the shadow map taken at random offsets. This approach effectively trades banding artifacts for noise artifacts, and in many situations (such as when the receiver's texture is relatively noisy as well) noise artifacts are much less noticeable to the human eye. The implementation here uses only 32 randomly offset shadow map samples per pixel and achieves a grainy but relatively smooth penumbra. The implementation uses the same optimization described in section 9.6, where in this case eight samples are taken to initially determine which pixels are within the penumbra and need further shadow map sampling. Additionally, the results from the initial set of samples are averaged into the second set of samples, which means that this first set of samples is not wasted.

In this implementation, the random offsets are pre-computed and stored in a three dimensional texture. This pre-computed texture is necessary because current generation hardware has no support for random number generators in any stage of the programmable pipeline. The third dimension in the offset texture reduces reoccurring patterns from appearing in the noise in the rendered shadows.

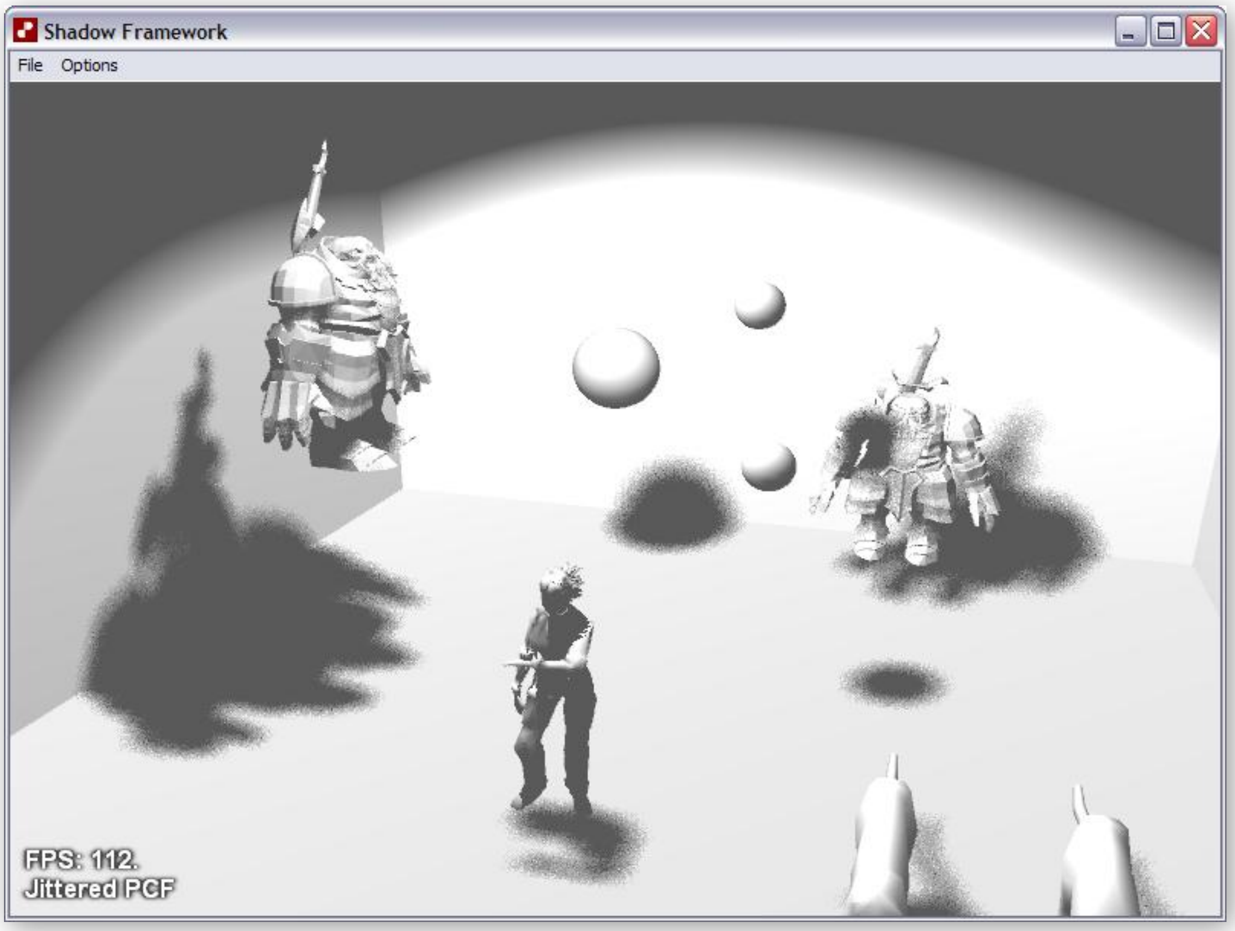
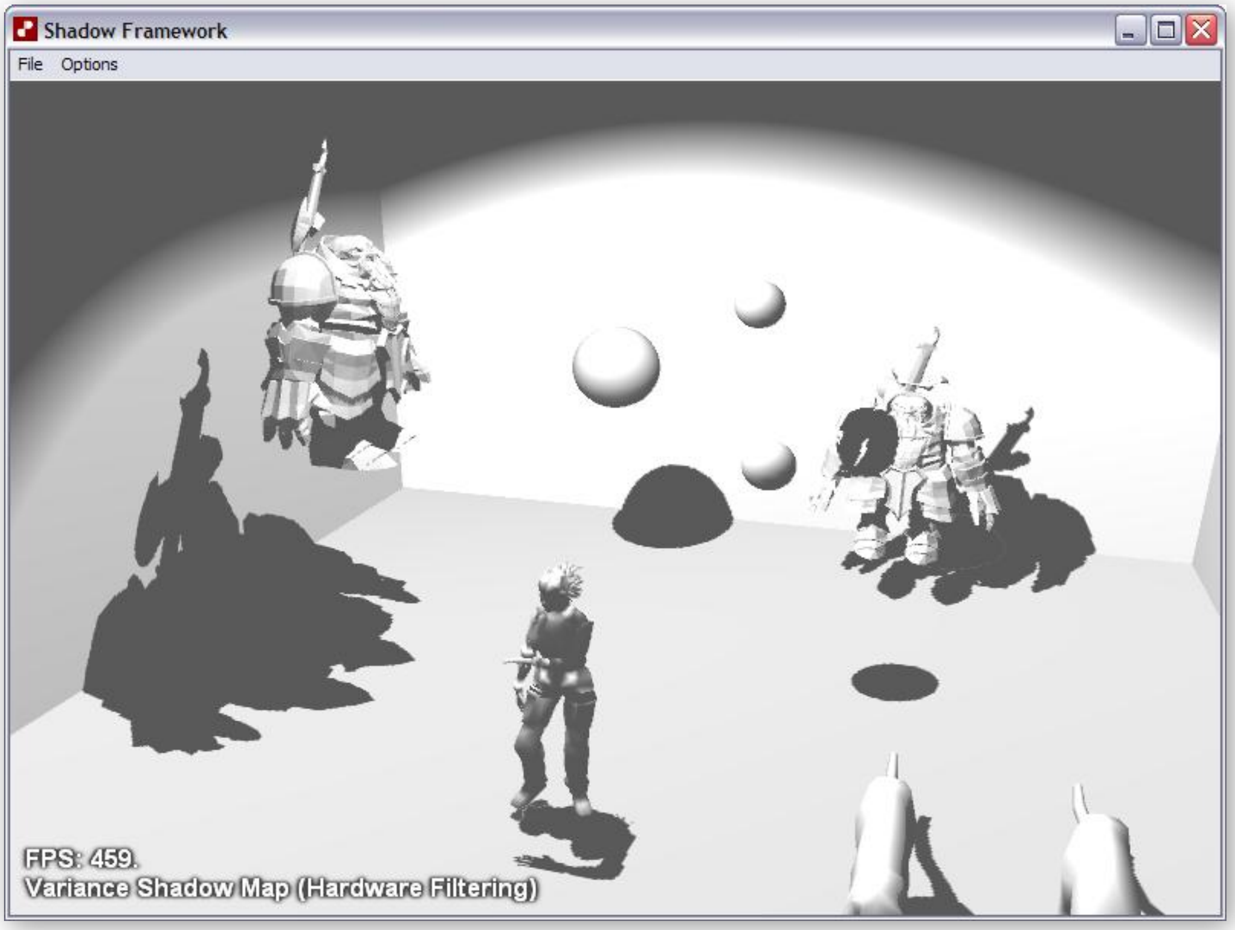


Figure 9.7.1: Percentage closer shadow map filtering, using jittered sample offsets.

9.8 Variance Shadow Maps (Unfiltered)

The following sections present several implementations of Variance Shadow Maps, which are described in section 5.4.1. In each of these implementations, the Variance Shadow Map is stored as a 64-bit floating point texture with two 32-bit components (one for depth and one for depth squared), and an additional texture is required for the pre-filtering stage. Pre-filtering requires two textures because the filtering passes cannot be performed in-place (reading from and writing to the same texture is not supported by current generation hardware). One texture needs to be set as the source and the second as the destination. In this implementation, the textures are 1024 x 1024 pixels instead of 2048 x 2048. The smaller texture size greatly improves the speed of the pre-filtering phase, and the artifacts introduced by the smaller texture size are hidden by the extra amount of softness in the shadows' edges.

The first implementation uses only hardware antialiasing when rendering the shadow map and hardware-based bilinear filtering when sampling from it. This implementation is provided as a reference.

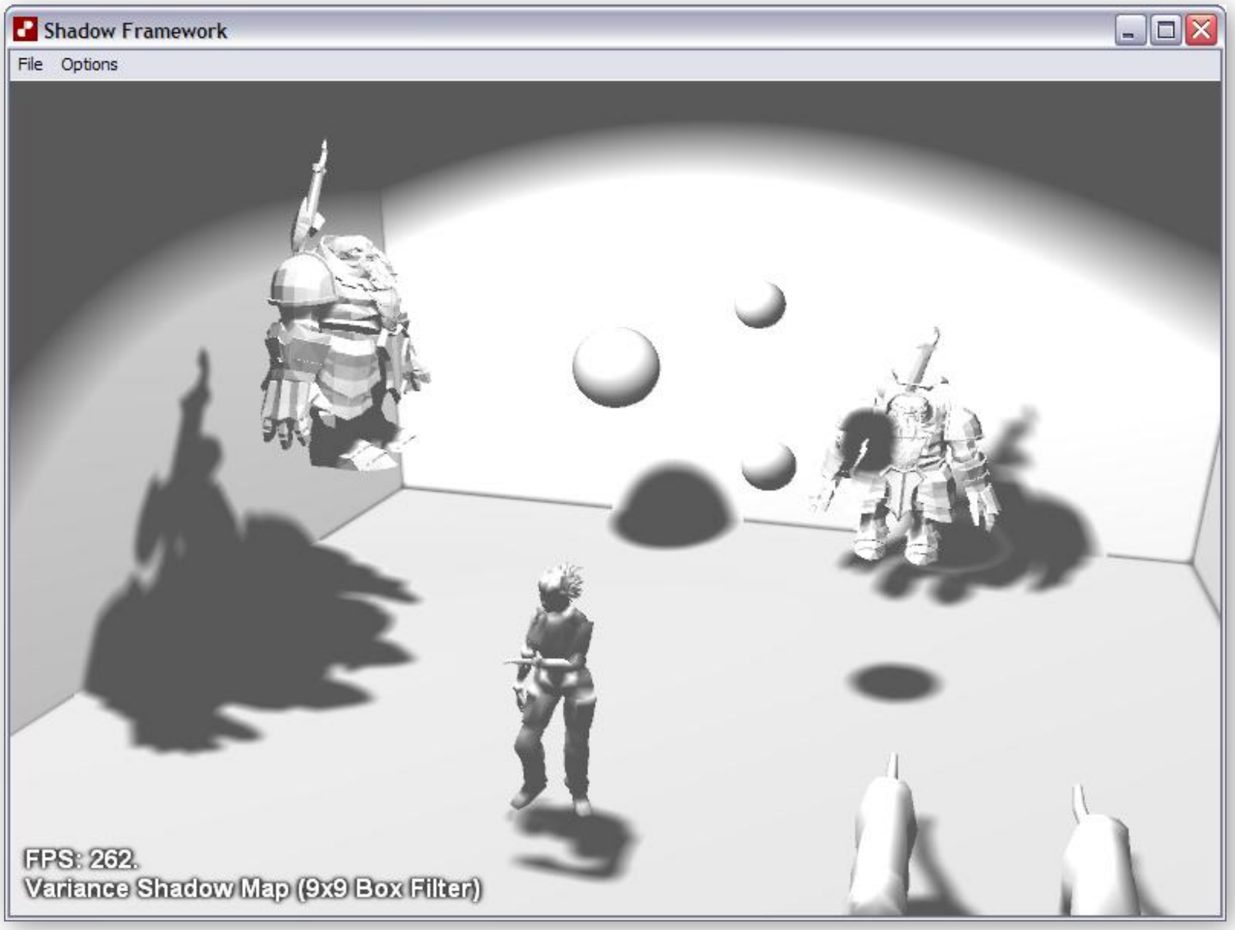


FPS: 459.
Variance Shadow Map (Hardware Filtering)

Figure 9.8.1: Variance shadow maps without pre-filtering.

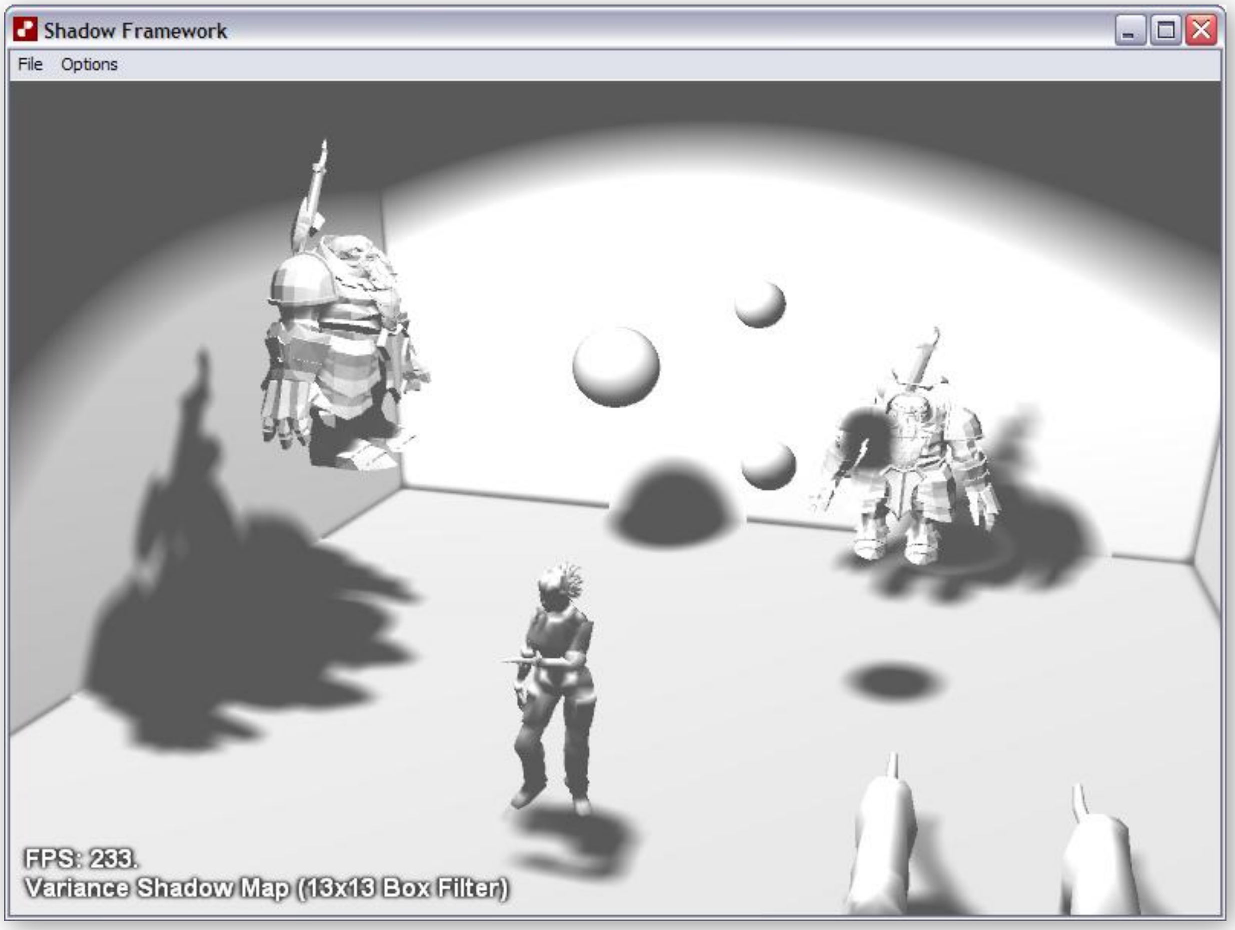
9.9 Box Filtered Variance Shadow Maps

The following images show scenes rendered using Variance Shadow Maps that were pre-filtered using a box filter. There are no workarounds for light bleeding, and light bleeding artifacts are readily apparent when shadows overlap.



FPS: 262.
Variance Shadow Map (9x9 Box Filter)

Figure 9.9.1: Variance Shadow Maps pre-filtered using a 9x9 box filter.



FPS: 233.
Variance Shadow Map (13x13 Box Filter)

Figure 9.9.2: Variance Shadow Maps pre-filtered using a 13x13 box filter.

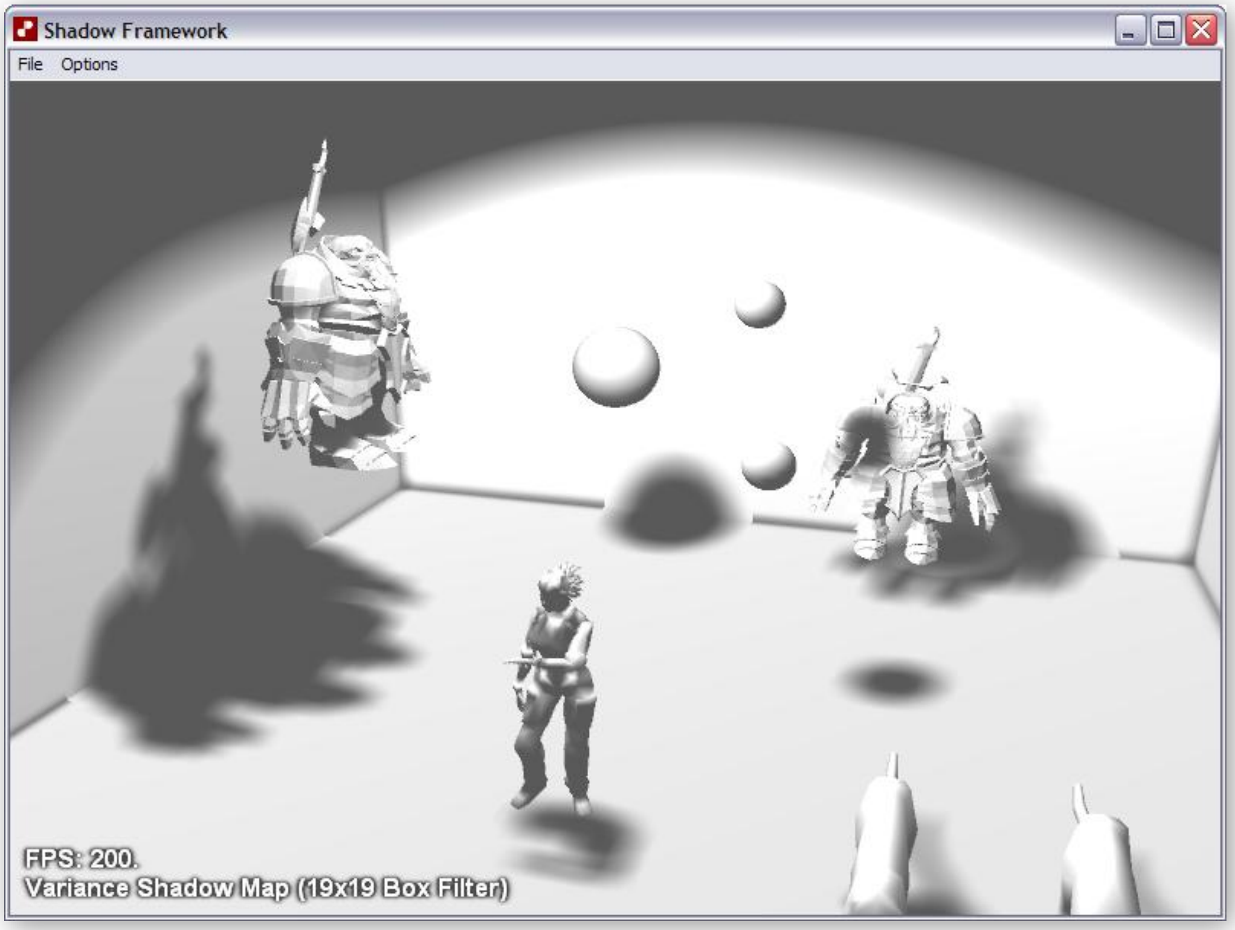


Figure 9.9.3: Variance Shadow Maps pre-filtered using a 19x19 box filter.

9.10 Gaussian Filtered Variance Shadow Maps

The following images show the scene rendered using Variance Shadow Maps that were pre-filtered using a Gaussian filter. As was the case with percentage closer filtering, the shadows filtered using Gaussian filters are more appealing than their box filter counterparts but have smaller penumbra widths.

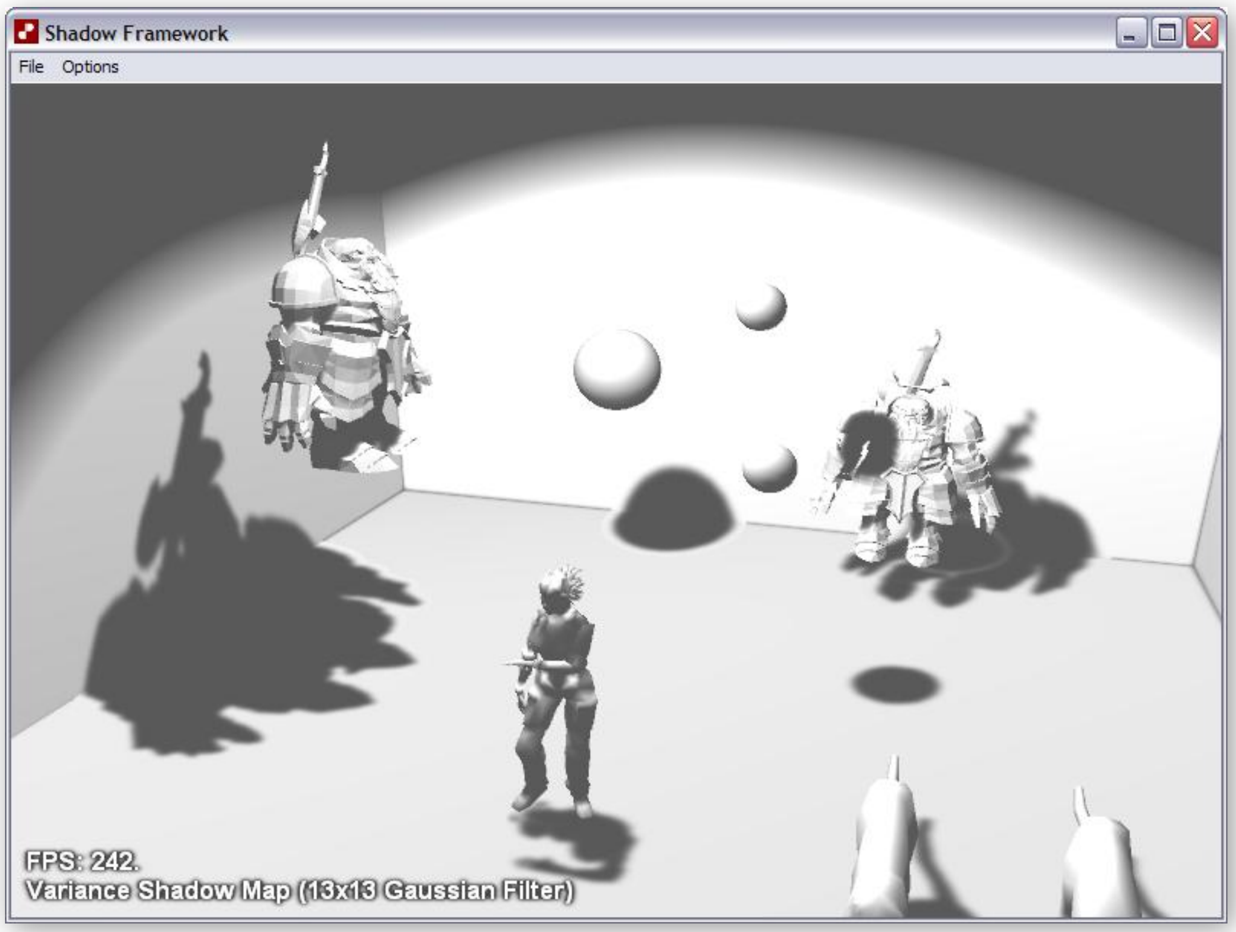


Figure 9.10.1: Variance Shadow Maps pre-filtered using a 13x13 Gaussian filter.

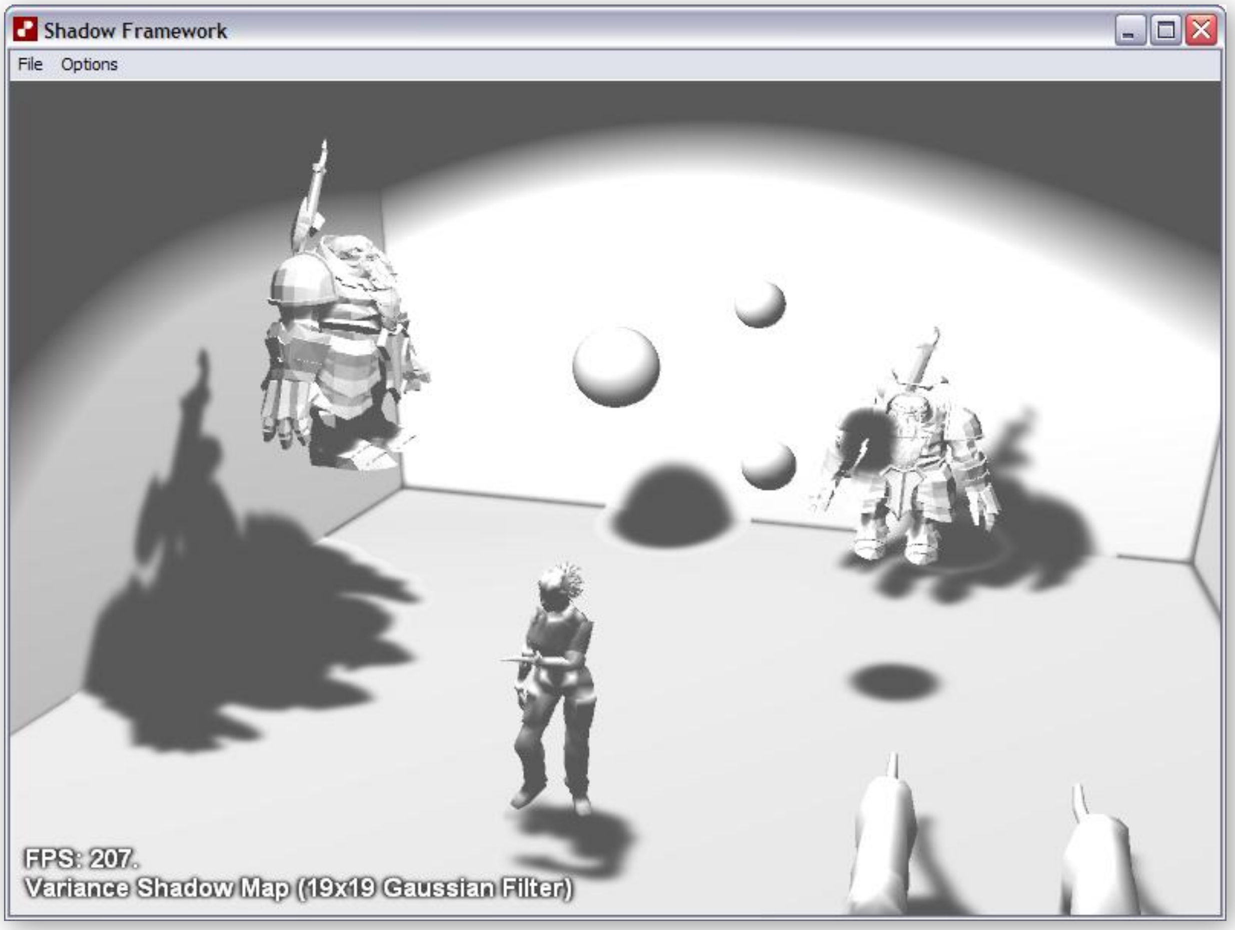
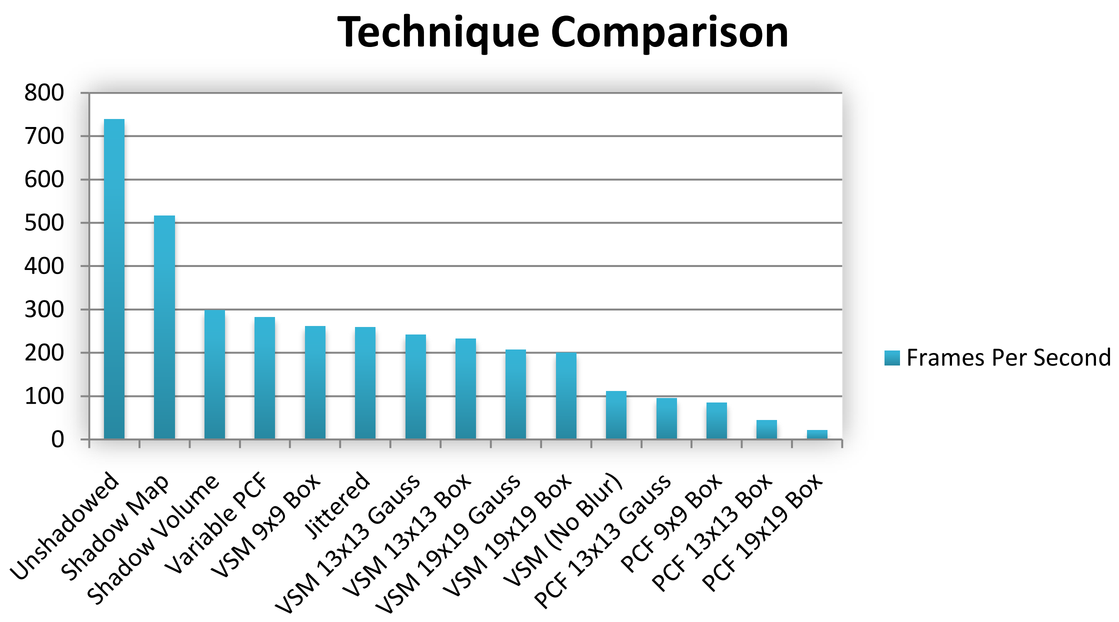


Figure 9.10.2: Variance Shadow Maps pre-filtered using a 19x19 Gaussian filter.

9.11 Graph of Rendering Time by Technique



References

[ADMA03] Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller. "An Optimized Soft Shadow Volume Algorithm with Real-Time Performance"

[ADMBSK08] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. "Real-Time, All-Frequency Shadows in Dynamic Scenes. "

[AHLHHS06] Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Charles Hansen, and François X. Sillion. "Soft Shadow Maps: Effcient Sampling of Light Source Visibility"

[AHT04] Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. "Approximate Soft Shadows with an Image-Space Flood-Fill Algorithm"

[AL04] Timo Aila and Samuli Laine. "Alias-Free Shadow Maps"

[Ali05] Timo Alia. "Efficient Algorithms for Occlusion Culling and Shadows"

[AM02] Ulf Assarsson and Tomas Akenine-Möller. "Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges"

[AM03] Ulf Assarsson and Tomas Akenine-Möller. "A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware"

[AM04a] Timo Alia and Tomas Akenine-Möller. "A Hierarchical Shadow Volume Algorithm"

[AM04b] Ulf Assarsson and Tomas Akenine-Möller. "Occlusion Culling and Z-Fail for Soft Shadow Volume Algorithms"

[AMBSK07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, Jan Kautz. "Convolution Shadow Maps"

[AMSFK08] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, Jan Kautz. "Exponential Shadow Maps"

[Ann02] Thomas Annen. "Advanced Shadow Map Parameterization"

[ARHL00] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. "Efficient Image-Based Methods for Rendering Soft Shadows"

[AW04] Jukka Arvo and Jan Westerholm. "Hardware Accelerated Soft-Shadows Using Penumbra Quads"

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. "Shadow Mapping for Hemispherical and Omnidirectional Light Sources"

[BCS08] Louis Bavoil, Steven P. Callahan, and Claudio T. Silva. "Robust Soft Shadow Mapping with Backprojection and Depth Peeling"

[Bli88] Jim Blinn, "Me and my (fake) shadow", IEEE Computer Graphics and Applications, 8(1), 82-6, Jan. 1988

[Bra03] Stefan Brabec. “Shadow Techniques for Interactive and Real-Time Applications”

[BS01] Stefan Brabec and Hans-Peter Seidel. “Hardware-Accelerated Rendering of Antialiased Shadows with Shadow Maps”

[BS02] Stefan Brabec and Hans-Peter Seidel. “Single Sample Soft Shadows Using Depth Maps”

[BS03] Stefan Brabec and Hans-Peter Seidel. “Shadow Volumes on Programmable Graphics Hardware”

[BS04] Ulf Borgenstam and Jonas Svensson. “A Soft Shadow Rendering Framework in OpenGL”

[Bun05] Michael Bunnell. “Dynamic Ambient Occlusion and Indirect Lighting”

[CD03] Eric Chan and Frédo Durand. “Rendering Fake Soft Shadows with Smoothies”

[CD04] Eric Chan and Frédo Durand. “An Efficient Hybrid Shadow Rendering Algorithm”

[CF92] Norman Chin and Steven Weiner. “Fast Object-Precision Shadow Generation for Area Light Sources Using BSP Trees”

[CG04] Hamilton Y. Chong and Steven J. Gortler. “A Lixel for every Pixel”

[Cro77] Franklin C. Crow. “Shadow algorithms for computer graphics.” In Proceedings of the 4th annual conference on Computer graphics and interactive techniques, volume 11, pages 242–248, July 1977.

[DL06] William Donnelly and Andrew Lauritzen. “Variance Shadow Maps”

[Dre94] George Drettakis. “A Fast Shadow Algorithm for Area Light Sources Using Backprojection”

[Dur00] Frédo Durand. “A Multidisciplinary Survey of Visibility”

[ED06] Elmar Eisemann and Xavier D´ecoret. “Plausible Image Based Soft Shadows Using Occlusion Textures”

[EK02] Cass Everitt and Mark J. Kilgard. “Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering.” Published online at http://developer.nvidia.com. 2002

[FBP06] Vincent Forest, Loïc Barthe, and Mathias Paulin. "Realistic Soft Shadows by Penumbra-Wedges Blending"

[Fer02] Randima Fernando. “Adaptive Techniques for Hardware Shadow Generation”

[Fer05] Randima Fernando. “Percentage-Closer Soft Shadows”

[FFBG01] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. “Adaptive shadow maps”

[FK03] Kasper Fauerby and Carsten Kjær. “Real-time Soft Shadows in a Game Engine”

[GBP06] Gaël Guennebaud, Loïc Barthe and Mathias Paulin. "Real-time Soft Shadow Mapping by Backprojection"

[GBP07] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. "High-Quality Adaptive Soft Shadow Mapping"

[GLYSM03] Naga K. Govindaraju, Brandon Lloyd, Sung-Eui Yoon, Avneesh Sud, and Dinesh Manocha. "Interactive Shadow Generation in Complex Environments"

[Gre03] Robin Green. "Spherical Harmonic Lighting: The Gritty Details"

[Hai01] Eric Haines. "Soft Planar Shadows Using Plateaus"

[HBS00] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. "Soft Shadow Maps for Linear Lights"

[Hec92] Paul S. Heckbert. "Discontinuity Meshing for Radiosity", 3rd Eurographics Workshop on Rendering, (Bristol 1992), pp. 203-216

[Hei91] T. Heidmann. "Real shadows, real time." Iris Universe, Silicon Graphics Inc., No. 18, 23–31. 1991

[Her97] Michael Herf. "Efficient Generation of Soft Shadow Textures." Technical Report CMU-CS-97-138, Carnegie Mellon University, 1997.

[HH97] Paul S. Heckbert and Michael Herf. "Simulating soft shadows with graphics hardware." Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.

[HHLH05] Samuel Hornus, Jared Hoberock, Sylvain Lefebvre, and John C. Hart. "ZP+: Correct Z-pass stencil shadows"

[HLHS03] J.M. Hasenfratz, M. Lapierre, N. Holzschuch and F.X. Sillion. "A Survey of Real-time Soft Shadows Algorithms"

[JMB04] Gregory S. Johnson, William R. Mark, and Christopher A. Burns. "The Irregular Z-Buffer and its Application to Shadow Mapping"

[KD03] Florian Kirsch and Jürgen Döllner. "Real-Time Soft Shadows Using a Single Light Sample"

[KL05] Janne Kontkanen and Samuli Laine. "Ambient Occlusion Fields"

[KM99] B. Keating and N. Max. "Shadow Penumbras for Complex Objects by Depth-Dependent Filtering of Multi-Layer Depth Images"

[Koz04] Simon Kozlov. "Perspective Shadow Maps: Care and Feeding"

[Kwo02] Hun Yen Kwoon. "The Theory of Stencil Shadow Volumes." http://www.gamedev.net/reference/articles/article1873.asp, December 3, 2002

[Lai05] Samuli Laine. "Split-Plane Shadow Volumes"

[Len02] Eric Lengyel. "The Mechanics of Robust Stencil Shadows." http://www.gamasutra.com/features/20021011/lengyel_pfv.htm. October 11, 2002.

[Li05] Xin Li. Course notes on shadow rendering. 2005.

[LLA06] Jaakko Lehtinen, Samuli Laine, and Timo Aila. “An Improved Physically-Based Soft Shadow Volume Algorithm”

[LM08] Andrew Lauritzen and Michael McCool. “Layered Variance Shadow Maps.”

[LTG92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. “Discontinuity meshing for accurate radiosity.” IEEE Computer Graphics and Applications, 12(6): 25-39, Nov. 1992.

[LV00] Tom Lokovic and Eric Veach. “Deep Shadow Maps”

[Mcc00] Michael D. McCool. “Shadow Volume Reconstruction from Depth Maps”

[MT04] Tobias Martin and Tiow-Seng Tan. “Anti-aliasing and Continuity with Trapezoidal Shadow Maps”

[Nea02] Andrew V. Nealen. “Shadow Mapping and Shadow Volumes: Recent Developments in Real-Time Shadow Rendering”

[NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. “All-Frequency Shadows Using Non-linear Wavelet Lighting Approximation”

[OF99] Marc J. Ouellette and Eugene Fiume. “Approximating the Location of Integrand Discontinuities for Penumbral Illumination Computation with Area Light Sources”

[PSS98] Steven Parker, Peter Shirley, and Brian Smits. “Single Sample Soft Shadows”

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. “Rendering antialiased shadows with depth maps.” SIGGRAPH ’87

[RT06] Guodong Rong and Tiow-Seng Tan. “Utilizing Jump Flooding in Image-based Soft Shadows.”

[RWSZLSSBPG06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. "Real-time Soft Shadows in Dynamic Scenes using Spherical Harmonic Exponentiation"

[SCH03] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. “Shadow Silhouette Maps”

[SD02] Marc Stamminger and George Drettakis. “Perspective Shadow Maps”

[SGNS07] Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. "Image-Based Proxy Accumulation for Real-Time Soft Global Illumination"

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments”

[SS07] Michael Schwarz and Marc Stamminger. "Bitmask Soft Shadows"

[SS08] Michael Schwarz and Marc Stamminger. "Quality Scalability of Soft Shadow Mapping"

[SS98] Cyril Soler and François X. Sillion. “Fast Calculation of Soft Shadow Textures Using Convolution”

[Ste] James Stewart. http://www.cs.queensu.ca/home/jstewart/shadows/

[Ura05] Yury Uralsky. “Efficient Soft-Edged Shadows Using Pixel Shader Branching”

[WE03] D. Weiskopf and T. Ertl. “Shadow Mapping Based on Dual Depth Layers”

[WH03] Chris Wyman and Charles Hansen. “Penumbra Maps: Approximate Soft Shadows in Real-Time”

[Wil78] Lance Williams. “Casting Curved Shadows on Curved Surfaces”

[Woo92] Andrew Woo. Graphics Gems III, chapter The Shadow Depth Map Revisited, pages 338–342. Academic Press, 1992.

[WPF90] Andrew Woo, Pierre Poulin, and Alain Fournier. “A Survey of Shadow Algorithms”

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. “Light Space Perspective Shadow Maps”

[YTD02] Zhengming Ying, Min Tang, and Jinxiang Dong. “Soft Shadow Maps for Area Light by Area Approximation”

[Zha98] Hansong Zhang. “Forward Shadow Mapping”

[ZHLGS05] Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. “Precomputed Shadow Fields for Dynamic Scenes”