Rendering Antialiased Shadows using Warped Variance Shadow Maps

by

Andrew Timothy Lauritzen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Shadows contribute significantly to the perceived realism of an image, and provide an important depth cue. Rendering high quality, antialiased shadows efficiently is a difficult problem. To antialias shadows, it is necessary to compute partial visibilities, but computing these visibilities using existing approaches is often too slow for interactive applications.

Shadow maps are a widely used technique for real-time shadow rendering. One major drawback of shadow maps is aliasing, because the shadow map data cannot be filtered in the same way as colour textures.

In this thesis, I present *variance shadow maps* (VSMs). Variance shadow maps use a linear representation of the depth distributions in the shadow map, which enables the use of standard linear texture filtering algorithms. Thus VSMs can address the problem of shadow aliasing using the same highly-tuned mechanisms that are available for colour images. Given the mean and variance of the depth distribution, Chebyshev's inequality provides an upper bound on the fraction of a shaded fragment that is occluded, and I show that this bound often provides a good approximation to the true partial occlusion.

For more difficult cases, I show that warping the depth distribution can produce multiple bounds, some tighter than others. Based on this insight, I present *layered variance shadow maps*, a scalable generalization of variance shadow maps that partitions the depth distribution into multiple segments. This reduces or eliminates an artifact — "light bleeding" — that can appear when using the simpler version of variance shadow maps. Additionally, I demonstrate *exponential variance shadow maps*, which combine moments computed from two exponentially-warped depth distributions. Using this approach, high quality results are produced at a fraction of the storage cost of layered variance shadow maps.

These algorithms are easy to implement on current graphics hardware and provide efficient, scalable solutions to the problem of shadow map aliasing.

Acknowledgements

I would like to thank my supervisor Michael McCool for sharing his wealth of knowledge, and for supporting me in whatever research I am doing. Thanks also to William Donnelly for providing a continual stream of useful ideas and suggestions. Additionally, I'd like to thank the members of the Computer Graphics Lab, who never fail to make the lab a fun and interesting place to talk and exchange ideas.

I extend a huge thanks to my good friend Chris Iacobucci, who is always willing to create 3D models and textures to show off whatever algorithm I'm working on. Without his contributions, there would be many more cubes and spheres in the example scenes!

Finally I'd like to thank my wife, Emily, for always supporting and encouraging me, and tolerating far too many late nights leading up to various deadlines.

This thesis is a summary of the following publications, coauthored by William Donnelly and Michael McCool:

- **Layered Variance Shadow Maps**, Andrew Lauritzen, Michael McCool, to appear in Graphics Interface 2008.
- **Summed-Area Variance Shadow Maps**, Andrew Lauritzen, in GPU Gems 3, Addison-Wesley, pages 157-182, 2007. [10]
- **Variance Shadow Maps**, William Donnelly, Andrew Lauritzen, Proceedings of I3D 2006, ACM Press, pages 161–165, 2006. [5]

This work was supported in part by the Ontario Centres of Excellence (OCE), ATI/AMD, the Natural Sciences and Engineering Research Council (NSERC) and the Ontario Graduate Scholarship program (OGS).

Contents

List of Figures

Chapter 1

Introduction

Rendering is the process of sampling a scene over a rectilinear grid of pixels. Rasterization performs this sampling by projecting the scene primitives (usually triangles) onto the viewing plane and drawing them directly, using a depth buffer to cull hidden pixels. These primitives are lit and shaded, and then written to the framebuffer for display on the screen. Lighting and shadows are important effects to capture for realistic rendering. This thesis will concentrate on shadows.

Rendering high-quality shadows at interactive rates is a difficult problem in computer graphics. While tracing rays can produce physically correct shadows, the algorithm is currently too slow for real-time applications like games. This is particularly true for soft-edged shadows, which require many shadow rays.

Conversely, current real-time shadow algorithms have significant quality, storage and usability issues. Shadow volumes [3] construct and rasterize explicit silhouette geometry to enclose the shadowed regions and thus are inefficient when dealing with complex, animated 3D models such as characters. It is also difficult to use shadow volumes to represent non-polygonal data such as tree leaves, which are often modelled using texture-based transparency (alpha) masks. Furthermore, because shadow volumes are rasterized in screen space, they cannot be queried arbitrarily, making them incompatible with techniques like per-pixel displacement mapping.

Shadow maps [19] have recently become popular for real-time shadow rendering. Being an image-space algorithm, shadow maps do not suffer from the issues faced by shadow volumes. In particular, shadow maps can be used with anything that can be rasterized into a depth map (including alpha-tested or displaced geometry) and are much less sensitive to the geometric complexity of the scene. They can also be queried arbitrarily, just like standard textures. Unfortunately, shadow maps suffer from aliasing artifacts due to mismatches in sampling rate between the shadow map and the framebuffer.

Magnification aliasing occurs in shadow maps due to oversampling, i.e., when the shadow map texels (pixels in the texture holding the shadow data) are spread over a large region in screen space. Using the obvious light space projection, the shadow

map resolution in camera space can be arbitrarily bad (as with any projective texturing technique). Single texels in the shadow map can be enlarged into huge blocks as they are projected into camera space. The edges of these blocks then flicker and crawl as the shadows move, which is distracting and unnatural. Addressing this form of aliasing involves improving the shadow map projection relative to the camera.

There has been a large amount of recent work in the area of light projection optimization. Perspective shadow maps [18], light space perspective shadow maps [21], trapezoidal shadow maps [15] and logarithmic shadow maps [12] all aim to eliminate magnification artifacts by adjusting the shadow projection so that its resolution is more uniform in camera space. Adaptive shadow maps [8] and resolution-matched shadow maps [11] attack the problem more directly by detecting oversampling in screen space and dynamically generating more shadow map resolution where necessary. Parallel-split shadow maps [22] partition the camera frustum into different regions and render a shadow map for each one, allocating more resolution to areas closer to the camera. For a discussion of the various tradeoffs between warping and partitioning algorithms, please refer to Lloyd et al.'s summary paper [13]. The techniques described in this thesis are complementary to projection and resolution management and can be used in conjunction with any of these techniques.

Shadow maps also suffer from minification aliasing due to undersampling, i.e., when adjacent pixels in screen space map to widely separated shadow map texels. In regions where one framebuffer pixel covers multiple pixels in the shadow map, a single texture sample is insufficient and produces aliasing, as with standard texture mapping. Several techniques are available to address this problem for linear colour textures, but all of these techniques depend on linear filtering. In linear filtering algorithms, a weighted average is computed over the entire region covered by each rendered pixel, suppressing the high frequency content that causes minification aliasing. Normally the performance of linear filtering is enhanced with preprocessed representations, such as image pyramids [20] or summed-area tables [4], that also depend on linearity. Unfortunately, the assumption of linearity is broken by the standard shadow map algorithm, which requires a depth comparison (a step function) per texture sample.

Percentage closer filtering (PCF) [16] provides a way to filter shadow maps by observing that we should average the results of many depth comparisons over a filter region, rather than the depth values themselves. This algorithm does not support prefiltering, and so for large filter sizes it is expensive. Additionally, PCF does not consider the receiver geometry within the filter region, which causes significant biasing and self-shadowing problems.

Deep shadow maps [14] store a distribution of depths for each shadow map texel. This representation requires a variable amount of storage per pixel, making deep shadow maps unsuitable for implementation on current graphics hardware. Furthermore averaging two distributions is nontrivial, which complicates the use of prefiltering approaches.

Convolution shadow maps [1] (CSMs) represent the visibility function with respect to a basis to allow linear prefiltering. CSMs are limited in their ability to represent discontinuities in the visibility function, and thus suffer from light bleeding near all occluders. Properly rendering any reasonably sized scene requires a huge number of coefficients, which makes CSMs impractical for real-time rendering of complex scenes. Furthermore, convolution shadow maps need to process all of the basis coefficients for a given shadow map texel on every access, which scales poorly as the number of coefficients increases.

In this thesis I present variance shadow maps (VSMs), an alternative, linear representation of shadow map data that allows minification aliasing to be addressed with standard texture prefiltering techniques. Projection optimization and resolution management are fully compatible and can be used simultaneously with the techniques presented in this thesis to address both types of shadow map aliasing.

VSMs represent the depth distribution over arbitrary filter regions using *moments*, which can be filtered and averaged freely. Chebyshev's inequality provides an upper bound on the visibility of a given fragment over an arbitrary filter region, which is used to estimate the partial visibility during rendering. This upper bound is relatively tight in most cases, but occasionally provides a bad approximation, producing light bleeding artifacts. I discuss the source of these artifacts and ways to combat them by warping the depth function. Warping the depth function enables the use of multiple estimators, and I present two strategies based on these multiple estimators: layered and exponential warps. Layering preserves constant access time independent of the number of estimators while allowing the bounds to be scalably improved by using more layers. Exponential warps improve the bounds so significantly that in many cases only two estimators need to be combined to eliminate most light bleeding, even in difficult scenes.

In summary, the techniques in this thesis achieve high-quality shadows with excellent performance, making these algorithms suitable for inclusion in modern games and other real-time rendering applications.

Chapter 2

Background

In this chapter I review the shadow mapping algorithm, which is the basis for the work in this thesis. I also discuss percentage closer filtering, which is currently the most common method used to filter shadow maps.

2.1 Shadow Maps

Shadow maps were introduced by Lance Williams in 1978 [19], and remain one of the most popular techniques for rendering real-time shadows. The algorithm works as follows:

1. Render the scene from the point of view of the light to generate a depth buffer (colour data is not needed).

2. Render the scene from the camera's point of view. While shading, project fragments into light space and compare the depth of the fragment (in light space) to the depth stored in the shadow map. If the fragment depth is greater than the stored depth, the fragment is shadowed. Otherwise, it is lit.

Figure 2.1 shows a visual representation of the shadow mapping algorithm. This example shows a directional light with parallel rays (vertical dashed lines). The solid points on these lines represent the occluder depths that are rasterized into the shadow map in Step 1. Two example eye rays are shown (diagonal dotted lines), resulting in fragments generated at the open circles. In Step 2, the left fragment would be considered to be lit since the fragment depth in light space (open circle) is equal to the shadow map depth (solid point). Conversely, the right fragment is unlit, because the fragment depth is greater than the shadow map depth.

In practice, fragments will not fall exactly on the light view sample locations, and a continuous version of the depth map needs to be reconstructed. Due to lack of linearity, linear interpolation does not work well, so usually a nearest neighbour lookup is performed.

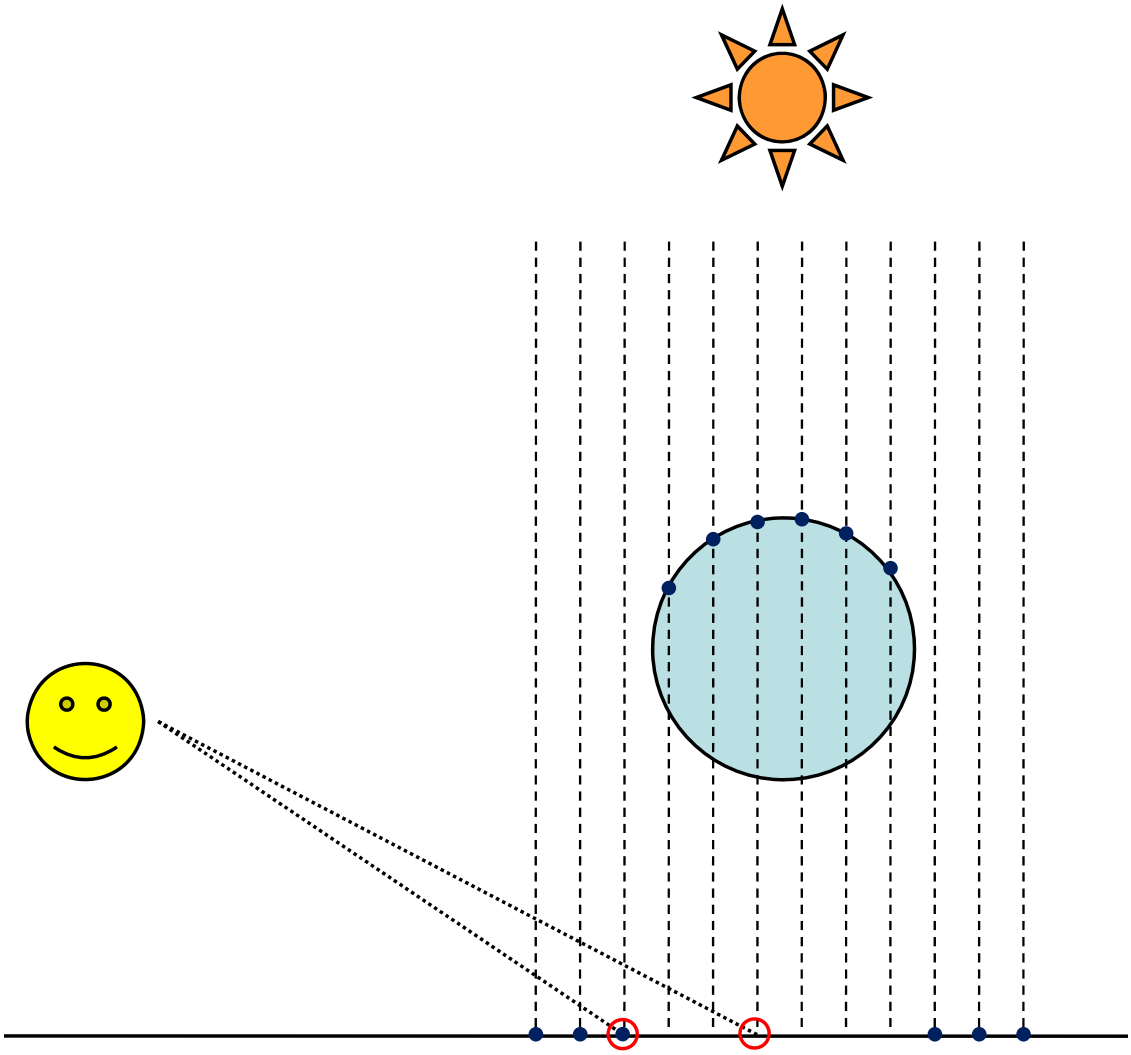


Figure 2.1: An illustration of the shadow mapping algorithm.

As discussed in Chapter 1, shadow maps are prone to aliasing from a variety of sources. For instance, a rendering of Figure 2.1 will have minification artifacts if the example eye rays are adjacent in the framebuffer. The two shadow rays (vertical dashed lines) between the shaded fragaments (open circles) are completely ignored by the shadow mapping algorithm, leading to the usual aliasing problems of point-sampled textures.

For example, in a static scene the shadows of thin objects may include periodic gaps. When the view is animated the problem is much worse, as the shadow cast by a small object may flash on and off repeatedly as the gaps in the sampling pattern sweep through the shadow map. Such flashing artifacts are distracting for the viewer and destroy the illusion of reality. Unfortunately, shadow volumes and ray sampling (when using a single ray per pixel) have the same problem.

Figure 2.2 shows an example of this artifact in practice. All of the shadow aliasing in this image is due to undersampling, and thus shadow volumes or single-sample ray-traced shadows would look the same as the top image. In contrast, the bottom image is rendered using the techniques described in this thesis, resulting in properly antialiased shadows.

2.2 Texture Filtering

Shadow maps are a form of projective texturing, wherein the depths of occluders in light space are projected onto the scene and compared to the depths of potential receivers. As with other projective texture mapping techniques, this process is prone to aliasing if the texture is not filtered properly.

Heckbert describes the problem as one of resampling the texture image onto the grid of framebuffer pixels [9]. If simple, nearest-neighbour resampling is used, undersampled frequencies in the texture will produce aliasing, as can be seen in Figure 2.3. This problem can be resolved by prefiltering the texture to produce band-limited versions that can be selected appropriately while shading to match the framebuffer sampling frequency.

To give a simple example, consider a texture mapped onto on a surface parallel to the viewing plane. If this plane is at a distance that produces a one-to-one mapping of texture pixels to framebuffer pixels then no resampling is necessary and the texture will be reproduced without aliasing. However, if the camera is pulled back so that the entire texture occupies only one framebuffer pixel, that pixel should be coloured using the average colour of the entire texture (i.e., all frequencies except the lowest have been removed) rather than choosing any one pixel from the texture.

Mipmapping [20] is one technique that efficiently addresses this problem by precomputing several "levels" of detail, with data that corresponds to averages over square regions in the texture. Modern graphics cards provide a hardware

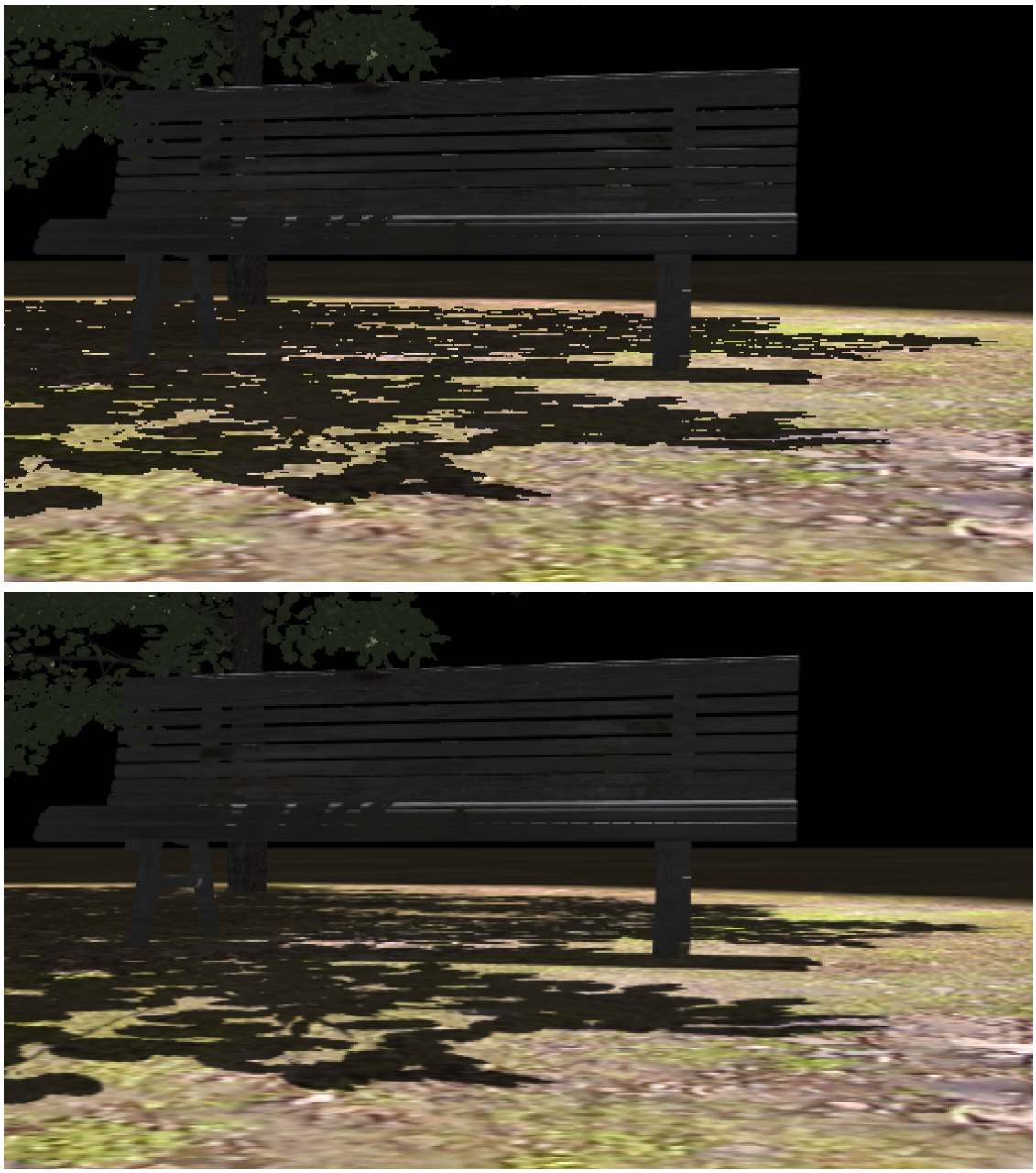


Figure 2.2: Minification aliasing with standard shadow maps (top) is solved by properly filtering the shadow map (bottom), as described in this thesis.

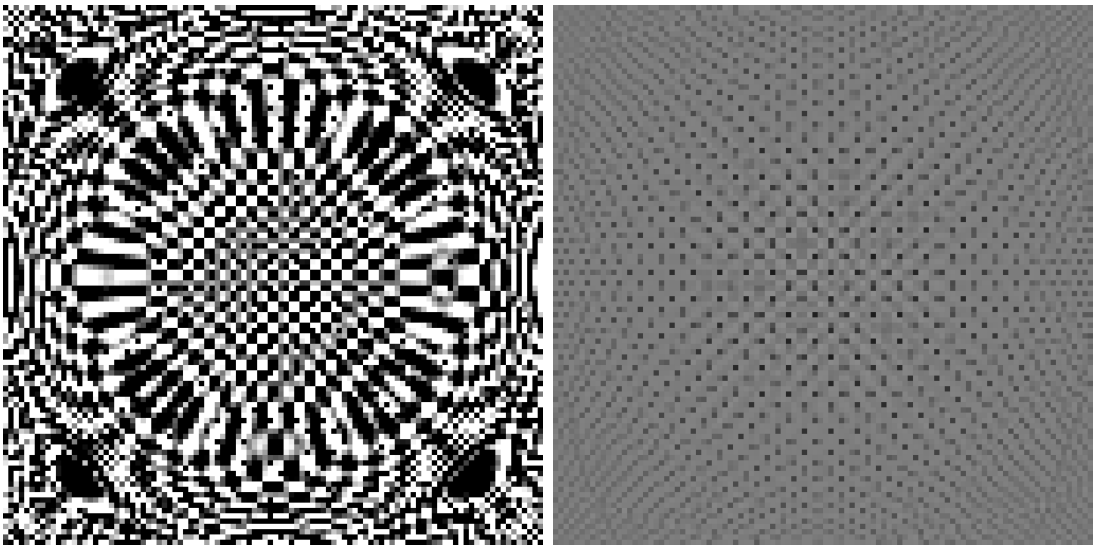


Figure 2.3: Nearest-neighbour sampling of a checkerboard-like texture warped by a fisheye projection produces significant aliasing artifacts due to undersampling (left). Properly filtering the texture produces an antialiased image (right).

implementation of mipmapping. By using forward differencing to compute the derivatives of the texture coordinates, the hardware can approximate the filter region in texture space that needs to be averaged. From the dimensions of this footprint, a mipmap level is selected and sampled appropriately.

It would be convenient to use these standard and optimized techniques to antialias shadow maps. However, because the shadow mapping algorithm requires a nonlinear depth comparison for each texture sample (Step 2), texture prefiltering cannot be used. Fundamentally, applying the depth test to an averaged value over some region of the texture will not produce the same result as applying the depth test to each of the samples in the region individually, and then averaging those results.

2.3 Percentage Closer Filtering

Applying linear filtering approaches to the depth samples used in shadow maps does not work. Percentage closer filtering [16] (PCF) provides a solution to the problem of shadow map filtering, based on the key insight that we actually want to average the results of the depth comparisons, rather than the samples themselves. The standard PCF algorithm does this directly using super-sampling.

The algorithm proceeds as follows: first, determine the filter region by projecting the current screen-space pixel extents onto the shadow map (similar to standard texture filtering). Then by taking many samples over the entire filter region and comparing each sampled depth to the reference depth, compute the percentage of

samples that pass the depth test using numerical integration. This provides the necessary information to attenuate the light reaching the pixel that is currently being shaded.

The visual results of percentage closer filtering are quite good, producing shadows similar to those seen at the bottom of Figure 2.2. Unfortunately, many samples are required to achieve these results: the scene in Figure 2.2 runs at less than 10 frames/second when using PCF compared to over 100 frames/second using the algorithms from this thesis. This poor performance is largely due to the impossibility of using prefiltered mipmaps [20] or summed-area tables [4] to accelerate percentage closer filtering. Consequently, in the worst case, we have to compare every individual texel in the shadow map to the fragment depth to compute the light attenuation for a single framebuffer pixel, which is too slow for interactive applications.

PCF can also be used to soften the edges of shadows, simply by increasing the size of the filter [16]. By clamping the minimum filter size, we can get arbitrarily soft edges while still avoiding minification aliasing. Unfortunately, this also puts a minimum bound on the cost of shading a pixel that increases with the desired edge softness.

Chapter 3

Variance Shadow Maps

To address the problem of efficiently filtering shadow maps, note that each texel of a standard shadow map can only represent the depth of a single point. Variance shadow maps (VSMs) improve on this scheme by representing a distribution of depths at each texel. To approximate such a distribution using a small amount of data, we store its first and second moments: the mean depth and the mean squared depth. One major advantage of this approach is that we can compute the average of two distributions by averaging the two moments, so this representation of the depth distribution is linear.

As noted earlier, one approach to filtering shadows is to compute the percentage of texture samples that are in shadow over the footprint of a framebuffer pixel. Percentage closer filtering does this exactly, but is expensive. Instead, when querying the variance shadow map, VSMs use the moments to compute a bound on the fraction of the distribution that is more distant than the surface being shaded. I show that this bound provides a good approximation for the amount of light reaching any given surface, and therefore can be used for rendering correctly antialiased shadows.

Because these moments can be interpolated and averaged, we can make use of the wide range of filtering techniques and hardware that are available to address aliasing in color textures.

In summary, variance shadow mapping

- reduces shadow map aliasing by enabling the use of filtering techniques such as mipmapping and anisotropic filtering,
- allows shadow maps to be prefiltered to compute an approximation of percentage closer filtering, and
- can be implemented on current graphics hardware at a cost comparable to that of ordinary shadow maps.

3.1 Algorithm Overview

As with conventional shadow mapping, we first render the scene from the light's point of view. For shadow mapping, we would render the depth as seen from the light; for variance shadow maps we render into a two-channel buffer, rendering both the depth and the square of the depth. Although in regular shadow mapping we would not want to use any type of antialiasing when rendering from the light's point of view, multisample antialiasing (MSAA) is actually a benefit when rendering variance shadow maps.

Once we have created the variance shadow map, we can preproces the texture to facilitate filtering. This preprocessing can include generating mipmaps [20] or computing summed area tables [4]. To further reduce aliasing and soften shadow edges we can also blur the variance shadow map.

Since we have rendered depth and squared depth in the texture, filtering the shadow texture recovers the expectation of these values, called the moments M_1 and M_2 over that filter region. These moments are defined as follows, where $f(x)$ is the probability density function of the depth distribution over the filter region:

$$\begin{aligned} M_1 &= E[x] = \int_{-\infty}^{\infty} x f(x) dx \\ M_2 &= E[x^2] = \int_{-\infty}^{\infty} x^2 f(x) dx \end{aligned} \tag{3.1}$$

From these we can compute the mean μ and variance σ^2 of the distribution:

$$\begin{aligned} \mu &= E[x] = M_1 \\ \sigma^2 &= E[x^2] - E[x]^2 = M_2 - M_1^2 \end{aligned} \tag{3.2}$$

The variance can be interpreted as a quantitative measure of the width of a distribution. As a result, it places a bound on how much of the distribution can extend away from the mean. This bound is stated precisely by Chebyshev's inequality:

Theorem 1 (Chebychev's inequality, one-tailed version)**.** *Let X be a random variable drawn from a distribution with mean μ and variance σ^2. Then for $t > \mu$*

$$P(X \geq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t - \mu)^2} \tag{3.3}$$

$P(X \geq t)$ in Equation 3.3 is exactly the quantity that percentage closer filtering computes, since it represents the fraction of pixels over a filter region that will fail the depth comparison with a fixed depth t.

Equation 3.3 is only an upper bound; there is no guarantee that it will allow us to compute the true value $P(X \geq t)$. Nevertheless it can provide a good approximation, as I show in the following example. Later in this thesis I will show how to improve on this bound by warping the depths and using multiple distributions, although in this chapter I will focus on a single distribution.

3.2 Planar Occluders and Receivers

Consider the case of a single planar occluder at depth d_1, casting a shadow onto a planar surface at depth d_2. Suppose we have a fixed filter region in the variance shadow map, where p is the percentage of the filter that is unoccluded. Then we have:

$$\begin{aligned} \mu = E(x) &= pd_2 + (1-p)d_1 \\ E(x^2) &= pd_2^2 + (1-p)d_1^2 \\ \sigma^2 &= pd_2^2 + (1-p)d_1^2 - (pd_2 + (1-p)d_1)^2 \\ &= (p - p^2)(d_2 - d_1)^2 \end{aligned}$$

Using these values, we can compute $p_{\max}$ according to Equation 3.3:

$$\begin{aligned} p_{\max}(d_2) &= \frac{\sigma^2}{\sigma^2 + (\mu - d_2)^2} \\ &= \frac{(p-p^2)(d_2-d_1)^2}{(p-p^2)(d_2-d_1)^2 + (pd_2 + (1-p)d_1 - d_2)^2} \\ &= \frac{(p-p^2)(d_2-d_1)^2}{(p-p^2)(d_2-d_1)^2 + (1-p)^2(d_2-d_1)^2} \\ &= \frac{p-p^2}{1-p} \\ &= p \end{aligned}$$

Thus in this simple situation, we see that Chebyshev's inequality is an equality, and gives the exact result of percentage closer filtering.

Although this is a special situation, it provides a reasonable approximation to a common situation in many real scenes. In the case of a single occluder and single receiver, we can take a small neighbourhood in which the depth of the occluder and receiver are approximately constant. If the depths vary over the filter region due to the slope of the occluder and receiver, Equation 3.3 will not provide an exact value, but will still give a close approximation. Therefore it is practical to use $p_{\max}$ in rendering as an approximation to the true value p, and high visual quality can be obtained.

3.3 Results

Now that we have chosen a linearly filterable representation of the depth information needed for shadow mapping, many techniques that deal with linear data are available to us. In particular, Figure 3.1 shows the result of simply using texture filtering mechanisms such as mipmapping, trilinear and anisotropic filtering on the

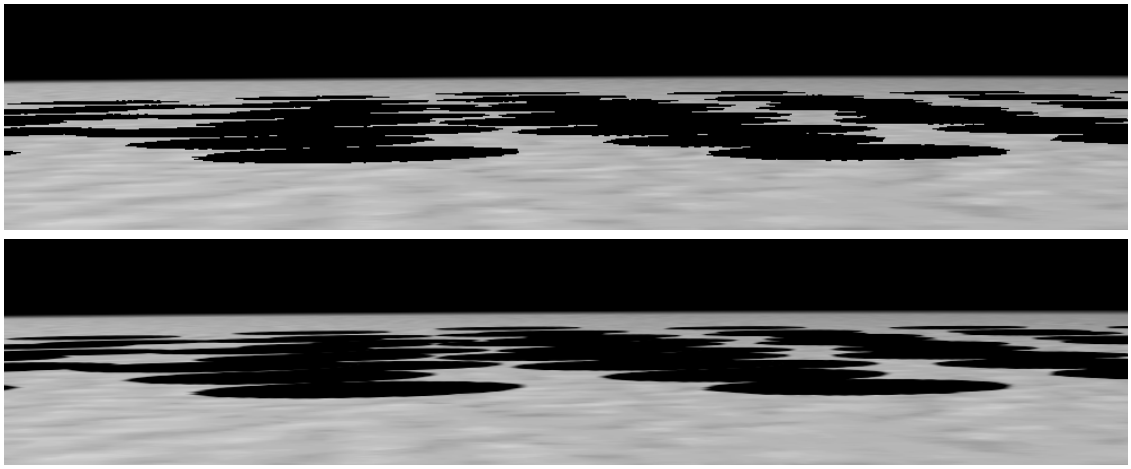


Figure 3.1: Comparison of standard shadow maps (top) with variance shadow maps using mipmapping and anisotropic filtering (bottom).

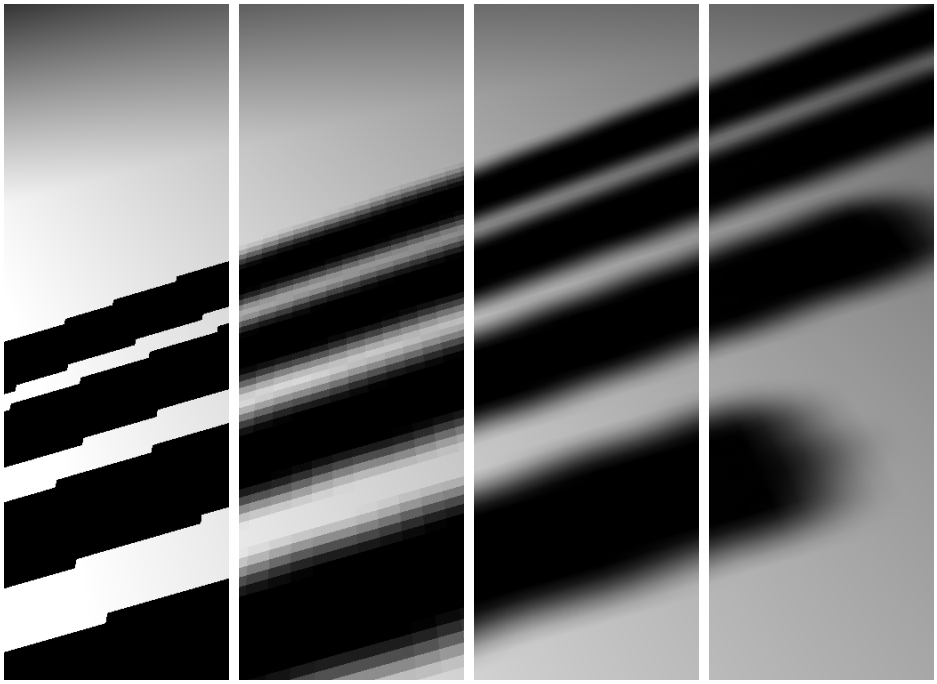


Figure 3.2: Left to right: standard shadow mapping, 5×5 percentage closer filtering, 5x5 bilinear percentage closer filtering, and variance shadow maps with 5×5 separable Gaussian blur.

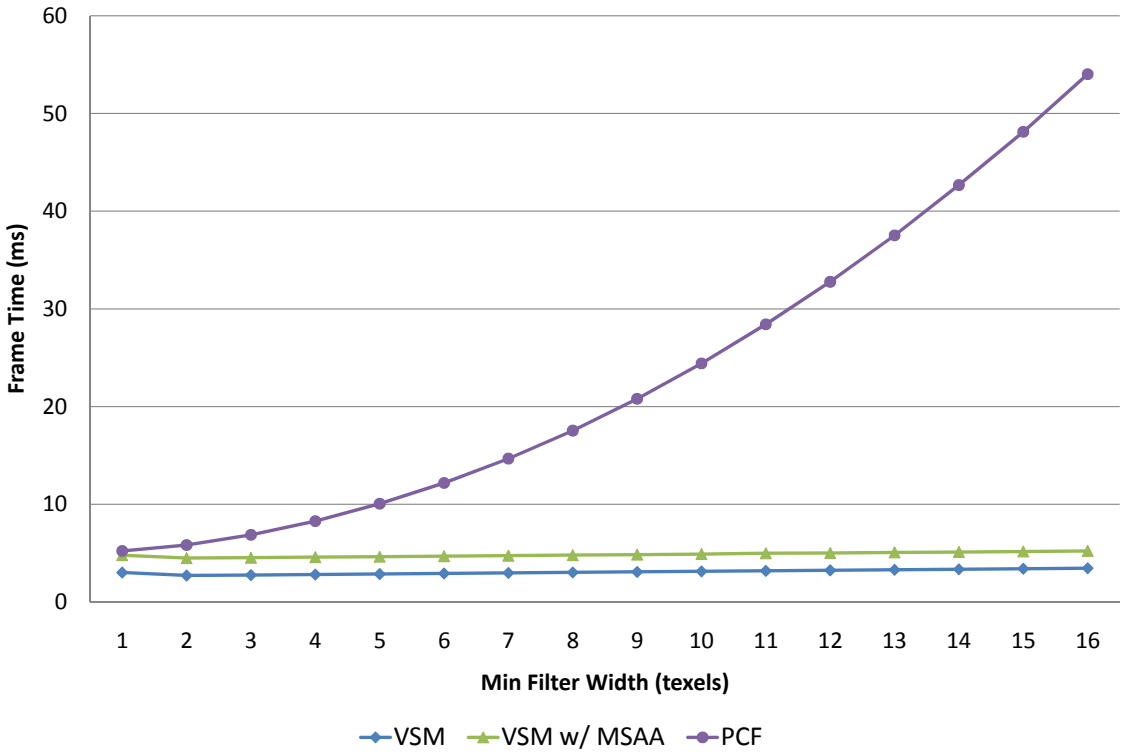


Figure 3.3: Relative frame rendering times of PCF and VSM for a simple scene (lower is better). Performance results were measured on a GeForce 8800 GTX in Direct3D 10 at 1600×1200 with 4×MSAA, using a 512×512 shadow map. VSM with shadow map MSAA produces the best shadows, while the other techniques have comparable quality.

variance shadow map. As the figure demonstrates, the quality of the shadows is greatly improved and aliasing is effectively eliminated.

We can also apply a separable blur to the variance shadow map to achieve inexpensive edge softening. Blurring the VSM is equivalent to clamping the minimum filter region size. Figure 3.2 shows a side-by-side comparison of the results of using a prefiltered variance shadow map and an equivalent percentage closer filter. As the figure demonstrates, the outputs of these two methods are almost identical, and vastly superior to simple shadow mapping and standard nearest neighbor percentage closer filtering.

Variance shadow maps are also fast on modern hardware. Figure 3.3 compares the relative performance of PCF (with hardware acceleration [7]) to VSMs (with and without shadow multisampling). Because both of these algorithms operate in image space, I used a simple scene (approximately ten thousand polygons) to avoid equalizing bottlenecks due to scene complexity.

As the results demonstrate, variance shadow maps are faster than percentage closer filtering, even with shadow multisampling (which is inapplicable to PCF).

Additionally, PCF scales quadratically with the filter size, whereas VSMs scale linearly since a separable two-pass prefilter can be used.

3.4 Light Bleeding

Variance shadow maps are simple and efficient, but unfortunately they suffer from a visual artifact: light bleeding. While Chebyshev's inequality gives an upper bound on $P(X \geq t)$, there is no guarantee that the upper bound is a good approximation. As an example, consider the situation shown in Figure 3.4.

Let objects A, B and C be at depths a, b and c respectively. Note that only objects A and B will be represented in the filter region since object C is not visible from the light. Thus if we are shading a fragment at the center of the outlined region, we will recover the following moments from Equation 3.1:

$$\begin{aligned} M_1 &= \frac{a+b}{2} \\ M_2 &= \frac{a^2+b^2}{2} \end{aligned}$$

Then from Equation 3.2, we compute:

$$\begin{aligned} \mu &= \frac{b+a}{2} \\ \sigma^2 &= \frac{(b-a)^2}{4} \end{aligned}$$

Now consider shading the fragment on the surface of object C.

In terms of $\Delta x = b - a$ and $\Delta y = c - b$ (shown in Figure 3.4), Equation 3.3 yields the visibility function:

$$\begin{aligned} p(\Delta y) &= \frac{\frac{1}{4}\Delta x^2}{\frac{1}{4}\Delta x^2 + (\Delta y + \frac{1}{2}\Delta x)^2} \\ &= \frac{\frac{1}{4}\Delta x^2}{\frac{1}{2}\Delta x^2 + \Delta x \Delta y + \Delta y^2} \end{aligned} \tag{3.4}$$

Therefore for a given Δx, $p(\Delta y)$ falls off like $O(\frac{1}{\Delta y^2})$.

The correct visibility function equals zero for all $\Delta y > 0$, since everything further from the light than object B is fully occluded by it. This discrepancy causes light bleeding: the $O(\frac{1}{\Delta y^2})$ "tail" of Chebyshev's inequality means that unwanted light can show up in regions that should be in shadow.

This is not a problem with the inequality itself. We simply have not kept enough information about the depth distribution to disambiguate more complex combinations of occluders and receivers.

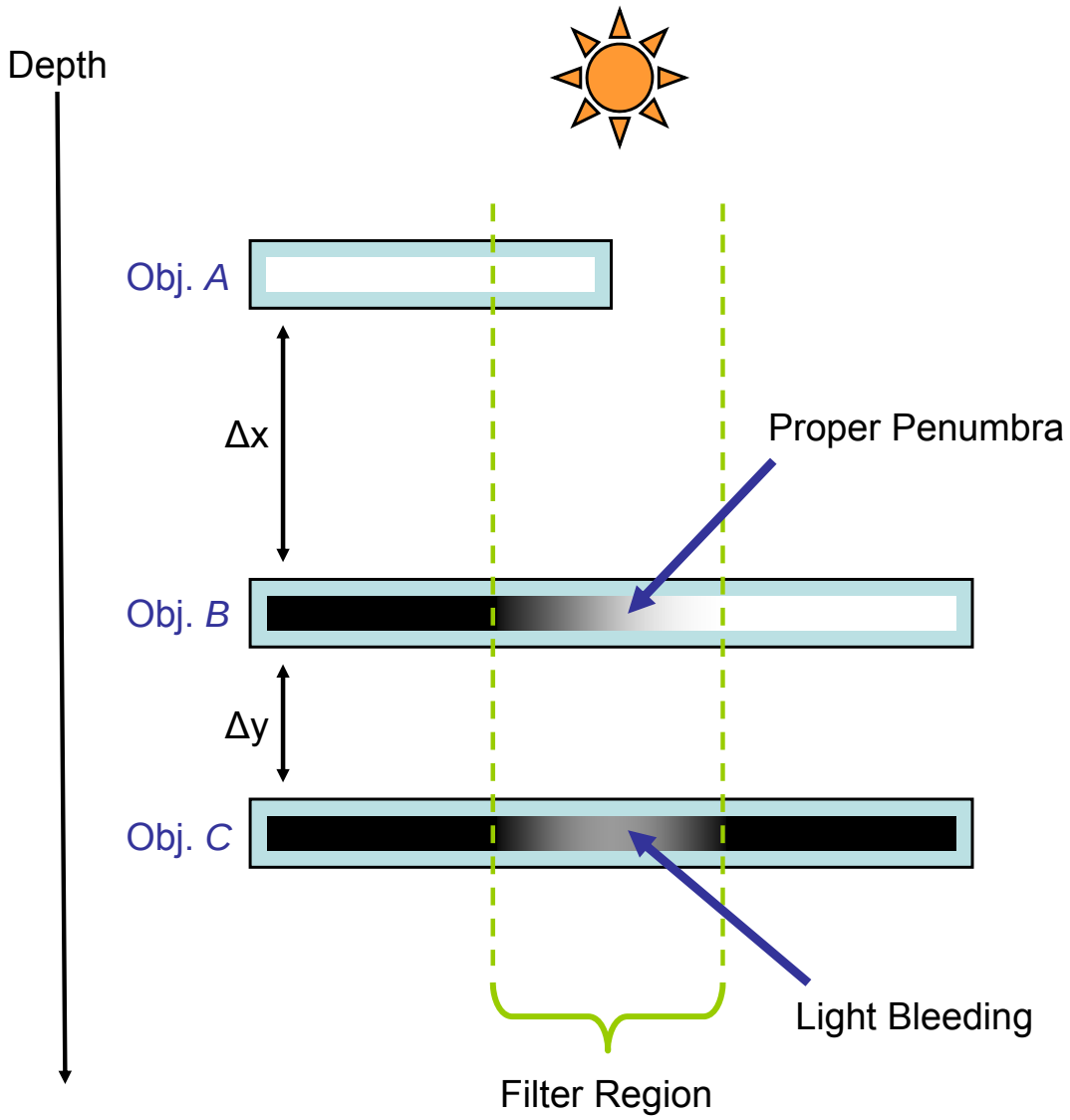


Figure 3.4: A simple situation in which light bleeding occurs. Object A casts a proper shadow onto object B. Object C should be fully occluded by object B, but some of the penumbra from object A's shadow bleeds through incorrectly.

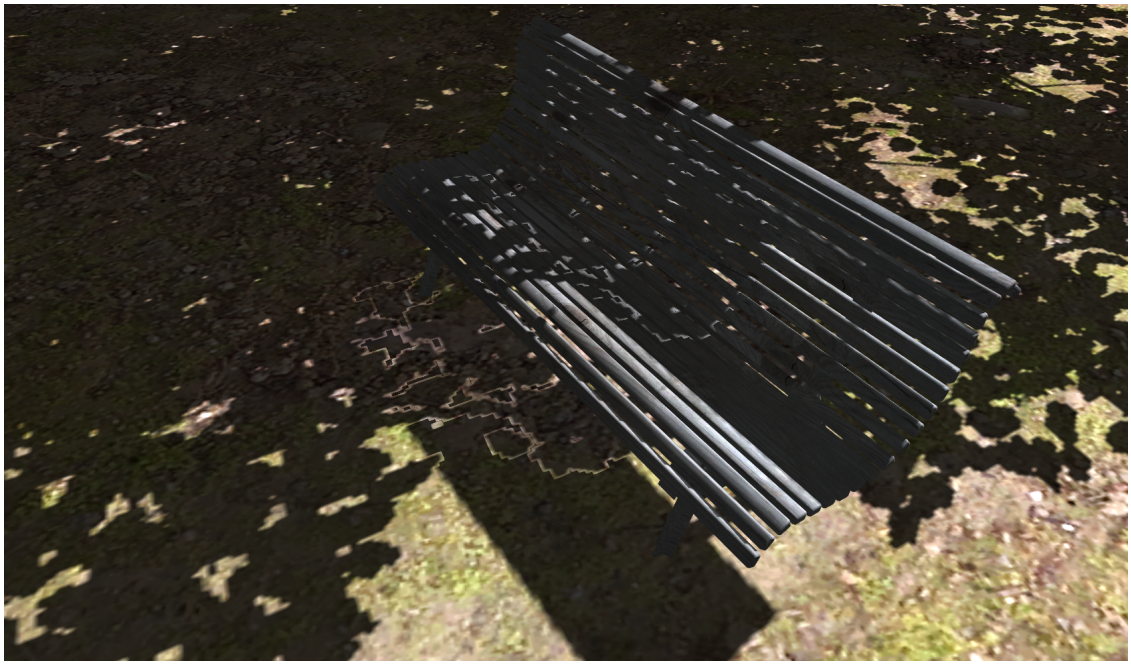


Figure 3.5: Light bleeding artifacts: the penumbra of the tree shadow is improperly bleeding through the bench.

Figure 3.5 shows an example of the visual effect of light bleeding in practice. In this example, object A is a tree, object B is the bench and object C is the ground. Note that the ratio of Δx (tree to bench) to Δy (bench to ground) is high, leading to significant artifacts.

An easy way to reduce these artifacts is to clamp out light that falls below some minimum intensity [10]. While simple and often effective in practice, this over-darkening of the shadow penumbrae can reduce shadow detail, and even reintroduce aliasing if the intensity function is clamped too aggresively. For scenes with significant light bleeding, there is no way to fully eliminate the artifacts without the shadows degenerating into hard-edged, formless blobs.

To eliminate light bleeding without over-darkening, we need to store more information. Using more moments is undesirable as higher-order moments are numerically unstable and deriving suitable inequalities is extremely difficult. Instead, note that whether or not an object is in shadow is not dependent on the exact depth of an occluder, but only if it is closer to the light than the occluder. In other words, shadows are a function of the order of objects, not of their exact depths. We can therefore apply any monotonic warp we want to the depths before computing the moments, and we can even combine multiple warps. Both of these techniques can be used to acheive tighter bounds without giving up the advantages of linearity. I explore these ideas in more detail in the next chapter.

Chapter 4

Warped Variance Shadow Maps

Recall that we are trying to approximate $P(X \geq t)$. This probability does not depend on the actual depth values stored in the VSM, but only on their ordering. On the other hand, Chebyshev's inequality does depend on the actual depth values, but is guaranteed to provide an upper bound regardless. Thus by using different monotonic warps of the depth values we can obtain different upper bounds for $P(X \geq t)$, some tighter than others. We can then choose a warp that gives a good bound, or combine multiple warps to get a tighter overall bound.

Consider for example the following "step function" warp φ_c, again using the example from Figure 3.4 (recall that c is the depth of object C):

$$\varphi_c(t) = \begin{cases} 0 & \text{if } t < \text{c} \\ 1 & \text{if } t \geq \text{c} \end{cases} \tag{4.1}$$

Proceeding as before:

$$\begin{aligned} M_1 &= \frac{\varphi_c(a) + \varphi_c(b)}{2} = 0 \\ M_2 &= \frac{\varphi_c(a)^2 + \varphi_c(b)^2}{2} = 0 \\ \mu &= 0 \\ \sigma^2 &= 0 \end{aligned}$$

Now when we evaluate $p(t)$ for objects B and C:

$$\begin{aligned} p(\varphi_c(b)) &= p(0) = 1 \\ p(\varphi_c(c)) &= p(1) = \frac{\sigma^2}{\sigma^2 + (1-\mu)^2} = 0 \end{aligned}$$

This time the visibility for object C is correct. Although the value for object B is no longer correct (it should be $\frac{1}{2}$), it still provides an upper bound, albeit a trivial one. The function $\varphi_c(t)$ is an example of a "perfect" warp for evaluating the

shadowing of an object at depth c. Such a function warps Δx to 0 so that all light bleeding is eliminated as per Equation 3.4.

Of course this is a crude warp, and as discussed it actually makes the bound worse elsewhere (for example at object B). In the next section, I describe how to choose warps that result in an approximation that is at least good as standard VSMs, but can also be scaled up to reduce light bleeding as much as desired.

4.1 Layered Variance Shadow Maps

While any monotonic warp function is potentially useful, roughly linear ones have several desirable characteristics. First, assuming that the original depth metric has a good distribution of precision along the depth range, linear warps will maintain that distribution.

Second, since we do not know the points and filter regions at which we will need to evaluate the visibility function when rendering the shadow map, we need to be able to reconstruct the function over the entire depth range with good accuracy. To this end, I split the scene into multiple depth layers (in light space) and define a warp function for each layer that can be used to approximate the visibility for fragments that fall into the layer's depth range.

Thus for each layer L_i covering an interval of depths $[p_i, q_i]$, I define the following linear warp:

$$\varphi_i(t) = \begin{cases} 0 & \text{if } t \leq p_i \\ \frac{t-p_i}{q_i-p_i} & \text{if } p_i < t < q_i \\ 1 & \text{if } q_i \leq t \end{cases} \tag{4.2}$$

This new warp has light bleeding reduction properties similar to the $\varphi_c(t)$ function defined in the previous section. Specifically, if all occluders have depths at most p_i and the receiver has depth greater than p_i, the visibility function will be computed exactly. Additionally, occlusion within a single layer is resolved in the same manner as with variance shadow mapping.

For a geometric interpretation of what is happening, consider the silhouette cast by some set of occluders onto some receiver. Imagine flattening this silhouette onto a plane sitting above the receiver, turning the occluders into a simple occlusion "cutout". The shadow cast on the receiver will be identical, assuming a point source, but will be easier to compute since we have transformed the problem into one of a single occluder and receiver, which VSMs handle perfectly.

When shading a surface, we could theoretically sample all of the layers, compute Chebyshev's inequality and take the minimum value over all layers. However with the layered warp functions we only need to sample a single layer: the layer L_j that contains the receiver surface that we are shading (at depth t).

For all layers closer to the light ($i < j$), the visibility function will be at least as high (lit) since as we move closer to the light, occluders will be removed which can

only increase the light reaching the surface in question. Conversely for all layers further from the light ($i > j$), the layer warp will clamp the surface depth to zero. Because $\mu \geq 0$, the probability $p(\varphi_i(t)) = 1$, which is naturally not a useful upper bound. Therefore, for a given depth we only need to sample a single layer.

Another advantage of the layered approach is that each of the layers requires less numerical precision, since they each cover a smaller depth range. This allows layered variance shadow maps (LVSMs) to run on a wider range of graphics hardware than VSMs since high precision texture filtering is not required.

In summary, instead of building the entire visibility function using only two moments, I use a piecewise function that is exact at the interval boundaries, with each interval being reconstructed independently. When only one layer is used, the technique reduces to standard VSMs.

4.1.1 Layer Overlaps

My analysis thus far has focused on the simple case shown in Figure 3.4. Nevertheless, with a few additional details the arguments generalize to different occluder and receiver distributions.

One problem that must be handled is layer splits that cut through receiver distributions. If this occurs in shadowed regions the minimum variance clamp can cause a small, lit gradient while approaching the layer edge. Conversely, lit regions can have small shadow gradients due to bilinear filtering. Both of these artifacts can be seen in Figure 4.1.

Fortunately these problems can be avoided by allowing a small overlap between adjacent layers. For N layers with split locations $s_i, i = 0 \ldots N$, I widen the layer intervals by $\pm\delta$:

$$\begin{aligned} L_1 &= [s_0, s_1 + \delta] \\ L_2 &= [s_1 - \delta, s_2 + \delta] \\ L_3 &= [s_2 - \delta, s_3 + \delta] \\ &... \\ L_N &= [s_{N-1} - \delta, s_N] \end{aligned}$$

The only remaining question is how to choose the split locations s_i, which is the topic of the next section.

4.1.2 Placing Layers

The strategy used to partition the depth range depends on application goals. If the goal is to achieve the best numeric precision over the entire depth range, a simple uniform split scheme is ideal. If the goal is to eliminate light bleeding on a specific receiver surface, one or more split points can be placed near the surface.

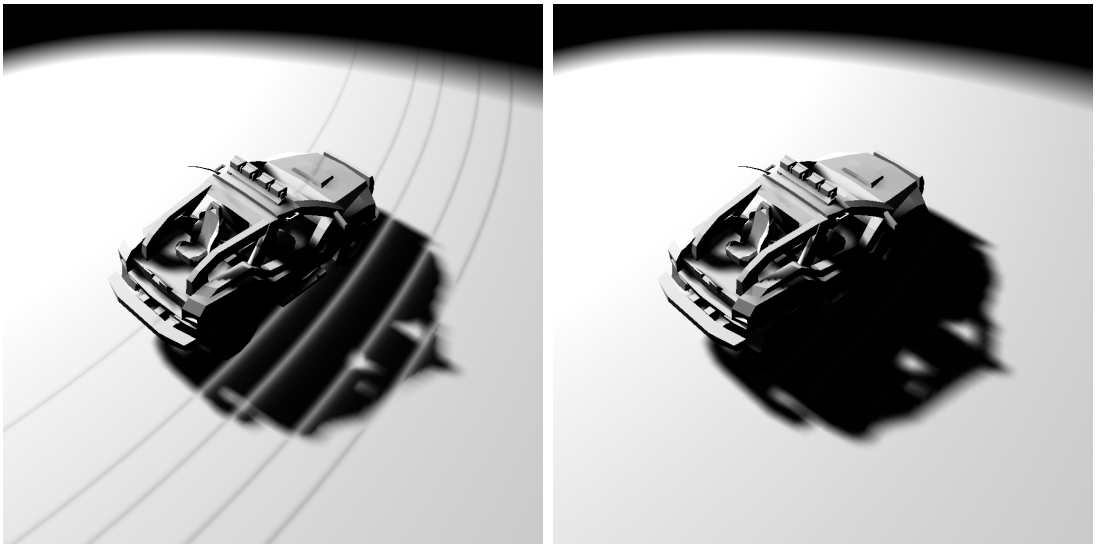


Figure 4.1: Artifacts visible at layer edges (left). These can be completely eliminated by employing small overlaps between layers (right). Note that the filter sizes have been exaggerated to demonstrate the artifact more clearly.

For arbitrary scenes, a more automated approach is desirable. I describe one such algorithm in this section. Ideally we want to place layers between the second and third objects in a light bleeding situation. Generally we do not know where light bleeding is going to occur, so in practice we want to find depth discontinuities that may be susceptible to this artifact and place a layer boundary to avoid it.

Note that even if we cannot place a layer "perfectly" between the second and third objects, if we place a layer between the first and second we will still reduce any light bleeding. Placing such a layer clamps the magnitude of Δx, and thus reduces the relative intensity of the light bleeding (see Equation 3.4).

I will now present an iterative algorithm based on Lloyd's relaxation algorithm [6] that attempts to place the split points as intelligently as possible without any specific knowledge of the scene.

Lloyd's Relaxation Algorithm

Light bleeding happens at depth discontinuities in the shadow map, that is, at shadow edges. My approach treats the problem of placing layer splits as a 1-dimensional weighted k-means clustering problem, where the "means" are the second occluder depths and the weights are chosen based on the likelihood of light bleeding. By applying Lloyd's relaxation algorithm using these definitions, we can place k layer boundaries at peaks in the second occluder depth histogram.

Lloyd's algorithm [6] is a popular iterative refinement algorithm used to solve the k-means clustering problem. Given a partitioning of the input values into k

sets, Lloyd's algorithm computes the mean of each set and repartitions the data, assigning values to the nearest mean cluster. By iterating this relaxation step, the algorithm will converge towards a locally optimal placement of the k means. I use a weighted variant of this algorithm that replaces the computation of the mean with a weighted mean.

To find potentially problematic discontinuities in the depth distribution, consider small neighbourhood filter regions in the unwarped variance shadow map. For each of these regions, we can estimate the second occluder depth and compute suitable weights using the first two moments of the depth distribution.

A useful weight is the variance (σ^2) over some neighbourhood in the unwarped VSM, optionally clamping low or high variances. This has the benefit of more strongly weighting discontinuities that could produce severe light bleeding due to high variance. While I have found this weight to work quite well in practice, application-specific knowledge could potentially produce even better choices.

To recover the second occluder depth, consider the case of only two discrete depths a and b in a given filter with coverages α and $1-\alpha$ respectively:

$$\begin{aligned}
\mu &= \alpha a + (1-\alpha)b \\
\sigma^2 &= M_2 - M_1^2 \\
&= \alpha a^2 + (1-\alpha)b^2 - (\alpha a + (1-\alpha)b)^2 \\
&= \alpha(1-\alpha)(a^2 - 2ab + b^2) \\
&= \alpha(1-\alpha)(a-b)^2
\end{aligned}$$

When the filter is centered over the depth discontinuity ($\alpha = \frac{1}{2}$), σ^2 is maximized. In this situation:

$$\begin{aligned}
\mu + \sigma &= \frac{a+b}{2} + \sqrt{\frac{(a-b)^2}{4}} \\
&= \frac{a+b}{2} + \frac{|a-b|}{2} \\
&= \max\{a, b\}
\end{aligned}$$

Thus when $\alpha = \frac{1}{2}$ we can recover the second occluder depth simply by computing $\mu + \sigma$. For other values of α we will not get an exact answer, but $\alpha = \frac{1}{2}$ is the most important case, since it will be weighted the most highly by the variance weight.

The real benefit of this approach is that it will still produce something reasonable even with large filters and complicated distributions that may not be bimodal. The same cannot be said for assuming a bimodal distribution and attempting to solve for a or b directly.

With these "means" and weights, we can perform an iterative Lloyd relaxation entirely on the graphics hardware (the implementation is described in the next section). The result of the algorithm is to move the layer split points toward the peaks in the weighted "second occluder" histogram that we have computed.

4.1.3 Implementation

I have implemented layered variance shadow maps in Direct3D 10 [2]. Although the algorithm is fully compatible with previous APIs and graphics hardware, several features of Direct3D 10 are beneficial for LVSMs. Specifically, I make use of texture arrays to hold the layer data.

Generating the layered variance shadow map is straightforward. Geometry is rendered from the light's perspective as usual, and some depth metric is computed for each fragment. My implementation uses the normalized Euclidean distance to the light point or plane. For each layer, the depth is warped using Equation 4.2, and the resulting warped depth and warped depth squared are output to a layer render target. Since only two components per layer are needed, I pack two layers into each four-component data texture.

I currently use multiple render targets (MRTs) to output all of the layer data in one pass. Alternatively all of the layer data could be computed in a post-process from a single unwarped depth texture (using a fast z-only pass). Using the latter approach, the additional cost of layered variance shadow maps over standard or variance shadow maps is independent of the scene complexity, since all of the warp functions are computed entirely in image space. Moreover the latter approach does not require MRTs, which may not be supported efficiently on some platforms.

I generally use 16-bit per component fixed-point (normalized) textures to store the layer data as they provide sufficient precision when used with even a small number of layers. A different texture format may be desirable depending on the filtering capabilities of the target hardware.

After rendering the LVSM, I optionally blur all of the layers and generate mipmaps. I store the layer data in a texture array, but the data can be packed into a single texture atlas if texture arrays are unavailable.

While rendering the scene, I determine which layer the current fragment falls into and sample the associated shadow map using hardware anisotropic filtering. The relevant layer index can be computed directly for the uniform split scheme, or it can be determined in the general case by searching the monotonic split point sequence. After retrieving the two warped moments, I warp the fragment depth using the same layer warp function. I then evaluate Chebyshev's inequality (Equation 3.3) and the resulting visibility value is used to attenuate the light reaching the surface.

To avoid any discontinuities at layer boundaries in light bleeding regions, I also use a small transition region, in which the visibility functions of two adjacent layers are blended (recall that we already needed a small overlap between layers). This step can often be skipped with minimal quality loss to avoid the additional sampling and computation.

Lloyd's relaxation algorithm is implemented by rendering a low precision (16-bit fixed point) unwarped VSM in addition to the warped layers. Subsequently

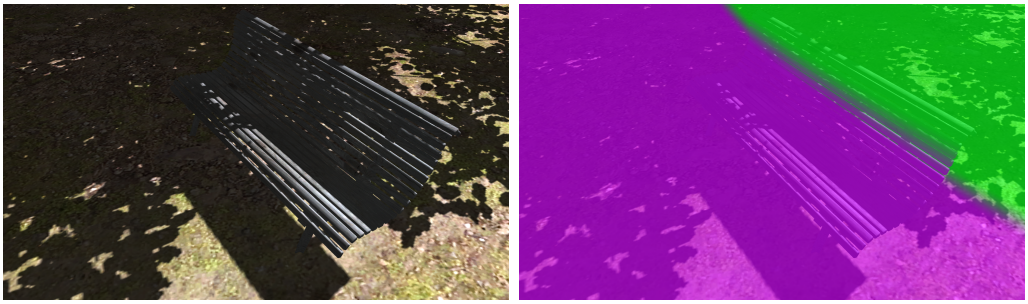


Figure 4.2: Light bleeding artifacts are not present in an image rendered with LVSMs (left). The layers are visualized using different colors in the right image.

I choose two split points to update and generate the K-means clustering data by taking bilinearly filtered samples from the unwarped VSM, positioned in between the texel locations, resulting in a simple 2x2 filter region. I compute the K-means value ($\mu+\sigma$) and determine whether it falls into either (or both) of the split clusters that are being updating (i.e., the given split point is the nearest split point to the sample). If so, I assign a sample weight based on the variance and write out the weighted sample values and the weights for both splits to a four component render target.

At this point it is necessary to compute the weighted mean of all of the data in the new render target. I do this by generating a mipmap pyramid for the texture and reading back the coarsest mipmap level. Dividing the resulting average weighted values by the average weight produces the desired result. In my implementation I defer reading back the results to the CPU (and updating the splits) for a few frames to avoid stalling the graphics pipeline. Alternatively if the split point data is not required on the CPU it can simply be left on the GPU.

If convergence latency is an issue, more splits can be updated per frame (even all of them). In my experience convergence is fast (less than a second), even when only updating two split points per frame.

To avoid the algorithm getting stuck in local optima, I also use a simple split "teleportation" heuristic: if the average weight in a cluster is zero (i.e., the layer has no objects in it), I merge the two adjacent layers and split whatever layer has the highest weight. This simple heuristic greatly improves the quality of the final solutions.

4.1.4 Results

All of the pictures and performance numbers in this section were generated on an Intel Core 2 Duo machine with a single NVIDIA GeForce 8800 GTX using Direct3D 10. Unless otherwise stated, all of the layer positions were determined automatically using the modified relaxation algorithm.

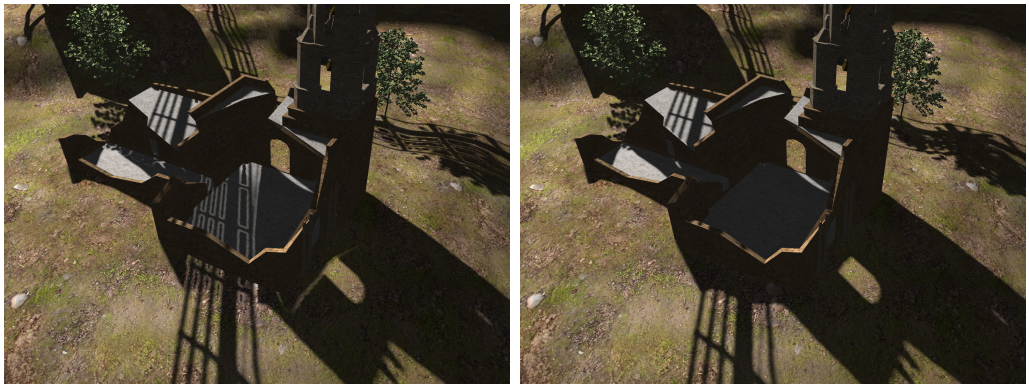


Figure 4.3: Comparison of regular VSMs (left) with LVSMs (right). The scene demonstrates a difficult case for shadow filtering algorithms.

Figure 4.4: High quality shadow filtering using layered variance shadow maps.

Figure 4.2 shows the same scene as in Figure 3.5, except now rendered with a 4-layer LVSM. Note that the light bleeding has been completely eliminated by a single well-positioned layer boundary. Figure 4.3 shows an example from a large scene with many overlapping shadows (Figure 4.4). This scene was designed as a worst-case scenario for VSM-based techniques due to the extreme depth ratios of the overlapping shadows. Artifacts are less pronounced in more typical scenes.

Figure 4.5 compares convolution shadow maps [1] to layered variance shadow maps, with both uniform and Lloyd's relaxation-driven layer positions.

Both CSM and LVSM are scalable and can increase the shadow quality at the cost of additional storage. Each row in the table repesents an equal amount of storage; note that CSM can use twice as many coefficients (as denoted by M in [1]) as LVSM can use layers because CSM can store the coefficients in 8-bit textures while LVSM requires 16-bit textures when few layers are used. It should also be noted that both techniques can fully eliminate the artifacts present in this example by using even more storage.

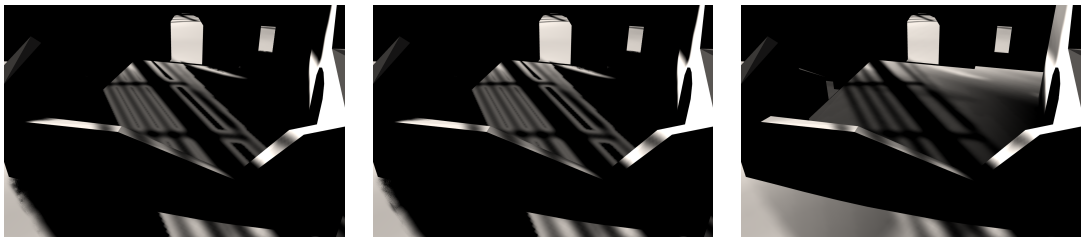
As the figure demonstrates:

- LVSMs are usable with less storage than CSMs particularly when Lloyd's relaxation algorithm is used. The quality of both techniques scales similarly with increased storage.
- LVSMs and CSMs both have light bleeding in similar areas, but LVSMs only have a problem with overlapping occluders while CSMs have a problem with all occluders.
- As the amount of storage increases, LVSMs outperform CSMs since they require only a single texture sample per shaded fragment. CSMs must sample all of the coefficients for every fragment.

Figure 4.6 shows a performance comparison of LVSMs using different numbers of layers with standard VSMs. An entire 1920×1200, 4×MSAA framebuffer is covered and shaded, and every LVSM layer is visible onscreen. Since both algorithms operate in image space, I used a simple scene to eliminate equalizing bottlenecks (Figure 4.1 with textures). Complex scenes impose an equal performance cost on all image-space shadow map algorithms, including layered variance shadow maps.

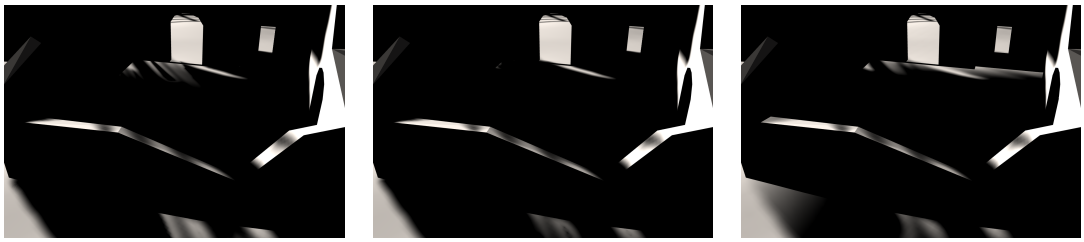
The scene is fully dynamic and all of the shadow data is recomputed every frame. My LVSM implementation is full-featured: it employs Lloyd's relaxation algorithm in real-time, layer transition regions, trilinear and 16× anisotropic filtering, and operates on 16-bit per component fixed-point textures.

Performance data demonstrates that LVSMs are well-suited for use in real-time applications. Even with a 14-layer 1024×1024 LVSM performance remains high, dipping to just under 100 frames per second at 1920×1200 with 4×MSAA. This good performance scaling is largely due to the layer parameterization, which allows



Uniform LVSM-2 (132fps) Lloyd LVSM-2 (109fps) CSM, M=4 (119fps)

Uniform LVSM-4 (112fps) Lloyd LVSM-4 (84fps) CSM, M=8 (79fps)



Uniform LVSM-6 (94fps) Lloyd LVSM-6 (73fps) CSM, M=12 (65fps)

Uniform LVSM-8 (82fps) Lloyd LVSM-8 (63fps) CSM, M=16 (49fps)

Figure 4.5: Comparison between convolution shadow maps and layered variance shadow maps. LVSM-N indicates LVSM with N layers. Each row of the grid uses an equal amount of storage. Performance numbers were taken at 1920×1200, 4×MSAA with a 1024×1024 shadow map.

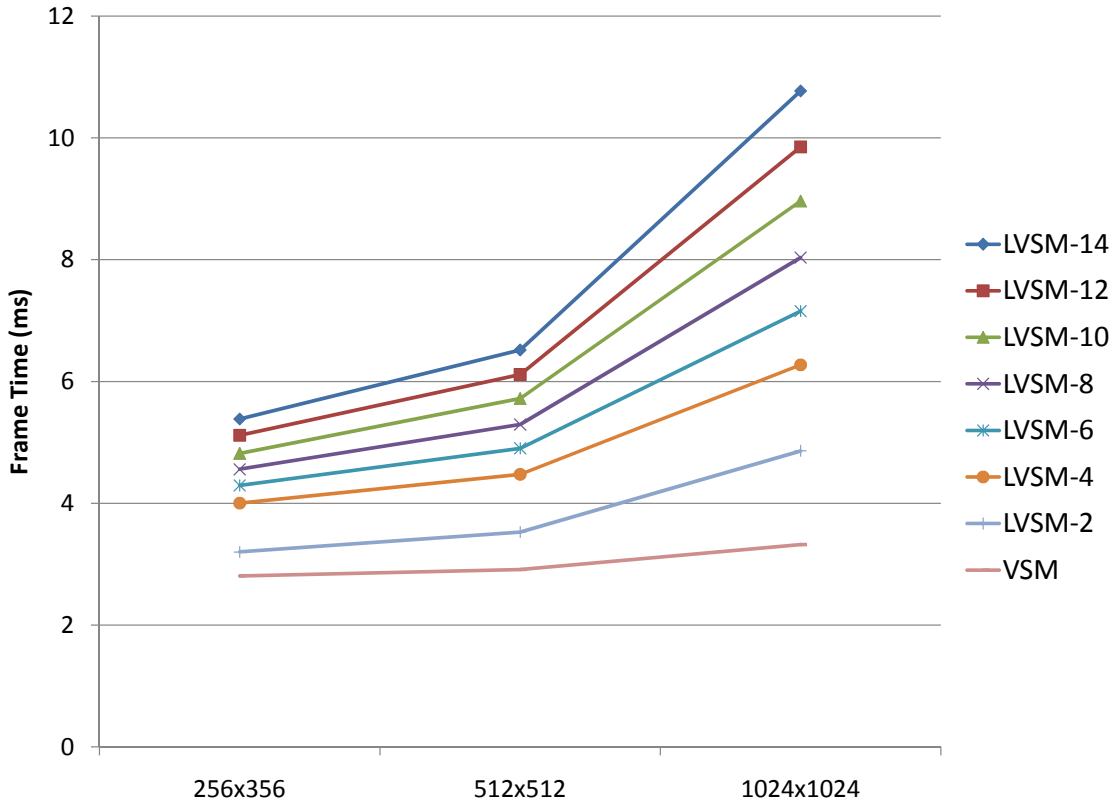


Figure 4.6: Frame rendering time comparison of VSM and LVSM for various shadow map sizes. LVSM-N indicates LVSM with N layers. All tests were performed with with a framebuffer resolution of 1920×1200 with 4×MSAA.

us to sample a single layer per shaded fragment. For more complex scenes the difference in performance between VSMs and LVSMs is even less pronounced.

For instance, a 2-layer, 16-bit per component, uniform LVSM uses the same amount of memory as a standard VSM and provides similar performance. Indeed using a 2-layer LVSM can be conceptualized as a simple way of distributing data precision into multiple components, with the additional benefits of reduced light bleeding and improved texture filtering performance.

4.2 Exponential Variance Shadow Maps

While layered variance shadow maps have several good scaling characteristics, they require a nontrivial amount of storage to fully eliminate light bleeding. Thus it is interesting to consider some other ways to warp the depth function.

As discussed in Section 3.4, light bleeding artifacts increase in severity as the ratio of Δx to Δy grows. LVSMs address this by attempting to clamp Δx to zero. There are also other monotonic warps that decrease this ratio.

For instance, it is interesting to consider the exponential warp function e^{cx} (where c is a constant) as suggested by Salvi [17], but still using a second moment and Chebyshev's inequality. This warp has the effect of relatively moving object B toward object A which reduces the above ratio. The resulting *exponential variance shadow map* (EVSM) has greatly reduced VSM-like light bleeding while still avoiding any bleeding near occluders.

As c is increased, light bleeding is more aggressively eliminated. However, Salvi notes that large c values produce artifacts on nonplanar or multiple receivers [17]. With EVSMs, this problem can be avoided by using the positive e^{cx} warp in conjunction with its negative counterpart, $-e^{-cx}$, in other words, by using multiple warps. This negative exponential warps object B toward object C, making the shadows on B smooth (as they are on C) and thus avoiding the nonplanarity problem with exponential shadow maps (ESMs) [17]. These two warps can be used together: since they both provide upper bounds on the visibility function (via Chebyshev's Inequality), taking the minimum of the two bounds gives a good approximation to the visibility function in most cases. Artifacts will only occur in places where both VSMs and ESMs have artifacts, and increasing c will reduce those instances.

Figure 4.7 shows the result of this exponential warp compared to the algorithms in Figure 4.5. EVSMs not only produce better-looking shadows than both CSMs and LVSMs, but they are faster and require only a single, four-component 32-bit float texture (two moments each for the positive and negative warps). Figure 4.8 demonstrates how EVSMs can render efficient, shadows without artifacts in a scene with many overlapping shadows.

While these results are preliminary, they suggest that there are many interesting warps that can be used in conjunction with variance shadow maps. Exponential warps in particular appear to be a promising direction for future research.

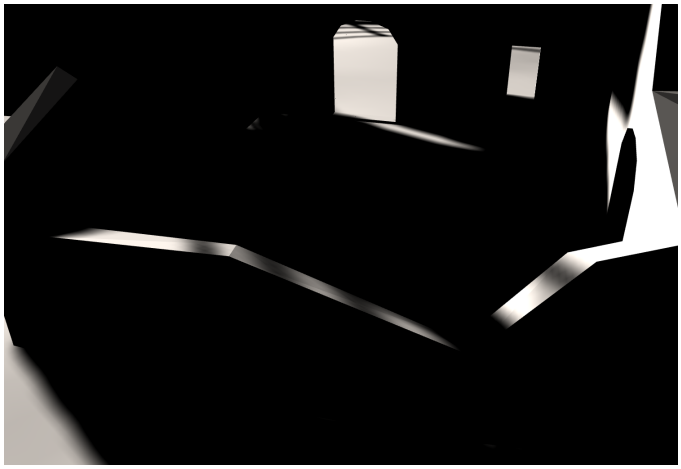


Figure 4.7: Exponentially-warped variance shadow maps show promising results with good performance (140fps).

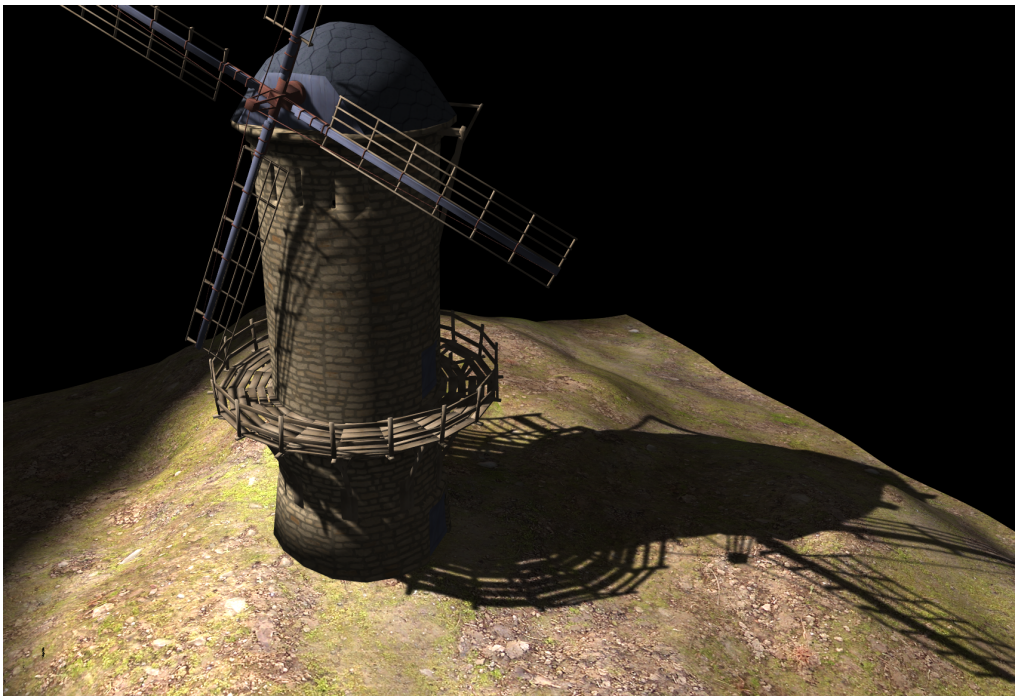


Figure 4.8: A windmill with many overlapping shadows is rendered artifact-free with exponential variance shadow maps.

Chapter 5

Conclusions and Future Work

I have described variance shadow maps as a way to render high-quality, antialiased shadows in real-time applications. By using a linear representation of the distribution of depths in a shadow map, VSM make possible the direct application of a wide variety of efficient texture filtering, prefiltering and antialiasing algorithms.

To remove the visual artifacts of standard variance shadow maps, I have extended the technique to combine the results from multiple warped distributions. I have applied this approach in two ways: layered variance shadow maps and exponential variance shadow maps.

Layered variance shadow maps provide a scalable method to reduce or eliminate the light bleeding artifacts associated with VSMs. In particular, they can be accessed in constant time, independent of the number of layers used. Moreover, I have presented a useful algorithm for automatically placing layer boundaries using a relaxation algorithm that can be implemented at interactive rates. Further improving layer placement is a potentially fruitful research area. One approach would be to perform the relaxation in view space, so that the layer boundary positions are computed based on the size of potential light bleeding regions as seen on-screen, rather than on regions that may be completely hidden.

Secondly, I have demonstrated that exponential variance shadow maps, which combine the results from only two exponentially warped distributions, are able to produce shadows with fewer artifacts and using much less storage than even layered variance shadow maps. Investigating other warps is another promising direction for future research.

In conclusion, I have presented a family of shadow map representation and filtering techniques that produce good quality shadows while maintaining high performance. They are quite suitable for use in games and other real-time graphics applications that require high quality shadow filtering.

List of References

[1] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In Jan Kautz and Sumanta Pattanaik, editors, *Proc. Eurographics Symposium on Rendering 2007*, volume 18, pages 51–60. Eurographics Association, 2007.

[2] David Blythe. The Direct3D 10 system. *ACM Transactions on Graphics*, 25(3):724–734, 2006.

[3] F. Crow. Shadow algorithms for computer graphics. In *Proc. SIGGRAPH 1977*, volume 11, pages 242–248. ACM Press, 1977.

[4] Franklin C. Crow. Summed-area tables for texture mapping. In *Proc. SIGGRAPH 1984*, volume 18, pages 207–212. ACM Press, 1984.

[5] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proc. Symposium on Interactive 3D Graphics and Games 2006*, pages 161–165. ACM Press, 2006.

[6] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev.*, 41(4):637–676, 1999.

[7] Cass Everitt, Ashu Rege, and Cem Cebenoyan. Hardware shadow mapping. Technical report, NVIDIA Corp., 2000. Available at http://www.nvidia.com/.

[8] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *Proc. SIGGRAPH 2001*, pages 387–390. ACM Press, 2001.

[9] Paul S. Heckbert. Survey of texture mapping. In M. Green, editor, *Proc. Graphics Interface 1986*, pages 207–212, 1986.

[10] Andrew Lauritzen. Summed-area variance shadow maps. In *GPU Gems 3*, pages 157–182. Addison-Wesley, 2007.

[11] Aaron E. Lefohn, Shubhabrata Sengupta, and John D. Owens. Resolution matched shadow maps. *ACM Transactions on Graphics*, 26(4):20:1–20:17, October 2007.

[12] Brandon Lloyd, Naga K. Govindaraju, David Tuft, Steve Molnar, and Dinesh Manocha. Practical logarithmic shadow maps. In *SIGGRAPH 2006 Sketches*, page 103, 2006.

[13] Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proc. Eurographics Symposium on Rendering 2006*, pages 215–226. Eurographics Association, 2006.

[14] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proc. SIGGRAPH 2000*, pages 385–392, 2000.

[15] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proc. Eurographics Symposium on Rendering 2004*. Eurographics Association, 2004.

[16] W. Reeves, D. Salesin, and R. Cook. Rendering antialiased shadows with depth maps. In *Proc. SIGGRAPH 1987*, volume 21, pages 283–291, 1987.

[17] Marco Salvi. Rendering filtered shadows with exponential shadow maps. In *ShaderX6*, pages 257–274. Charles River Media, 2008.

[18] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proc. SIGGRAPH 2002*, pages 557–562. ACM Press, 2002.

[19] Lance Williams. Casting curved shadows on curved surfaces. In *Proc. SIGGRAPH 1978*, volume 12, pages 270–274, aug 1978.

[20] Lance Williams. Pyramidal parametrics. In *Proc. SIGGRAPH 1983*, pages 1–11. ACM Press, 1983.

[21] Michael Wimmer, D. Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proc. Eurographics Symposium on Rendering 2004*. Eurographics Association, 2004.

[22] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proc. VRCIA 2006*, pages 311–318, 2006.