LOGARITHMIC PERSPECTIVE SHADOW MAPS

D. Brandon Lloyd

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2007

Approved by:

Dinesh Manocha

Anselmo Lastra

Ming Lin

Steven E. Molnar

Naga K. Govindaraju

ABSTRACT

D. BRANDON LLOYD: Logarithmic Perspective Shadow Maps
(Under the direction of Dinesh Manocha)

The shadow map algorithm is a popular approach for generating shadows for real-time applications. Shadow maps are flexible and easy to implement, but they are prone to aliasing artifacts. To reduce aliasing artifacts we introduce logarithmic perspective shadow maps (LogPSMs). LogPSMs are based on a novel shadow map parameterization that consists of a perspective projection and a logarithmic transformation. They can be used for both point and directional light sources to produce hard shadows.

To establish the benefits of LogPSMs, we perform an in-depth analysis of shadow map aliasing error and the error characteristics of existing algorithms. Using this analysis we compute a parameterization that produces near-optimal perspective aliasing error. This parameterization has high arithmetical complexity which makes it less practical than existing methods. We show, however, that over all light positions, the simpler LogPSM parameterization produces the same maximum error as the near-optimal parameterization. We also show that compared with competing algorithms, LogPSMs produce significantly less aliasing error. Equivalently, for the same error as competing algorithms, LogPSMs require significantly less storage and bandwidth. We demonstrate difference in shadow quality achieved with LogPSMs on several models of varying complexity.

LogPSMs are rendered using logarithmic rasterization. We show how current GPU architectures can be modified incrementally to perform logarithmic rasterization at current GPU fill rates. Specifically, we modify the rasterizer to support rendering to a nonuniform grid with the same watertight rasterization properties as current rasterizers. We also describe a novel depth compression scheme to handle the nonlinear primitives produced by logarithmic rasterization. Our proposed architecture enhancements align with current trends of decreasing cost for on-chip computation relative to off-chip bandwidth and storage. For only a modest

increase in computation, logarithmic rasterization can greatly reduce shadow map bandwidth and storage costs.

ACKNOWLEDGMENTS

I am grateful for the opportunity that I have had to work on this dissertation. I have truly enjoyed my time at the University of North Carolina. I have tried to work hard, always remembering that attending graduate school is a privilege that is not afforded to everyone who would like to attend. I have not arrived at this point on my own and I would like to thank those who have helped along the way.

I would first of all like to thank Parris Egbert, my advisor at Brigham Young University who introduced me to the wonderful world of graphics research as an undergraduate. He encouraged me to publish my work at BYU, which provided valuable preparation for my time at UNC.

I would also like to thank my advisor at UNC, Dinesh Manocha. I think I started working on shadows from the first time I met him. I am grateful for his hands-on work in helping to write my papers, even when that meant staying up until the wee hours of the morning before a deadline. I am grateful for his encouragement and his passion for graphics.

I thank my committee Anselmo Lastra, Min Lin, Steve Molnar, and Naga Govindaraju for their helpful feedback and support.

I owe a lot to my coauthors Jeremy Wendt, David Tuft, Sung-eui Yoon, Cory Quammen, Steve Molnar, and Naga Govindaraju. Jeremy, David, and Cory spent endless hours helping to prepare demos, figures, and videos for papers. Sung-eui Yoon contributed his view-dependent renderer for walkthroughs of massive models. Steve provided invaluable discussion and feedback on questions regarding graphics hardware. I have written several papers with Naga and I am grateful for his contributions, particular with the depth compression algorithm.

Several other people have also contributed to this dissertation. Jon Hasselgren and Akenine-Möller provided their depth compression code. Ben Cloward created the robot model. Aaron Lefohn and Taylor Holliday contributed the town model, which was further embellished

by Sean Curtis. I am especially grateful to Aaron Lefohn and Michael Wimmer for insightful discussions of shadow rendering. I would also like to acknowledge Nico Galoppo and David Feng for proof reading papers and early drafts of this dissertation. Thanks go to the GAMMA group at UNC for great discussions and lively meetings.

This research was made possible by an Graduate Fellowship from the National Science Foundation, as well as a fellowship from NVIDIA. I am grateful for the funding that gave me the freedom to pursue my research.

I would like to thank my family and friends for their personal support. Eric Burns was a wonderful roommate for four years. I enjoyed his humor and long discussions on graphics, politics, religion, and life in general. I thank Tracy Shaw for her upbeat cheerfulness, her friendship, the meals she brought me during paper deadlines, and her example of dedication and tenacity. My parents have been very supportive and have always encouraged me to do and be my best. I thank my wife Amy and her continual support and inspiration to me. She too has borne the burden of long hours in preparing this dissertation and has fallen asleep several times on the floor of Sitterson Hall while keeping me company. Finally, I thank my God for the richness of life with all of its joys and sorrows.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Our perception of the world around us comes largely through what we see. Every day the sun rises and bathes the world in light, and when it sets we continue our activities by use artificial lights. Nearly as important to our sense of sight as the light itself are the parts of the world that remain in darkness – the shadows.

1.1 Importance of shadows

A shadow can be defined as a region of darkness resulting from the blockage of all or part of the light that would otherwise fall on the region. The boundary between light and darkness on a shadow edge helps us to understand the shape of an object or its position relative to another. For example, consider the images shown in Figure 1.1. The first image shows only *attached shadows*, which are shadows that occur at a point when light is blocked by the point

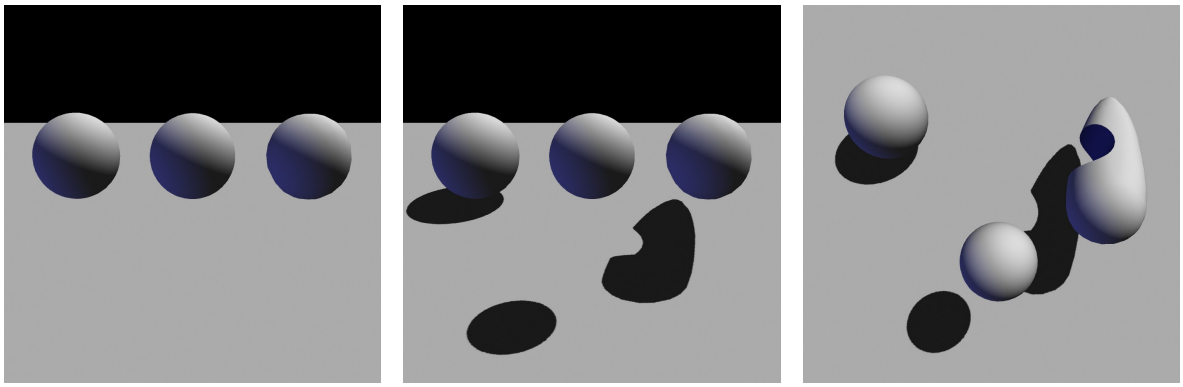

Figure 1.1: ***The benefit of shadows for understanding spatial relationships and shape.*** *(Left) With only attached shadows, this image appears to be three identical spheres floating above the ground. (Middle) Adding cast shadows, we can see that the spheres on the left is touching the ground, the sphere in the middle is floating, and the sphere on the right is not a sphere at all. (Right) The same scene from a different point of view.*

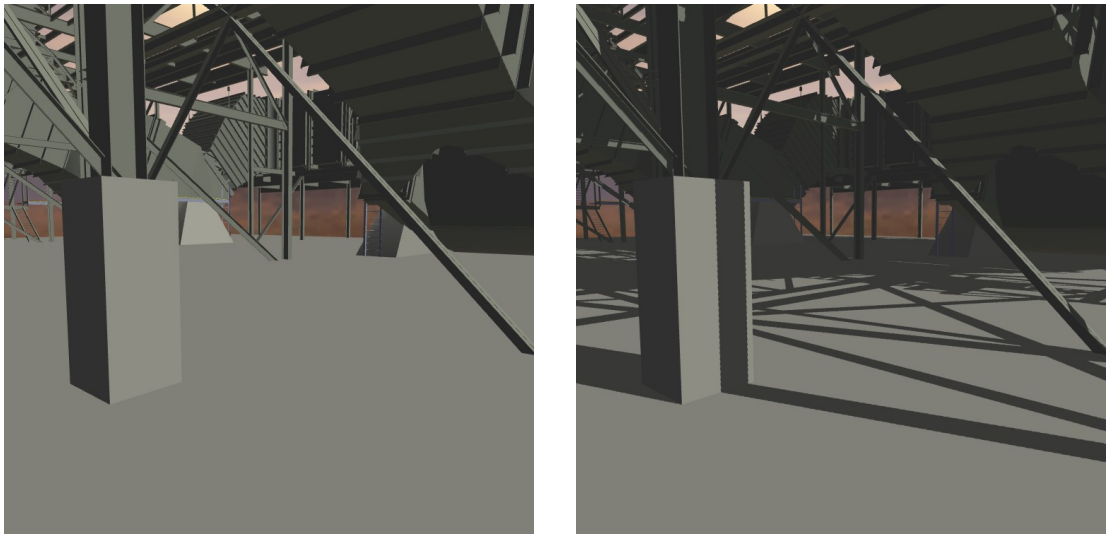

Figure 1.2: ***Shadows in an interactive walk-through.*** *(Left) Without shadows the structures of this power plant model seem to float above the ground plane. (Right) Shadows aid spatial understanding and add realism.*

itself, i.e. when the point lies on a surface facing away from the light. The attached shadows help us to see the curvature of the objects. The image appears to be of three identical spheres lined up in a row. In the next image we add *cast shadows*, which are shadows that occur at a point when light is blocked by some other point. In this case, the objects cast shadows onto the ground plane. The cast shadows reveal that the first two objects are spheres of different sizes and that one of them is actually not a sphere at all! Furthermore, we see that the objects are positioned at different heights above the ground plane. Figure 1.2 shows a less contrived example from a walk-through application in a power plant model. Without cast shadows, the structures appear to float above the ground plane. The spatial understanding gained from shadows is important in still images, but it can be even more useful for interactive applications such as walk-through, simulators, and video games. Several user studies have shown how shadows aid the user in interacting with a virtual environment (Wanger, 1992; Hubona et al., 1999; Hu et al., 2000). They also provide powerful motion cues (Kersten et al., 1996; Kersten et al., 1997).

One of the goals of interactive computer graphics is to produce an experience in a virtual world that visually is as close to reality as possible. In achieving this goal, shadows are of vital importance. Shadows exist nearly everywhere in the real world, and without them the

Figure 1.3: ***Shadows convey time of day.*** *The long shadows indicate a time late in the day.*

illusion of reality is destroyed. Shadows can also strongly influence more subjective aspects such as the "mood" or "feel" of an environment and can be used for dramatic effect. Consider a video game in which the player first sees the shadows of his or her opponent approaching from around the corner, thus heightening the suspense of a pinnacle moment in the gameplay. Shadows can also influence the perceived time of day. In Figure 1.3, the low lighting with long cast shadows gives the impression a late time of day.

Attached shadows are fairly easy to compute because they depend only on the local surface orientation. Computing accurate cast shadows, however, can be computationally challenging. Offline rendering used for creating single images or the still frames of a film can generate very realistic lighting and shadows, but the time spent to render each image can range from minutes to hours. Interactive applications must operate under much tighter timing constraints. Ideally, they must maintain frame rates of greater than about 60 frames per second to create the illusion of smooth motion. In a virtual world where the user can go anywhere and change the view at any moment, the frames cannot be precomputed. This means that each frame must be rendered on the fly. The application has 1/60th of a second to compute the positions of objects, the lighting and shadows, and the final image. In video games other components such as physics and artificial intelligence compete for a slice of the

limited frame time. But even with these constraints, shadows are so important that the popular video game Doom III reportedly spends one half of the frame time on shadows alone (Shoemaker, 2003).

Shadow rendering involves the computation of the visibility of the light source. *Hard shadows* are the easiest type of cast shadows to compute because they are generated from a light source consisting of a single point. When a point light is infinitely far away, it is referred to as a directional light. Unlike real lights, point and directional light sources have no physical extent. They are referred to as *ideal light sources*. To compute the visibility of an ideal light source at a given point in space, we must determine whether the path from the point to the light source is blocked by an occluder. If the path is free, the point is illuminated. Otherwise, it lies in shadow. On a surface, the sudden transition from the light being visible to occluded produces a hard shadow edge. Small or distant lights in the real world can often be approximated by ideal light sources. For example, a point light can be a good approximation for the illumination from a small light fixture in a room, a street light, or even a flash light. A directional light can be a good approximation for what is perhaps the most ubiquitous light source, the sun. However, ideal light sources produce a poor approximation for many other sources of illumination, e.g. fluorescent tubes in an office building or an overcast sky. These light sources give rise to *soft shadows*. The boundaries of soft shadows have a *penumbra*, a smooth transition from lit to unlit. The penumbra arises from partial visibility of a light source having appreciable physical extents. These light sources are referred to as *extended light sources*. Computing shadows from extended light sources is significantly more expensive than computing hard shadows from ideal light sources because we must not only determine whether or not the light source is visible at a point, but also what fraction of the light source is visible. Due to the high cost of soft shadow computation, the shadow algorithms most commonly used in today's interactive applications are based on hard shadows. But even hard shadow computation can pose significant challenges.

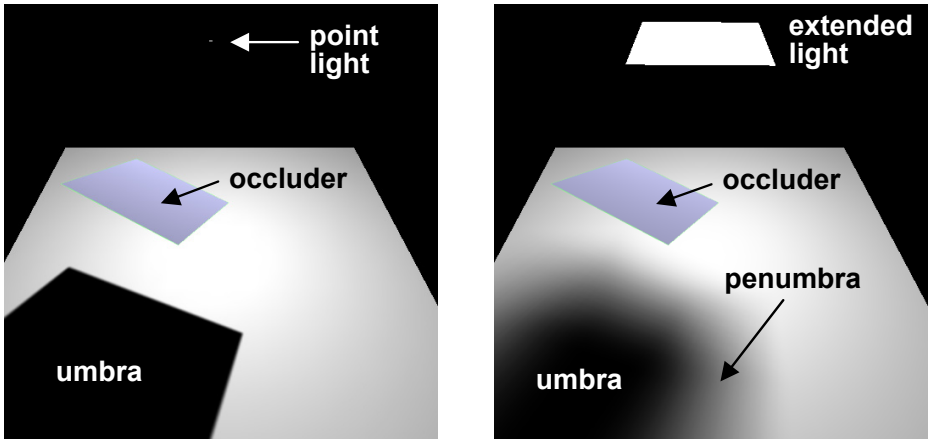

Figure 1.4: ***Hard and soft shadows.*** *(Left) Hard shadows cast by a point light source. (Right) Soft shadows cast by an extended light source. Hard shadows are easier to compute because we need only determine whether the light source is visible or not. Soft shadows have a penumbra region where the light source is partially visible.*

1.2 Shadow maps

One of the most popular techniques for generating hard shadows is the shadow map algorithm (Williams, 1978). A shadow map is a discrete representation of the surfaces in the scene which is used to compute the visibility of the light. The algorithm consists of two rendering passes. First, the shadow map is rendered from the view point of the light. Shadow maps are typically stored as a depth buffer which records at each texel the depth of the surface closest to the light. Second, the image is rendered from the viewpoint of the eye. The point in the scene corresponding to each image pixel is projected into the shadow map. If the depth of the point relative to the light is greater than that stored at the corresponding location in the shadow map, then it is assumed that some other surface lies between the point and the light and the point is classified as in shadow. The basic algorithm is summarized in Figure 1.5.

Shadow maps are an attractive approach for rendering shadows for several reasons. The algorithm is quite general and easy to use. Shadow maps can be implemented using projective texturing (Segal et al., 1992), which is widely available on current graphics hardware. They require no pre-processing and are therefore suitable for dynamic scenes. Shadow maps can be generated using a wide range of geometric representations. For example, in addition to the standard polygon mesh, shadow maps may also be used for alpha test geometry. This

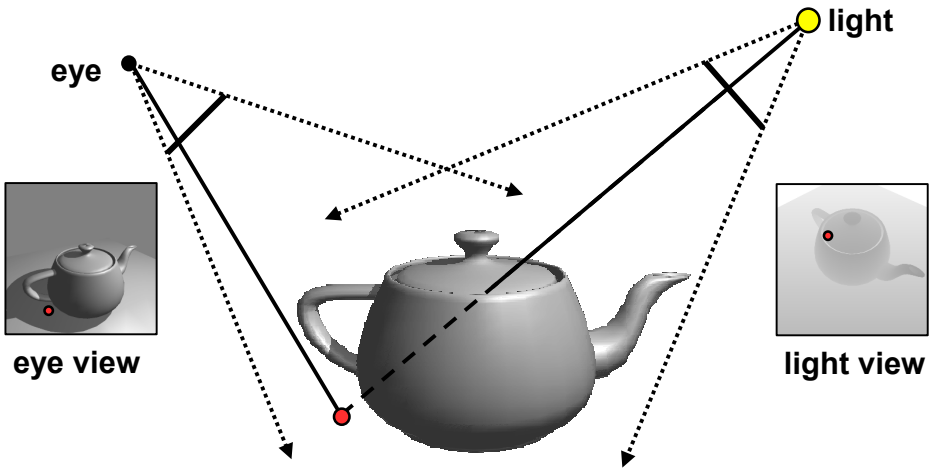

Figure 1.5: ***Shadow map algorithm.*** *A shadow map is created by rendering a depth buffer from the point of view of the light. When rendering the image from the point of view of the eye, the point in the scene corresponding to each pixel is transformed into the light's view and its depth is compared to that stored in the shadow map. If the depth of the point is greater, it lies in shadow. The positions of a sample point in the image, the scene, and the shadow map are shown here.*

representation uses the alpha test in conjunction with the alpha channel of a texture map to "knock-out" regions of the base polygon to which the texture map is applied. This approach is often used in interactive applications to simplify the geometric representations of objects like a plant or a chain link fence, which would otherwise require a large number of polygons. Rendering shadows using shadow maps also tends to have better performance than other shadow algorithms. Because shadow maps contain only depth information, they can take advantage of optimizations available on some GPUs that rasterize to depth-only buffers at higher speeds. Though shadow maps are used primarily for hard shadows, they can also produce plausible soft shadows from small light sources by simply blurring the shadow edges. Applying the blur is fairly easy because of the image-based nature of the algorithm.

The main disadvantage of the shadow map algorithm is that it can suffer from several sampling artifacts. One of these artifacts is a jagged appearance along shadow boundaries that should otherwise be smooth curves (see Figure 1.6). These artifacts occur in parts of the scene that are undersampled by the shadow map. In these regions, a large number of points corresponding to neighboring pixels map to the same texel in the shadow map. This causes the texel boundaries to become distinguishable in the image and results in jagged shadow

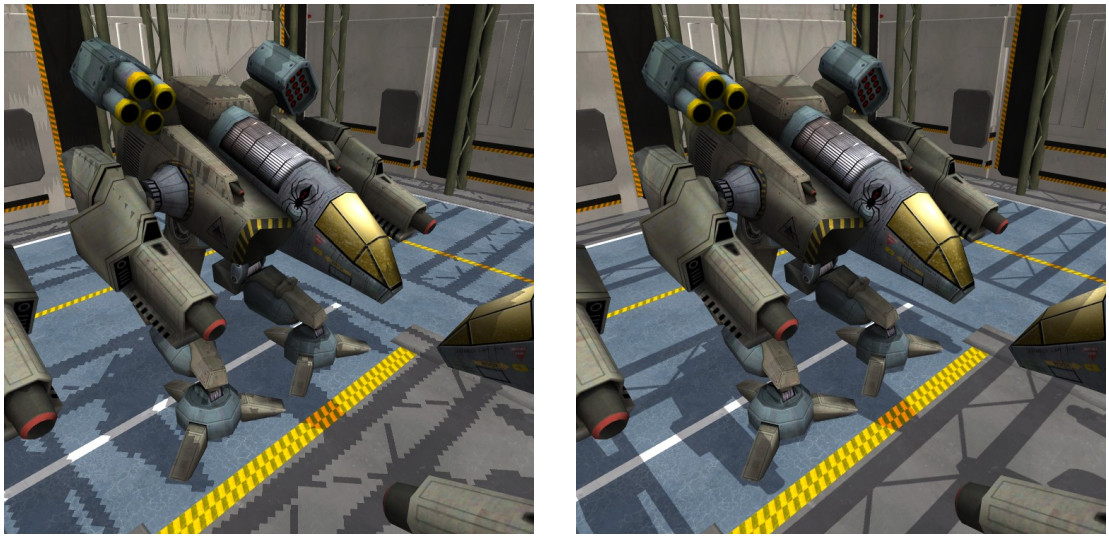

Figure 1.6: ***Shadow map artifacts.*** *(Left) In this image rendered with a standard shadow map the shadow boundaries are jagged. (Right) The same image rendered with our shadow map algorithm without shadow map artifacts .*

edges.

1.3 Avoiding sampling artifacts

There are several ways to avoid the sampling artifacts shown in Figure 1.6. These include using an object-based approach that avoids sampling altogether, increasing the shadow map resolution, or using a better sample distribution. We will briefly discuss each of these approaches.

1.3.1 Object-based approaches

Approaches that compute shadows in object-space do not suffer from sampling artifacts. A popular object-space method is the shadow volume algorithm (Crow, 1977). This algorithm explicit constructs the volume of points that lie behind an occluder and are therefore not visible to the light. The shadow volumes are created by extruding the silhouette edges of the occluder away from the light source toward infinity. To determine if a point lies in shadow, we can trace a ray away from the point in any direction, incrementing a counter whenever the ray enters a shadow volume and decrementing when it exits. If the final counter value is

non-zero then the point lies inside a shadow volume. On graphics hardware, the rays used are those that pass through the eye and the pixels in the image. The stencil buffer is used to provide a counter for each ray.

The shadow volume algorithm consists of three passes. In the first pass, the scene is rendered with ambient lighting. In the second pass, depth buffer writes are disabled and the shadow volumes are rendered to the stencil buffer, incrementing for front faces and decrementing for back faces. In the final pass, the image is rendered again, this time with full lighting and the stencil test is used to disable writes to pixels with a nonzero count, thereby preserving the ambient lighting in shadowed regions.

While the shadow volume algorithm produces accurate shadow edges, it has several drawbacks. Shadow volumes require that occluders be represented with a polygonal mesh, and thus are less flexible than shadow maps. They cannot be used with alpha test geometry or point-based representations, which do not have explicit silhouette edges. As with all object-based approaches, the performance of the shadow volume algorithm is highly sensitive to the geometric complexity of the scene. Individual shadow volumes often cover large areas of the screen. Rendering many large shadow volumes generated by complex scenes can lead to poor performance on current graphics hardware.

1.3.2 Increasing shadow map resolution

The obvious approach to the undersampling problem is to simply increase the resolution of the shadow map. The magnitude of shadow map error is reduced in proportion to the resolution of the shadow map. However, there are several limitations to this approach as well. Large shadow maps strain the limited storage capacities of current graphics hardware. While the amount of memory available on GPUs continues to increase, so does the visual richness of the rendered scenes. High resolution shadow maps must compete for storage in GPU memory with the texture maps, vertex data, and shaders that provide this richness. Increasing the shadow map resolution can also decrease rendering performance. Rendering high resolution shadow maps can take longer because there is more rendering work. The rendering speed is more likely to be limited by bandwidth bottlenecks. Generating the image using a high

resolution shadow map can also be slower due to poor cache performance. Regions farther from the viewer tend to be oversampled by the shadow map. In these regions, shadow map queries exhibit poor spatial locality in memory which can hurt cache performance. Increasing the resolution of the shadow map makes this problem worse.

1.3.3 Explicit sampling

Rather than simply increasing the number of samples to combat aliasing by brute force, a better approach is to place the shadow map samples more carefully. Ideally, the shadow map resolution would be no larger than the image resolution with a single shadow map sample corresponding to each pixel. Approaches such as raytracing (Whitted, 1979) and the irregular z-buffer (Johnson & Cohen, 2004; Aila & Laine, 2004a) establish this correspondence explicitly. Both of these approaches can produce artifact-free shadow edges with far fewer samples than a standard shadow map. Ray tracing can compute shadows at any point in the scene simply by tracing a ray towards the light to check for intersections with occluders. Thus ray tracing can sample exactly at those locations in the scene that correspond to pixels in the image. There is a price to be paid for this flexibility. The entire scene database must be accessible because rays may potentially intersect any object in the scene. In other words, ray tracing must operate in retained mode. Due to the high cost of intersection tests, spatial data structures are typically used to reduce the number of intersection tests that have to be performed in complex scene. Building acceleration data structures for complex models and maintaining them in dynamic scenes can have substantial overhead.

The irregular z-buffer algorithm rasterizes polygons using an arbitrary set of sample locations on the image plane. To generate a shadow map with an irregular z-buffer, the scene is first rendered from the viewpoint of the eye. The eye sample locations are transformed into light space and used to render the shadow map. Finally, the resulting depth buffer is used to compute shadows at the eye sample locations. The irregular z-buffer can produce the same results as raytracing for an ideal light source. Because each polygon can be processed independently, the algorithm fits well with the traditional graphics pipeline. The main challenge for the algorithm is the irregularity of the sample distribution. From the point of view of the

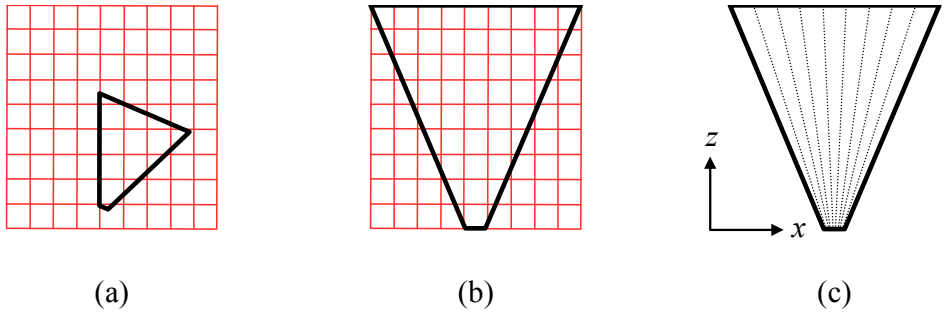

Figure 1.7: ***Fitting the shadow map to the view frustum.*** *These images show the trapezoidal view frustum as seen from an overhead directional light. (a) Shadow map resolution wasted on unseen portions of the scene. (b) The shadow map fit tightly to the view frustum makes better use of the available samples but is still a poor match for the eye sample distribution. Areas near the viewer are extremely undersampled. (c) The eye sample spacing generally increases linearly with z.*

light, the eye sample locations can be highly nonuniform. Indexing these samples requires irregular data structures, such as a quad tree (Aila & Laine, 2004a) or a grid of lists (Johnson et al., 2005), which are more difficult to implement efficiently in hardware than the regular grid used for standard rasterization.

The sample locations for a standard shadow map are derived implicitly using a function parameterized by the texel grid locations. Using a parameterization places constraints on the sample locations, but it has two important advantages – simplicity and a high degree of coherence. Sample locations can be computed and indexed in constant time, and they can be stored in a regular grid. Some important operations, such as filtering, are much simpler using a regular grid. Rasterization using a parameterization is also easy to parallelize. The coherence between samples can be exploited to increase performance via incremental computations and optimizations such as depth buffer compression. The performance of modern GPUs is due largely to these advantages afforded by a parameterization. Recent improvements in ray tracing that better exploit what coherence is available show promising results for static scenes, but these algorithms still cannot compete with the performance of GPUs for moderately complex, dynamic scenes at high resolutions (Wald et al., 2007). Hardware architectures have been proposed for accelerating raytracing (Woop et al., 2005) and the irregular z-buffer (Johnson et al., 2005), but compared to standard rasterization they are more difficult to implement efficiently. As of yet, no mass market implementation exists for either approach.

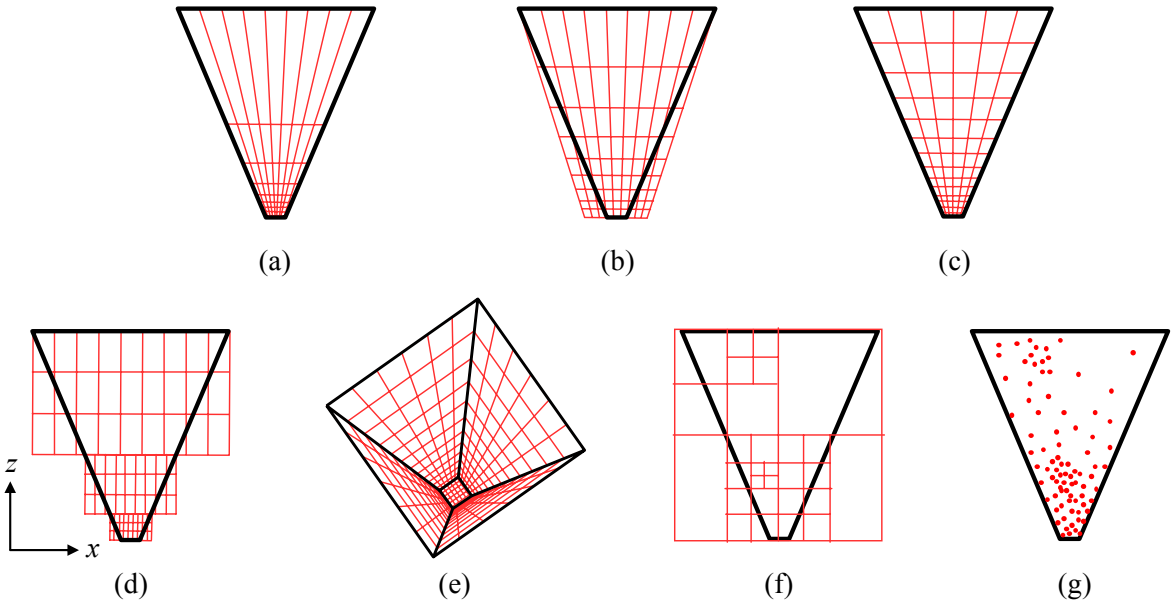

Figure 1.8: ***Techniques for improving shadow map sample distribution.*** *(Top row) Warping techniques: (a) perspective shadow maps (PSMs) (Stamminger & Drettakis, 2002), (b) light-space perspective shadow maps (LiSPSMs) (Wimmer et al., 2004), and (c) logarithmic perspective shadow maps (LogPSMs). (Bottom row) Partitioning methods: (d) z-partitioning, (e) face partitioning, (f) adaptive partitioning, (g) explicit sampling. Explicit sampling can be viewed as using one partition per sample.*

1.3.4 Warping and partitioning

While it is difficult to establish an exact correspondence between eye and light sample *positions* in the scene with a parameterization, it is possible to obtain a better match between the eye and light sample *distributions* than that produced by a standard shadow map. To begin with, the shadow map can be tightly fit to parts of the scene that are actually visible to the eye (Brabec et al., 2002b) so that shadow map resolution is not wasted on parts of the scene that will never be seen. This approach, however, still requires a high resolution shadow map to remove undersampling artifacts for views in which objects are visible throughout the entire view frustum. The problem is illustrated in Figure 1.7. Here the view frustum is seen from a directional light directly overhead. The tapered shape of the view frustum causes the spacing between eye sample locations, in general, to increase linearly with distance from the viewer. However, when using the uniform parameterization of a standard shadow map, the sampling rate is constant over the entire view frustum, leading to undersampling near the viewer.

Several algorithms have been proposed to obtain a nonuniform sampling rate over the view frustum. We classify these algorithms into two main categories: warping and partitioning. Warping controls the sample distribution using a nonuniform parameterization. Partitioning provides local control over the sample distribution by using multiple shadow maps.

Figure 1.8 shows several examples of warping and partitioning. Existing warping algorithms use perspective projections to warp the rectangular sample grid of the shadow map to the trapezoidal view frustum. This warping can also be viewed as expanding the view frustum in shadow map space to fill the rectangular shadow map. Projective warping algorithms include perspective shadow maps (PSMs) (Stamminger & Drettakis, 2002), light-space perspective shadow maps (LiSPSMs) (Wimmer et al., 2004), and trapezoidal shadow maps (TSMs) (Martin & Tan, 2004). For an overhead directional light source, the principal difference between these algorithms is the way in which the parameter is computed that controls the strength of the warping. These algorithms are based solely on the shape of the view frustum and do not consider the surfaces in the scene. As such, they are not guaranteed to completely eliminate shadow artifacts. However, they are fairly cheap to implement. Because perspective projections can be encoded in the 4×4 transformation matrices available on current GPUs, existing warping algorithms need only modify the camera matrix used for rendering the shadow map.

Partitioning algorithms divide the shadow map resolution among multiple, smaller shadow maps to better control the local sampling distribution. The simplest of the partitioning schemes is what we call z-partitioning. z-partitioning splits the view frustum along the eye space z-axis (view direction) into smaller subfrusta and applies a separate shadow map to each (see Figure 1.8d). Multiple shadow maps are better able to conform to the shape of the view frustum than a single one. Adaptive partitioning algorithms use multiple shadow maps, typically arranged in a quadtree, to adapt to local variation in sample density requirements. Because adaptive methods can take the position and orientation of surfaces in the scene into account, they can be more accurate. They also tend to be more expensive because of the added cost of the scene analysis to determine where to refine the partitioning. In addition, they tend to require many more shadow maps than simpler, scene-independent partitioning

techniques. The irregular z-buffer algorithm can also be viewed as a partitioning scheme that uses a partition per sample.

Warping can be combined with partitioning. For instance, each subfrustum generated by z-partitioning can be rendered with a warped shadow map. Adaptive techniques can be applied on top of a global warping of the scene. Another technique, which we call *face partitioning*, subdivides the view frustum into regions corresponding to its faces and applies a warped shadow map to each.

1.4 Problems with existing shadow map techniques

As can be seen, there are quite a few variations on the basic shadow map algorithm that can often be combined. However, the error characteristics of many of the techniques are not well understood. Several of the algorithms are based on a theoretical error analysis, but the analysis is usually limited to a specific light/camera configuration. The behavior of the algorithms for general light positions is poorly understood, which can often lead to unexpected results that make the algorithms more difficult to use. This is particularly true of some warping methods for which the error can vary dramatically with the light position. Some methods are based entirely on heuristics such as maximizing the projected area of the scene in the shadow map. The lack of theoretical analysis makes it difficult for a developer to answer such fundamental questions as "Which algorithm is best for my application?", "How much shadow map resolution will I need?", or "How many z-partitions should I use, and where should the partition be made?"

One problem with the existing warping methods based on perspective projections is that this parameterization does not produce the best match to the eye sample distribution. As can be seen in Figure 1.8a, a perspective projection can produce a good fit to the view frustum. However, along the length of the view frustum the spacing between samples increases quadratically, while the eye sample spacing increases only linearly. Because the spacing distribution in both directions is coupled in a perspective projection, the optimal fit along the view direction can only be obtained at the expense of the fit in the orthogonal direction (see Figure 1.8b).

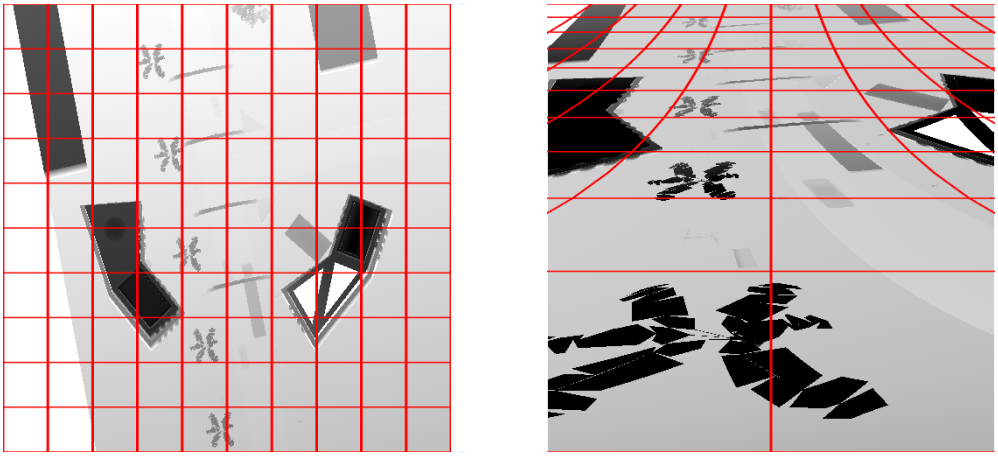

Figure 1.9: ***Standard and logarithmic shadow maps.*** *A uniform grid is superimposed on the shadow map to show the warping from the logarithmic parameterization. Note that under the logarithmic transformation straight lines become curved*

Several researchers (Wimmer et al., 2004; Zhang et al., 2006a) have suggested the use of a logarithmic parameterization along the view direction, which produces a linear spacing distribution. However, we know of no published algorithms that use a logarithmic parameterization for rendering shadow maps for real-time applications. Due to its nonlinear nature, a logarithmic parameterization introduces new problems that have not been explored. As shown in Figure 1.9, a logarithmic parameterization causes straight lines and planar surfaces to become curved. Current GPUs have no native support for these kinds of surfaces. The logarithmic parameterization could be approximated by adaptively tessellating the scene and using a programmable shader to apply a logarithmic transformation to the vertices, but this increases the vertex transformation load on the GPU and the complexity of the shadow rendering code. The logarithmic parameterization can also be implemented with brute-force rasterization in a fragment shader. While this approach can generate accurate results, it is more complex to implement and is much slower than standard rasterization. In addition, techniques for avoiding self-shadowing artifacts on planar surfaces do not work as well for the curved surfaces.

1.5 Thesis goals

This dissertation seeks to address these two issues. We aim to provide a more complete understanding of the capabilities and limitations of the various warping and partitioning methods. This will permit developers to more easily select the algorithm that meets the performance and error requirements of their particular applications.

We also seek an improved shadow map algorithm for real-time applications. The algorithm should have similar performance and implementation complexity as existing algorithms but have lower error. Reducing error is important not only for improving visual quality, but also for increasing performance. Shadow map rendering does not usually involve expensive shaders so the performance is often bandwidth limited. An algorithm with lower error can meet a specific error bound with less shadow map resolution, and thus requires less storage and bandwidth.

We show that a combination of a logarithmic and perspective parameterization can be used to produce such an algorithm. In addition, we also show that such a parameterization can be implemented on GPUs at low cost via incremental modifications that leverage existing graphics hardware architectures. This leads us to our thesis statement.

1.5.1 Thesis statement

Logarithmic perspective shadow maps (LogPSMs) can produce significant error reductions over existing shadow map algorithms for real-time applications. The logarithmic rasterization required to support LogPSMs can be implemented with performance similar to linear rasterization through incremental modifications to current graphics hardware.

1.6 Overview of our approach

1.6.1 Scope

In this thesis we restrict our attention to hard shadows, though our methods may also be useful for algorithms that use shadow maps to generate soft shadows. Because we target real-time applications, we concentrate mainly on scene-independent algorithms that require

only a small number of partitions (< 10). These algorithms are the simplest and have the best performance in practice. Specifically, this means that we do not consider algorithms like adaptive shadow maps, although these can produce lower error than scene-independent algorithms. Recently proposed algorithms based on adaptive shadow maps report near real-time performance for simple scenes (Lefohn et al., 2007). But it is difficult to say when graphics hardware will be fast enough that adaptive algorithms can be competitive with the simpler, scene-independent algorithms used on today's GPUs. LogPSMs aim to provide the same performance as the simpler algorithms, but with lower error.

To establish the benefit of LogPSMs, we first perform an analysis of shadow map error. We seek to minimize the magnitude of the undersampling error that causes jagged shadow edges, commonly referred to as aliasing. Aliasing error can be factored into two parts: *perspective aliasing* which is due to the tapered shape of the view frustum, and *projective aliasing* which is related to the orientation of the surfaces in the scene relative to the light and eye. Minimizing projective aliasing error requires a scene-dependent analysis and the use of adaptive techniques that perform a large number of render passes. Therefore we seek only to minimize perspective aliasing error. In practice, this means that we may not completely eliminate aliasing in most images, but we seek to reduce it significantly while maintaining good performance.

In our analysis we focus primarily on bounding the worst case error. The actual error in the image depends on the position of the shadows inside the view frustum. In an interactive application where the view is unconstrained and the scene geometry is arbitrary, the shadows may appear in any part of the view frustum. Therefore we measure the perspective aliasing error over the entire frustum using the L_∞ norm, or max norm. Using this metric we can give scene-independent guarantees on the worst case error.

1.6.2 Logarithmic perspective parameterization

We formulate the LogPSM parameterization as a perspective projection, followed by a logarithmic transformation that is applied in only one direction. Suppose that the light space y-axis is aligned with the eye's view direction, and that shadow map coordinates are repre-

sented by s and t. A perspective projection along y has the following form:

$$s_p = \frac{a_0 x + a_1}{y} \qquad t_p = \frac{a_2 y + a_3}{y}, \tag{1.1}$$

where $a_0 \ldots a_3$ are constants. This projection can produce a good spacing distribution in s, but the distribution in t can be improved by using a logarithmic parameterization along y:

$$t_l = b_0 \log(b_1 y), \tag{1.2}$$

where b_0 and b_1 are constants. Our parameterization transforms Equation 1.1 into Equation 1.2 by applying an affine transformation (a scale and translation) to t_p obtain $1/(b_1 y)$, applying the log to get $\log(1/(b_1 y))$, and scaling by $-b_0$ to get $b_0 \log(b_1 y)$. Our logarithmic perspective parameterization has the following form:

$$t_{lp} = c_0 \log(c_1 t_p + c_2), \tag{1.3}$$

where c_0, c_1, and c_2 are constants. The main advantage of this formulation is that the perspective portion of the parameterization can be handled by the standard graphics pipeline without any modifications. Most importantly we do not have to clip curved primitives. To support the logarithmic transformation, only the rasterizer needs to be modified. (We include triangle setup, which also needs to be modified, as part of the rasterizer.)

Another advantage of our formulation is that it is easy to extend existing perspective warping algorithms to use the logarithmic perspective parameterization. In this thesis we extend three different algorithms to use our parameterization: a single shadow map algorithm similar to light-space perspective shadow maps (LiSPSMs) (Wimmer et al., 2004), a z-partitioning algorithm similar to parallel-split perspective shadow maps (PSSMs) (Zhang et al., 2006a), and a face partitioning algorithm similar to perspective-warped cubemaps (Kozlov, 2004). Each of these algorithms has different performance and error tradeoffs. Like all single shadow map algorithms, our single shadow map LogPSM produces the lowest error when the light direction is perpendicular to the view direction, but reverts to a uniform parameterization as

the light direction approaches the view direction. Adding z-partitioning reduces the error for the bad light configurations but at an added cost. The face partitioning algorithm is slightly more involved, but produces the lowest error over all light directions. In addition, it can be used for both directional lights and omnidirectional point lights for almost the same cost.

We demonstrate the benefit of these algorithms for several environments of varying complexity using a fragment program to simulate logarithmic rasterization. Our unoptimized simulator is quite slow but still interactive. With hardware acceleration, however, the Log-PSM algorithms would have performance similar to the existing algorithms upon which they are based.

1.6.3 Logarithmic rasterization

In order to implement logarithmic rasterization on GPUs, we propose several incremental modifications to current hardware architectures. We need to modify the rasterizer, polygon offset, and depth compression. The rasterizer in current high performance GPUs use edge equations to compute which pixels are covered by a primitive. It generally operates in two stages, a coarse stage that identifies potentially covered tiles of pixels, and a fine stage that computes coverage at each pixel. We extend the rasterizer to handle the edge equations for the curved edges generated by the logarithmic transformation. Our logarithmic rasterizer can utilize the same tile traversal and culling optimizations used by current rasterizers. Polygon offset, typically used to avoid self-shadowing artifacts, and depth compression also need to be modified.

Modern GPUs are typically the result of many years of careful optimization and tuning. Our modifications leverage existing hardware designs, which makes them easier to implement at low cost. Rasterizing with curved edge equations requires a modest amount of additional computational power, but can provide significant bandwidth savings. Thus these modifications align well with current hardware trends, where computational power continues to outpace memory bandwidth (Dally & Poulton, 1998).

1.7 Summary of results

The primary results of this dissertation can be divided into three main categories. First, we have a number of results related to the theoretical analysis of the error of shadow map methods. Second, we use this analysis to arrive at several algorithmic results which can be used to improve existing methods as well as LogPSMs. Third, we propose modifications to existing hardware to support LogPSM rendering. We will discuss each of these categories in more detail.

1.7.1 Theoretical analysis of shadow map algorithms

We analyze several shadow map algorithms both to better understand their behavior and to determine the benefits of LogPSMs over existing methods. Specific results include:

- **Equations for error from both point and directional light sources in general position:** Previous error analysis has focused primarily on overhead directional light sources. We derive equations that can be used to analyze the error for both types of light sources in any position.

- **Storage factor error metric:** We introduce a new metric called the *storage factor* for characterizing the error of a shadow map algorithm. The storage factor is the size of the shadow map in texels relative to the size of the image in pixels that would be required to remove perspective aliasing. This metric enables us to minimize the maximum error while taking into account the error in both shadow map directions simultaneously.

- **Error bounds:** Using the aliasing error equations we compute error bounds for different shadow map algorithms in terms of the storage factor.

We use this error analysis to compare a range of algorithms and to answer such questions as "When is a combination of face partitioning and z-partitioning better than z-partitioning alone?"

1.7.2 Algorithmic results

The analysis of shadow map aliasing leads to several improvements for shadow map algorithms.

- **LogPSM algorithms:** We show how to extend existing shadow map algorithms to incorporate the logarithmic perspective parameterization. We also demonstrate the benefit of these algorithms using a fragment program to simulate logarithmic rasterization.

- **Resolution redistribution:** Common practice is to use a square texture for the shadow map. However, the error in each shadow map direction is not necessarily the same. For partitioning methods, the error may also vary from one partition to the next. We minimize the overall error by redistributing the shadow map resolution according to the error. For example, a partition with high error will get more resolution than one with less error. The resolution is also divided between the shadow map directions in proportion to the error in each direction. We also provide a parameter to bias the resolution distribution toward one direction or the other to give the user more control.

- **Improved parameter selection for warping algorithms:** When a warping algorithm is used with a single shadow map covering the entire view frustum, the strength of the warping must be reduced as the light direction approaches the view direction in order to avoid excessive error. In some configurations, the current warping methods can produce error exceeding that of a standard shadow map. Using our error analysis we formulate an improved parameter selection function that keeps the error below that of a standard shadow map in the higher error configurations while maintaining low error for other light positions.

- **Pseudo-near plane algorithms:** When a surface is too close to the viewer it will be clipped by the near plane. To avoid this problem, developers typically choose a conservative value for the near plane distance that is as small as possible. However, this has an adverse effect on shadow map error, which depends on the far to near plane distance ratio. To minimize the error and to avoid wasting shadow map samples

on unoccupied regions of the view frustum, the near plane distance should ideally be set as large as possible, i.e. at the depth of the closest visible surface. We reconcile these two opposing criteria for setting the near plane distance by using a pseudo-near plane. When we compute the z-partition locations and/or warping parameters, we minimize the maximum error in only the part of the view frustum beyond the pseudo-near plane rather than the entire view frustum. Regions between the actual near plane and the pseudo-near plane are still covered by the shadow map but are permitted to have higher error. The pseudo-near plane gives the user a way to provide the shadow map optimization with rough information about parts of the view frustum that are more likely to contain geometry.

- **Approximation of LogPSM with z-partitioning:** On hardware without logarithmic rasterization, the closest we can come to the same error as a LogPSM is with z-partitioning. Using the error bounds that we derive, we can determine how many z-partitions are necessary to meet a user specified threshold on the error relative to that of a LogPSM.

We also discuss several practical considerations and aspects of implementing shadow map algorithms that are useful for developers.

1.7.3 Hardware support for LogPSMs

The logarithmic rasterization required by LogPSMs is not supported on current graphics hardware. Because the logarithmic perspective parameterization is formulated in such a way that it fits easily into the graphics pipeline, graphics hardware can be easily extended to support logarithmic rasterization through incremental enhancements to existing graphics hardware. These enhancements include:

- **Rasterization on a nonuniform grid:** Rendering with the curved edge equations produced by logarithmic rasterization can be thought of as rasterizing linear edges on a regular grid with nonuniform spacing in one direction. We extend rasterization on a uniform grid to handle nonuniform spacing. Using this approach, logarithmic

rasterization can exploit the same optimizations used by linear rasterization. It also provides water-tight rasterization of a mesh without holes or double-hitting of pixels.

- **Generalized polygon offset:** Polygon offset is used with shadow maps to avoid self-shadowing artifacts. The polygon offset on current GPUs is constant over a primitive. For logarithmic rasterization we need an offset that varies linearly in one direction. This can be implemented with a small change to constants of the equation used to compute depth at each pixel.

- **A new depth compression scheme:** Depth compression is important for reducing memory bandwidth requirements. Existing depth compression techniques that exploit the planarity of primitives perform poorly with logarithmic rasterization because planar primitives become curved. We present a depth compression scheme that is better suited for logarithmic rasterization, which, in practice, also works well for linear rasterization.

These enhancements enable logarithmic rasterization at fill rates comparable to the linear rasterization on current GPUs. Thus LogPSMs should have performance characteristics similar to the existing algorithms upon which they are based.

1.8 Thesis organization

In Chapter 2 we review the start-of-the-art in shadow map algorithms. We discuss the various kinds of shadow map artifacts and the methods that have been proposed to handle them. We also discuss alternative shadow generation algorithms. Chapter 3 introduces the equations for quantifying aliasing error at a point in both 2D and 3D. In Chapter 4 we discuss how to minimize the aliasing error using minimal shadow map resolution by computing an appropriate parameterization. The parameterization we derive is not suitable for practical use, but serves as a baseline for evaluating simpler parameterizations. We also discuss global error metrics over the entire view frustum and all light directions. Chapter 5 contains the analysis of the various warping and partitioning schemes and their combinations. We study the behavior of these algorithms for different warping parameter values, view frustum parameters, and light positions. We present the details of the LogPSM parameterization and show empirical results.

In addition, we propose several improvements to existing algorithms. In Chapter 6 we discuss various implementation details for existing shadow map algorithms and LogPSMs. Chapter 7 describes the hardware enhancements required to support logarithmic rasterization. Finally, we conclude in Chapter 8 with a brief summary and indicate directions for future work.

CHAPTER 2

Handling shadow map artifacts

Much of the work related to shadow maps addresses the artifacts to which they are prone. Depth-based shadow maps were originally proposed by Williams in 1978 (Williams, 1978). They were popular initially for offline rendering. Segal et al. (1992) later showed how shadow maps could be implemented on standard graphics hardware using an extension of texture mapping, which led to their adoption for interactive applications. The increased demand for visual realism in interactive applications has spurred a flurry of activity in recent years to address shadow map artifacts.

In this chapter we present a classification of shadow map artifacts. We then present the various existing algorithms according to the artifacts that are designed to handle.

2.1 A classification of shadow map artifacts

Shadow maps use discrete samples in both time and space to compute the visibility of the light on a surface. As a sampling method, it is prone to sampling artifacts. Figure 2.1 illustrates how some of these artifacts can arise when using discrete samples to represent an arbitrary 1D signal. Aliasing is introduced by the initial sampling of the signal when the sampling frequency is too low to capture high frequency details, causing the high frequencies to show up as lower frequency artifacts. Other errors can occur when resampling a sampled signal. Conceptually, resampling reconstructs a piecewise continuous signal from the samples, from which is drawn additional discrete samples. Both the filter used for reconstruction and the resampling frequency can affect the output.

For hard edged shadows, light visibility can be thought of as a piecewise continuous signal whose value is binary, either 0 or 1. Shadow maps do not encode the "visibility signal"

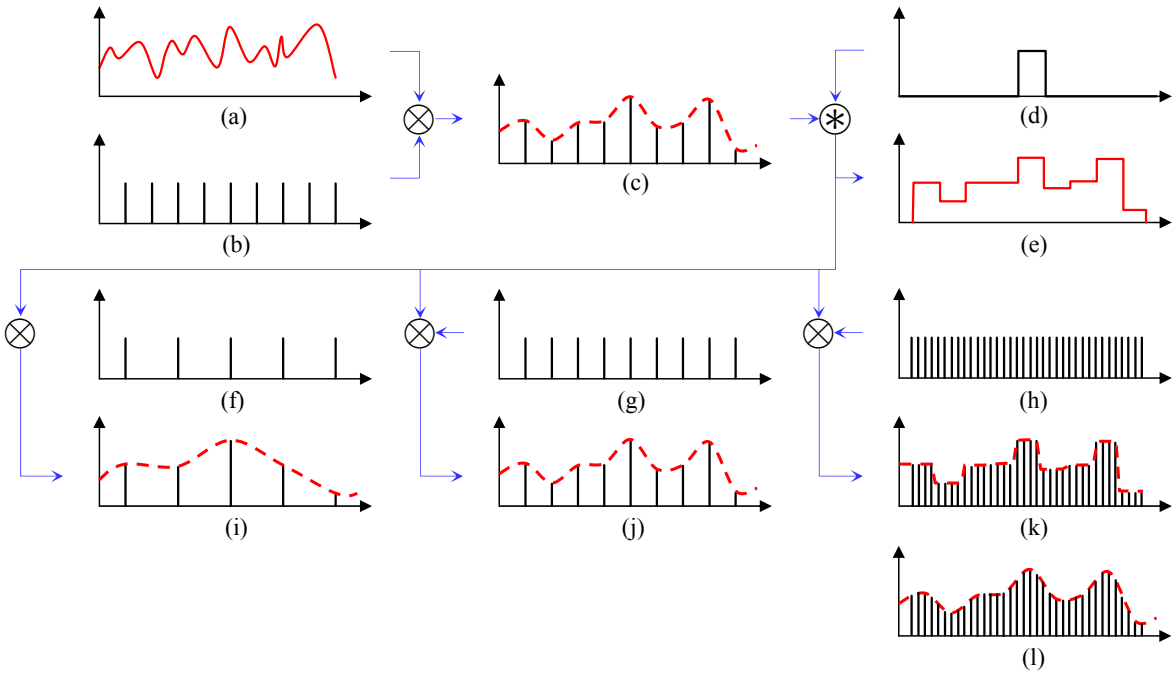

Figure 2.1: ***Resampling a sampled signal.*** *A 1D signal (a) is initially sampled by multiplying by a train of uniformly spaced delta functions (b) with frequency ω_0, which is lower than the Nyquist limit for this signal. This produces a discrete, sampled signal (c) which is an aliased version of the original. The sampled signal itself is now band-limited to $\omega_0/2$. The signal reconstructed from the sampled signal using an ideal band pass filter is shown with the dashed line. Using nearest-neighbor sampling in effect reconstructs the sampled signal by convolving with a box function that is one sample wide (e). This reconstructed signal (f) is then resampled at different frequencies. A resampling frequency of $\omega_0/2$ (g) is too low and causes aliasing in the resampled signal (j). Resampling at a frequency of ω_0 (h) results in a perfect resampling of the sampled signal. Resampling with a frequency of $4\omega_0$ (i) introduces high frequency reconstruction errors (k) not present in the original sampled signal. (l) Resampling a perfect reconstruction of the sampled signal at a high resampling rate.*

directly for each surface. Instead, the binary samples resulting from the shadow map depth comparisons are used to reconstruct the visibility signal. In analogy to the 1D signal example in Figure 2.1, the rendering of the shadow map can be thought of as the initial sampling and the shadow map queries generated during image rendering can be thought of as a resampling.

Ultimately, shadow map errors result from using discrete values to represent continuous ones. The following is a list of the different ways in which these errors are manifested:

Initial sampling aliasing: When rendering the shadow map, aliasing errors can occur if geometric features have higher spatial frequency than can be captured by the finite resolution of the shadow map, e.g. the needles of a pine tree in a shadow map covering

several square miles. Aliasing leads to strange patterns in the shadows or missing features. The silhouette edges that give rise to hard-edge shadows contain infinitely high frequencies that can never be sampled sufficiently to permit perfect reconstruction.

Resampling errors: When rendering the image, different parts of the shadow map are resampled at different frequencies. For instance, parts of the scene closer to the viewer are resampled at a higher frequency than parts of the scene the are far away. When the resampling frequency does not match the initial sampling frequency, two types of errors can occur:

- **Aliasing:** High frequency features in the shadow map alias when the resampling frequency is lower than the initial sampling frequency.
- **Reconstruction errors:** During resampling, the nearest sample in the shadow map is used to estimate the visibility at each queried location. If the resampling frequency is higher than the initial sampling frequency, a number of consecutive queries will use the same shadow map sample resulting in the familiar stair-cased edge artifacts at shadow boundaries (see Figure 1.6). These artifacts are commonly referred to as aliasing, though strictly speaking they may be more accurately described as reconstruction errors (Glassner, 1995, pg. 344). True aliasing occurs when high frequency energy appears in lower frequencies due to undersampling. These artifacts occur in areas where the resampling oversamples the shadow map (see Figure 2.1). Nearest-neighbor resampling in effect convolves the shadow map with a box function, introducing high frequency edges not originally represented in the shadow map. A perfect reconstruction would produce a smoother, lower frequency signal. This would still not give the correct result but is often preferred to the jagged reconstruction errors because the smooth signal resembles a penumbra. The smoothing of the signal is actually an initial sampling error instead of a reconstruction error. The sampled signal in the shadow map can be faithfully resampled even with nearest-neighbor reconstruction if the initial sampling frequency matches the resampling frequency.

Estimation errors: Because the shadow map is a discrete representation, the visibility for points between samples must be estimated from neighboring samples. When this estimate is inaccurate, errors can occur. Estimation errors show up as self-shadowing artifacts.

Depth quantization errors: Shadow maps store depth with some finite precision. Even when the shadow map query coincides exactly with a sample, depth quantization can lead to self-shadowing artifacts.

Temporal artifacts: If any of the preceding artifacts are present in an image, they can lead to temporal artifacts when they are not consistent from frame to frame. For instance, aliasing can cause shadows to flicker. The jagged shadow edges from reconstruction errors can appear to "crawl" when the light or objects in the scene move slowly. For quick movements, the effect is less noticeable.

Some of these errors are showed in Figures 2.2 and 2.3. The aim of most of the research in shadow maps since the original algorithm was proposed has been to address these errors. We will now discuss methods proposed to handle initial sampling, resampling, and estimation errors.

2.2 Handling initial sampling errors

Initial sampling aliasing is a problem for all sampling techniques. Some initial sampling aliasing artifacts can be mitigated by "prefiltering" the geometry through use of level-of-detail techniques which simplify the geometry and remove some of the high-frequency content. For example, if we have a plane with small holes casting shadows in the distance, some holes may be sampled and some may not, leading to temporal artifacts. If the holes are small enough, they would not be visible in the distance anyway, so the object may be simplified with a solid plane to avoid the artifacts. In general, however, the only way to accurately capture high-frequency geometric information is to increase the resolution of the shadow map.

Another approach to handling initial sampling aliasing is to use stochastic sampling (Cook, 1986). Stochastic sampling diffuses low-frequency aliasing patterns into high-frequency noise.

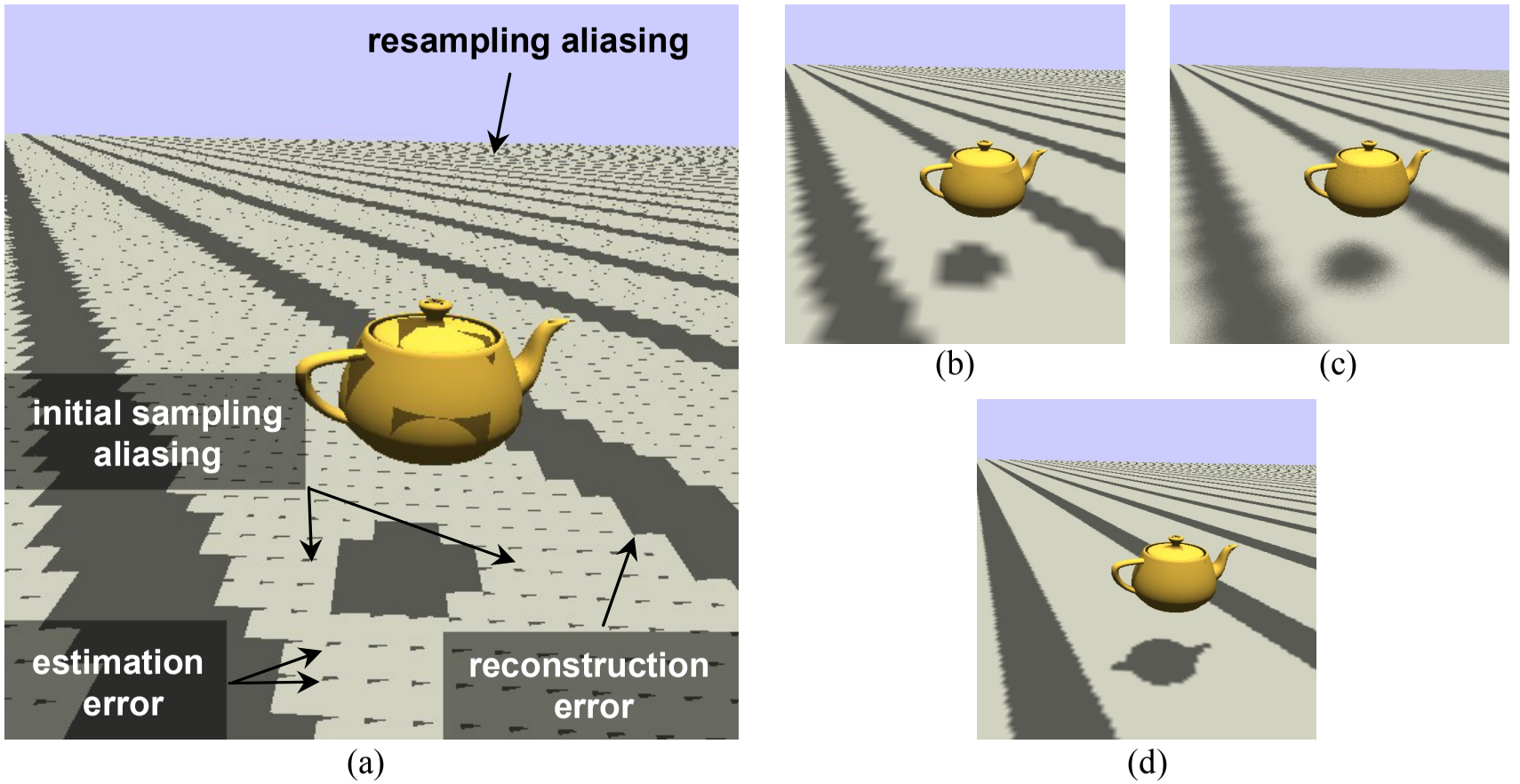

Figure 2.2: ***Shadow map errors.*** *(a) This image shows several kinds of errors that can occur with shadow maps. Shadows are cast by a grate above the scene (which creates the lines) and a teapot. Aliasing from the initial sampling causes features to be missed, like the teapot handle and spout. Resampling aliasing causes strange patterns in the undersampled regions in the background. Reconstruction errors result in jagged edges. Estimation errors cause false self-shadowing. (b) Shadow map bias alleviates estimation errors. Percentage closer filtering (PCF) removes some of the high frequency reconstruction errors resulting in smoothed out shadows. This image uses hardware PCF. (c) Stochastic PCF converts resampling aliasing into noise and breaks up the jagged reconstruction errors. (d) A high resolution shadow map reduces reconstruction errors and initial sampling aliasing, but resampling aliasing remains.*

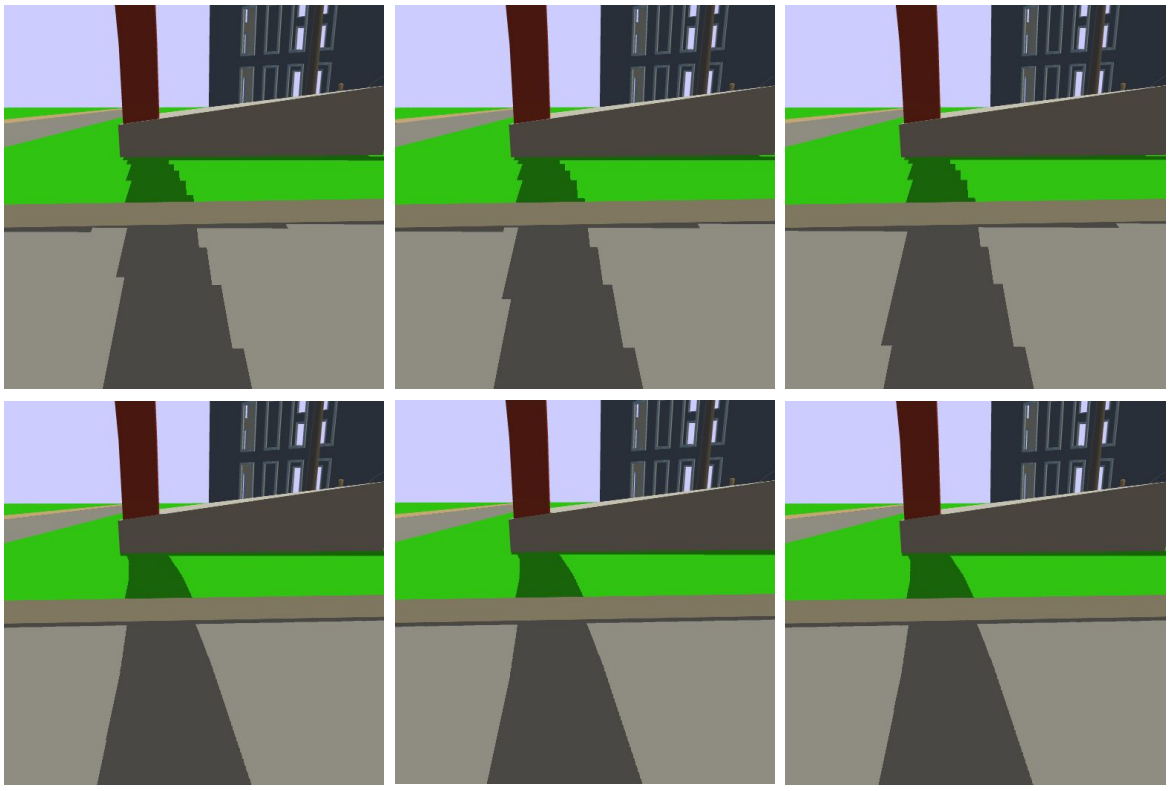

Figure 2.3: ***"Crawling" artifacts.*** *(Top) In this sequence of images the camera position changes slightly. The jagged reconstruction errors at the shadow boundaries shift even though the light and geometry is static. This is because the shadow map extents are fit to the camera. (Bottom) With sufficiently high resolution, no crawling artifacts are visible.*

To the human observer, noise is often less objectionable. Akenine-Möller et al. (2007) propose a hardware architecture for rasterizing triangles stochastically in time. This can be used to reduce temporal aliasing by rendering motion-blurred shadows.

2.3 Handling resampling errors

Standard texture mapping handles resampling errors with filtering techniques. Resampling aliasing can be handled efficiently by prefiltering the texture map using a technique such as mip-maps (Williams, 1983) or summed area tables (Crow, 1984). Reconstruction errors are typically handled with bilinear or higher order interpolation between sample values. However, these methods cannot be used directly with shadow maps. Shadow maps do not record visibility directly because it is not the same for each surface. Instead, the depth comparison

is used to compute the visibility signal independently on each surface. Because the depth comparison is nonlinear, the standard methods do not work properly. For example, consider an area of the image distant from the viewer which contains high frequency shadows. Over the extents of the scene that fall within a single pixel there may be alternating shadowed and unshadowed regions. Proper filtering over pixel's extents would give a light visibility somewhere between 0 and 1. If we stored the visibility signal on the surface, mip-mapping would give us this result. Instead we store depth. Mip-mapping depth values and then performing the depth comparison can only return 0 or 1. The same is true if we bilinearly interpolate depth values and then perform the depth comparison.

Resampling errors can be handled by using better reconstruction methods and by using a better initial sampling of the scene. In general, these techniques are orthogonal and can often be used in combination.

2.3.1 Improved reconstruction methods

Percentage closer filtering (PCF) (Reeves et al., 1987) first computes the visibility of samples in the neighborhood of a shadow map query and then filters the results. PCF can handle both resampling aliasing and reconstruction errors by choosing the appropriate size for the resampling neighborhood. To handle resampling aliasing, the neighborhood ideally should correspond to the projection onto the scene of the pixel that generated the query. For aliased regions, this neighborhood will cover a large number of shadow map samples. For regions where reconstruction errors occur, however, the neighborhood will be smaller than that of a single shadow map texel. Filtering with such a small neighborhood would offer no improvement. Thus the size of the neighborhood should be clamped to some minimum extent larger than 1 texel. Using a large neighborhood blurs the jagged shadow edges. This effect resembles the penumbra of a soft shadow and can be visually pleasing. It is not a true penumbra, however, because its width does not change with distance from the occluder. The more samples that are used for PCF, the smoother the edges become. Hardware implementations of PCF on current GPUs use bilinear interpolation instead of stochastic sampling. This produces a smooth transition from shadowed to unshadowed regions but preserves some blockiness along

the shadow edge. Figure 2.2 shows both the native hardware PCF and stochastic PCF applied using a fragment program.

Silhouette shadow maps (Sen et al., 2003) reconstruct a hard edge by augmenting the shadow map with extra information about the location of the silhouette edges of shadow casters. The silhouette edges are rendered into the shadow map storing at each texel the coordinates of a single point along the edge that falls within the texel. The point coordinates in the neighborhood of a shadow map query are used to reconstruct the silhouette edges. Depth information is used to determine whether the query lies on the shadowed or unshadowed side of the reconstructed edges. Only one point is stored per texel, so errors can occur when multiple edges fall within the same texel.

Variance shadow maps (Donnelly & Lauritzen, 2006) store an approximation of the distribution of depth values within a texel. These values can be filtered with standard techniques. The technique computes the exact result of performing PCF for a planar occluder parallel to a planar receiver. For other configurations, it gives only an upper bound on the fraction of the filtered region that is unoccluded, but it often works well in practice. However, when the variance in depth is high, the technique can cause light to leak into the shadows causing strange artifacts.

Convolution shadow maps (Annen et al., 2007) store an approximation of the visibility function instead of depth values. The visibility function generated by the depth comparison can be represented as a Heaviside step function. This function is expanded in terms of the Fourier basis functions. The advantage of this representation is that the nonlinear depth comparison can be "linearized" so that the shadow map can be prefiltered. A convolution shadow map is constructed by using a normal shadow map to generate a number of basis images. Prefiltering can be applied to visibility function simply by prefiltering the basis images. One disadvantage of this algorithm is that a large number of basis images may be required to obtain hard edges.

Scherzer et al. (2007) reuse the shadowing information from previous frames to provide both temporal smoothing and more accurate reconstruction of shadow edges. A history buffer stores for each pixel in the image a weighted average of the shadow map query results for

previous frames, as well as the eye space depth of the fragment of the last frame. The history buffer is updated each frame using a weighted combination of the buffer's previous value and the query results computed from the shadow map for the current frame. The updated history buffer is then used to render the shadows. The weights for query results are computed independently for each pixel. Each fragment is transformed back into the eye space of the previous frame and its depth is compared to that stored in the history buffer. If the depth is not within a user specified tolerance, then the previous history is assigned a weight of 0 and the query result for the current frame is assigned a weight of 1. This is necessary to properly handle disocclusions. Otherwise, a weight is computed using the confidence of the current query result. The confidence is based on the distance of the query location from the center of the corresponding texel in the shadow map. Query locations far from the center are assigned low confidence as the results of these queries is less likely to be accurate and should have less influence on the history buffer. Jittering the shadow map every frame ensures that over time, high confidence query results appear, which causes the history buffer to converge to the exact shadow edge. However, the algorithm may not converge for moving shadows caused by moving objects or a moving light source, leading to overly smoothed or smeared shadows.

2.3.2 Improved sampling methods

Another way to handle reconstruction errors without increasing the number of samples is to use a better sample distribution in rendering the shadow map. The more closely the initial sampling matches the shadow map query distribution, the lower the reconstruction errors. A better sampling can be obtained by fitting the shadow map to relevant portions of the scene, warping the shadow map, partitioning, and irregular sampling. We will now discuss these approaches.

2.3.2.1 Shadow map fitting algorithms

Fitting a shadow map consists of choosing the orientation and spatial extents of the light frustum used to render the shadow map so as to maximize the use of its spatial and depth resolution. Brabec et al. (2002) observe that the depth range and spatial extent of the

shadow map can be restricted to the visible portions of the scene. To make the best use of depth resolution they set the near and far planes of the light frustum to tightly enclose the visible objects. The depths of occluders between the light and near plane are clamped to 0, thus ensuring that their occlusion is represented in the shadow map while maximizing the available depth resolution for the visible regions. To obtain bounds on the visible portions of the scene, they render the scene from the light's point of view using projective texturing to mark the regions that lie inside the view frustum. The results are read back and used to compute the tightest fitting bounding box for the visible regions. Readbacks from the GPU tend to be expensive. A cheaper, though less accurate way of obtaining tight bounds on the visible region is to compute the intersection of view frustum, the scene bounding box, and for lights like a spotlight that have restricted extents, the light bounds.

As Martin and Tan (2004) observe, the tightest bounding rectangle of the visible regions does not always change continuously from one frame to the next. If the shadow map resolution is sufficiently high to eliminate aliasing and reconstruction errors, then a discontinuous shift in the orientation of the bounding rectangle may not produce noticeable changes in the shadows. Due to limitations on shadow map resolution, however, there are usually some errors visible that will "pop" when the change is not continuous. Martin and Tan propose that the shadow map be oriented with respect to the view frustum by aligning one of the axes of the shadow map with the view direction of the camera. This produces a consistent orientation for the reconstruction error artifacts as the light and camera move. One problem with this approach is that the jagged edges on shadows from static objects with a static light can still appear to "crawl" when the camera moves slowly. The shadows should remain fixed. This problem can be avoided by aligning the orientation of the shadow map with one of the world axes and shifting the extents of the shadow map from frame to frame in increments corresponding to world space width of a shadow map texel. Jagged edges will still crawl for moving objects or when the light moves, however. For static scenes with a moving camera it may also be desirable to render a shadow map that is larger than necessary so that it may be reused over multiple frames.

2.3.2.2 Warping algorithms

Brabec et al. (2002) used a nonlinear warping for omnidirectional light sources but not for the purpose of correcting shadow map errors. The use of shadow map warping to reduce reconstruction errors was first introduced with perspective shadow maps (PSMs) (Stamminger & Drettakis, 2002). A PSM is rendered in the post-perspective space of the camera. In post-perspective space, the view frustum is warped to a cube causing objects close the viewer to become larger and distant objects to become smaller. Rendering the shadow map in this space can increase the sample density near the viewer. The exact warping depends on the way the shadow map projection matrix is setup, and this is not explicitly specified in the original paper. The author's webpage provides a "recipe" for PSMs that suggests using a perspective projection centered on the center of the view frustum cube. Using a projection with a different image plane angle will produce a different warping (see Appendix A). Working in the post-perspective space of the camera is complicated by the fact that the perspective transformation maps some finite points to the infinity plane and vice versa. This means that directional lights can become point lights and vice versa, the depth order along a line can become inverted, and occluders behind the camera wrap around the plane at infinity. To avoid the latter problem, PSMs are rendered in the post-perspective space of a virtual camera whose eye position is shifted back far enough that all objects that occlude the visible portions of the scene lie in front of it. Kozlov (2004) proposes a formulation for the shadow map's projection matrix that can wrap around the plane at infinity so as to avoid the virtual camera shift.

Light-space perspective shadow maps (LiSPSMs) (Wimmer et al., 2004) are a generalization of PSMs that avoid the difficulties of post-perspective space. LiSPSMs warp the shadow map using a perspective projection that is oriented perpendicular to the light direction and aligned with the view direction. For point light sources, the light's projection matrix is applied first such that the light position is moved to infinity and becomes a directional light. Then the perspective warping is applied. The strength of the warping is controlled by the near plane distance of perspective projection n'. For an overhead directional light, setting n' equal to the near plane distance of the view frustum is equivalent to using a PSM. Setting

$n' = \infty$ produces a standard shadow map. LiSPSMs use a warping parameter that minimizes the maximum aliasing error along the shadow map dimension that is aligned with eye's view direction.

Trapezoidal shadow maps (TSMs) (Martin & Tan, 2004) are very similar to LiSPSMs except that they use a use a different formulation for the warping parameter. They compute the warping parameter using on a heuristic that maps a user-specified fraction of the view frustum (e.g. 0.5) to a user-specified fraction of the shadow map (e.g. 0.8). In this way, more shadow map samples can be devoted to regions near the viewer.

One drawback of all these methods is that the warping is not effective for all light positions. For directional light sources, the warping is generally most effective for light positions perpendicular to the view direction. As the angle γ between the light direction and the view direction approaches 0, a PSM reverts back to a standard shadow map. LiSPSMs also revert back to a standard shadow map to avoid excessive error by ramping off the warping as a function of γ.

Chong and Gortler (2004) use a general projective transform to establish a one-to-one correspondence between pixels in the image and the texels in the shadow map, but only for a single plane within the scene. They use a small number of shadow maps to cover a few prominent, planar surfaces. This can be viewed as a type of partitioning.

Chong and Gortler (2006) also propose a framework for computing an optimal perspective projection for rendering a scene. They note that the perspective warping of previous methods is equivalent to rendering a standard shadow map with an image plane that is not orthogonal to the light's view direction. Their method optimizes the orientation of the image plane according to an error metric computed for pixels near shadow boundaries. These pixels are found by rendering a low resolution image of the scene with shadows, reading it back from the GPU, and analyzing it on the CPU. (For more on the relationship between the warping parameter and the orientation of the image plane see Appendix A.)

2.3.2.3 Partitioning algorithms

Partitioning algorithms use multiple shadow maps to control the shadow map sample distribution. Several algorithms use an approach which we call z-partitioning, where the view frustum is partitioned along its z-axis (view direction) into subfrusta and a separate shadow map is rendered for each one. Plural sunlight buffers (Tadamura et al., 1999) use this approach for offline rendering of shadows from the sun with penumbrae. They partition the view frustum at intervals that increase geometrically with distance from the eye. Cascaded shadow maps (Engel, 2007) use a similar technique but are intended for use with real-time applications. Parallel-split shadow maps (PSSMs) (Zhang et al., 2006a) partition the view frustum using the average of the split locations generated by using geometric and uniform intervals.

Kozlov (2004) uses a cube map in the post-perspective space of the camera as a way to improve PSMs. A separate shadow map is fit to the back faces of the unit cube of the post-perspective view frustum. In world space this is equivalent to fitting perspective-warped shadow maps to side faces and standard shadow maps to end faces. We call this face partitioning. The benefit of this approach over single shadow map warping is that the warping can be used for all light positions.

z-partitioning and face partition are based solely on the view frustum and do not take into account the positions and orientations of surfaces in the scene. Adaptive partitioning schemes can take these factors into account in order to refine the shadow map where the resolution is needed most, and therefore tend to produce higher quality shadows.

Adaptive shadow maps (ASMs) (Fernando et al., 2001) represent the shadow map as a quadtree of fixed resolution tiles. The quadtree is progressively refined near shadow edges. Each iteration, the algorithm renders the scene, encoding polygon IDs in the color channels using a special mipmap texture to encode the projected area of a shadow map texel in the alpha channel. The resulting image is read back and analyzed using a cost metric to determine where the shadow map should be refined.

Resolution matched shadow maps (RMSMs) (Lefohn et al., 2007) also use a quadtree to store shadow map tiles. Unlike ASMs, however, the quadtree is not iteratively refined at

shadow edges. Instead, an entire quadtree with sufficient resolution at every pixel is produced every frame. The view from the eye is first rendered using a fragment program that encodes for each pixel a request for a tile of sufficient resolution, referred to as a page. The page requests for adjacent pixels on the same surface are likely to be the same. RMSMs leverage this coherence to efficiently remove duplicate page requests before reading them back to the CPU. Spatially adjacent pages are then grouped together by binning them into superpages. The algorithm renders the superpages and copies out the relevant pages into the quadtree. RMSMs have several advantages over adaptive shadow maps. RMSMs perform the scene analysis on the GPU which greatly reduces the size of the readback. ASMs reduce this cost by rendering a lower resolution image, but this can cause some shadow edges to be missed. The RMSM algorithm is also more suitable for dynamic scenes. The quality of ASMs may be quite low when the light moves because the quadtree is invalidated every frame and a large number of frames are required before the quality converges. By not refining at shadow edges, RMSMs trade-off faster generation of high quality shadows at the cost of increased memory requirements.

Queried virtual shadow maps (QVSMs) (Giegl & Wimmer, 2007b) are another adaptive partitioning scheme based on a quadtree refinement. QVSMs first render the shadows using a single shadow map tile at the root of the quadtree. The shadow map is refined into 4 subtiles and the shadows are updated each subtile. Occlusion queries are used to count the number of pixels that change their value. Subtiles are further refined until the number of changed pixels drops below a user specified threshold. QVSMs also use warping (LiSPSM) to achieve better quality. To avoid the high cost of repeated scene renderings, QVSMs use a variation of the deferred shading, using an eye-space depth buffer to compute the positions of shadow map queries. While this approach can offer greater flexibility, it also means that tiles cannot be cached and reused over multiple frames.

Fitted virtual shadow maps (FVSMs) (Giegl & Wimmer, 2007a) use a noniterative quadtree partitioning similar to RMSMs, except that the quadtree is more coarse grained. RMSMs use a tile size of 32×32. FVSM uses tiles up to the size of the largest allocatable texture (4096×4096). The scene analysis is carried out on the CPU using the information encoded

in an image that is read back from the GPU. The benefit of doing the analysis on the CPU is that more sophisticated algorithms can be used to determine where to refine the quadtree. The encoded image is rendered at low resolution (256×256) to minimize the cost of the readback, but this can cause important information to be missed, leading to errors. Like QVSMs, FVSMs use deferred shading for the shadows.

Tiled shadow maps (Arvo, 2004) are an adaptive scheme that partition a single, fixed-resolution shadow map into tiles of different sizes guided by an error measurement heuristic. This results in simpler data structures, but can cause some tiles to be overly compressed or elongated along one or both directions. Chong (2003) presents an algorithm for rendering 1D shadow maps in a 2D scene that uses a greedy partitioning scheme combined with optimized warping to minimize an error metric. Unfortunately, this approach is more difficult to generalize to 3D. Forsyth (2006) proposes a partitioning technique that clusters objects requiring similar sampling densities into multiple light frusta using a greedy algorithm. However, this method is based only on distance from the light source and therefore does not handle aliasing due to surface orientation.

2.3.2.4 Irregular sampling

The sample positions for the irregular z-buffer algorithm (Johnson et al., 2004; Aila & Laine, 2004b) can be specified explicitly. By choosing the positions of shadow map queries as the sample locations for the shadow map, reconstruction errors can be completely eliminated. The main drawback of this approach is that it uses irregular data structures that require fundamental changes to graphics hardware in order to be implemented efficiently (Johnson et al., 2005).

2.3.3 Object-space hybrids

Pure object-space shadow algorithms, such as shadow volumes, do not have resampling problems. Some hybrid algorithms combine object-space techniques with shadow maps to reduce aliasing. McCool et al. (McCool, 2000) find edges in a shadow map and use these to construct shadow volumes. The shadow map's image-based representation implicitly performs occluder

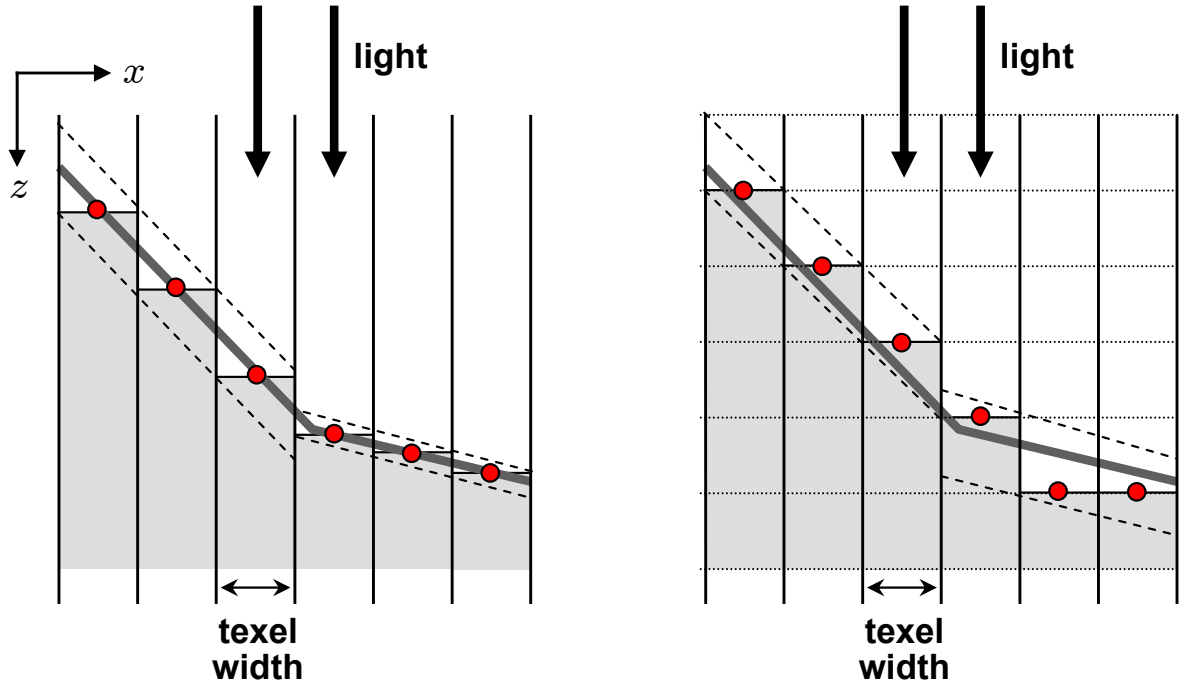

Figure 2.4: ***Self-shadowing artifacts.*** *(Left) The shaded region represents the discretized occlusion representation created by sampling the scene when rendering the shadow map. For points within a band around the boundary of the occlusion representation (dashed-line) the visibility of the light cannot be computed accurately. The width of this band depends on the slope of the surface. (Right) Self-shadowing can also be caused by the quantization that inevitably results from representing depth values with finite precision. The dotted lines represent discrete depth values.*

fusion. This eliminates redundant shadow volumes from being created inside other shadow volumes and reduces the fill-rate consumption. Chan and Durand (Chan & Durand, 2004) use a shadow map to restrict shadow volume rendering to regions near shadow edges. Both of these methods produce artifact-free edges, but may miss small features if the shadow map resolution is inadequate. Govindaraju et al. (2003) use shadow polygons for the areas of the image with the worst artifacts and a shadow map everywhere else.

2.4 Handling estimation errors (self-shadowing artifacts)

A standard shadow map stores a sampled representation of the occluding surfaces in the scene. Within a certain distance of the occlusion representation, the results of shadow map queries cannot be computed reliably due to discretization caused by sampling and quantization of

depth values (see Figure 2.4). This leads false self-shadowing. Williams (1978) adds a constant bias to depth values that in effect pushes the occlusion representation far enough away from the actual occluder surfaces so as to avoid self-shadowing artifacts. Choosing a bias that is too small fails to correct the problem, while choosing a bias that is too large causes a noticeable shift in the position of the shadows and can cause light to "leak" around silhouette edges. The amount of bias required is proportional to the depth slope of the polygon, so a single constant bias may not be suitable for a general scene. Instead, the bias should be computed taking the depth slope into account.

Woo (1992) stores the midpoint of the first and second depth layers to keep the occlusion representation away from the surfaces in the scene. Near silhouette edges where the occlusion representation approaches the surfaces, errors may occur. Wang and Molnar (1994) store the depth of the second surface in the shadow map. This technique is based on the observation that the second surface of closed objects is back facing. Even if the results are inaccurate for queries on the second surface, the lighting equations will shade the surface as shadowed anyway. This method also has problems near silhouette edges. Dual layer shadow maps (Weiskopf & Ertl, 2003) generalize these approaches and compute a bias based on a function of the first and second front-facing depth layers.

Gradient shadow maps (Schüler, 2006) bilinearly interpolate depth values adjacent to a query location prior to performing the depth comparison. This can produce a more accurate occlusion representation. Interpolation produces an occlusion surface across depth discontinuities, but the errors that this causes are no worse than those incurred by discretization. A fuzzy depth test that makes a transition from 0 to 1 over a range near the boundary of the occlusion representation can make self-shadowing artifacts less noticeable when they do occur. The algorithm uses the gradient to compute a bias. Grant (1992) computes a more accurate occlusion representation by storing the plane equations of the occluder surface instead of depth. This approach has problems for texels covered by multiple primitives because only one plane equation is stored per pixel. Hourcade and Nicolas (1985) store polygon IDs instead of depth. Polygon IDs are not subject to the same imprecisions as depth comparisons. Errors can still occur when a texel is covered by more than one polygon because only one ID

is stored per texel. Using a single ID for an entire object moves the self-shadowing problems from polygon to object boundaries, but then care must be taken to decompose objects into convex pieces with different IDs in order to get true self-shadowing.

Brabec et al. (2002) address the self-shadowing caused by quantization of depth values by tightly fitting the available depth range to the visible portions of the scene. They also remap the depth values to obtain a uniform distribution within the depth range. This is necessary because the perspective projection used to render a shadow map for a point light distributes depth values proportional to $1/z$. This causes most of the depth values to be bunched up close to the near plane and causes large quantization errors elsewhere. Remapping the depth values ensures that the maximum quantization error is minimized and that the quantization error is distributed uniformly over the entire depth range.

2.5 Summary

Though many methods have been proposed to address shadow map errors, no one technique can currently provide high-quality hard-edge shadows in complex, dynamic scenes for real-time applications. An effective shadow map algorithm will likely require a combination of several of the techniques presented in this chapter. Fortunately, many of these techniques are complimentary. For instance, the bias used to handle self-shadowing artifacts can be smaller when using sample redistribution techniques like warping and partitioning because the spacing between the samples tends to be smaller. However, combining filtering with these techniques can be more complicated. With warping techniques, the size and even the shape of the filter kernel needs to vary over the shadow map. For partitioning schemes, care must be taken to overlap the partitions slightly so as to get the correct filtered results near the partition boundaries.

Logarithmic perspective shadow maps are a step toward higher shadow map quality for real-time applications. With proper hardware support they can have the same good performance characteristics of existing warping algorithms, but provide higher quality.

CHAPTER 3

Shadow map aliasing error

In this chapter we describe how to quantify the aliasing error that gives rise to jagged shadow boundaries. We will use the equations we derive here in subsequent chapters to analyze various shadow map parameterizations. We first derive the equations for aliasing error in a 2D environment. To our knowledge, ours is the first analysis that quantifies perspective aliasing error for point lights in general position. The analysis also extends to directional lights. The analysis in previous work is typically performed for a directional light for a few specific configurations (Stamminger & Drettakis, 2002; Wimmer et al., 2004; Lloyd et al., 2006). A notable exception (Zhang et al., 2006b) computes perspective aliasing error for directional lights over a range of angles, but only along a single line through the view frustum. We also extend our 2D analysis to 3D.

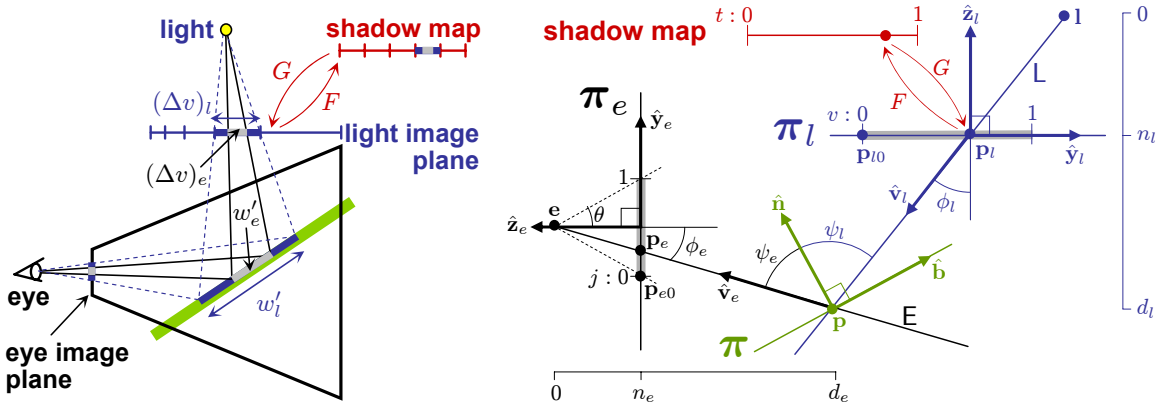

Figure 3.1: ***Computing aliasing error.*** *(Left) Sample locations corresponding to shadow texels and image pixels. Aliasing error can be quantified as the ratio of the spacing of these shadow map and image sample locations. (Right) The sample spacing is related to the derivatives of the function that maps a point $t \in [0,1]$ in the shadow map to $\mathbf{p}_l$ on the light's image plane $\boldsymbol{\pi}_l$, projects $\mathbf{p}_l$ through the light to $\mathbf{p}$ on a planar surface $\boldsymbol{\pi}$, and then through the eye to a point $\mathbf{p}_e$ on the eye's image plane $\boldsymbol{\pi}_e$.*

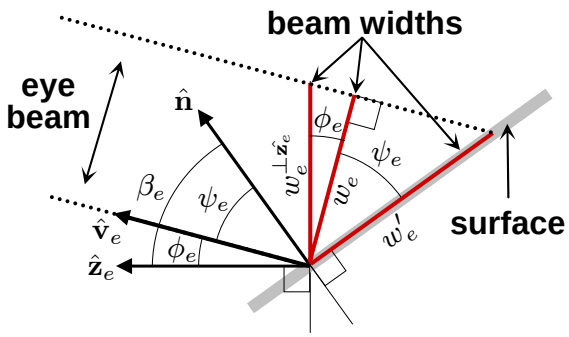

Figure 3.2: ***Computing the projected eye beam width on a surface.*** *w_e is the actual width of the beam, w'_e is the width of the projection onto the surface, and $w^{\perp \hat{\mathbf{z}}_e}$ is the width of beam measured perpendicular to $\hat{\mathbf{z}}_e$.*

3.1 Quantifying aliasing error

Aliasing error is caused by a mismatch between the sample locations used to render the shadow map from the viewpoint of the light and the image from the viewpoint of the eye. Ideally, the eye samples would correspond exactly with the light samples, as with raytracing or the irregular z-buffer. Shadow map warping methods instead seek to match sampling rates. For our derivations we choose to work with the spacing between samples since it is geometrically more intuitive. Figure 3.1 shows the sample spacing corresponding to eye and light samples at various locations in a simple 2D scene. Aliasing error occurs when the light sample spacing is greater than the eye sample spacing. We define $j \in [0,1]$ and $t \in [0,1]$ as normalized image and shadow map coordinates, respectively. We choose j and t as coordinates due to the fact that we are essentially looking at the side view of a 3D frustum. We will also use the coordinates i and s later when we look at the equations for 3D. The size of a pixel in the image is $1/r_j$ where r_j is the image resolution. The size of a texel in the shadow map is likewise $1/r_t$. The aliasing error can be quantified as:

$$m = \frac{r_j}{r_t}\frac{\mathrm{d}j}{\mathrm{d}t}. \tag{3.1}$$

Aliasing occurs when $m > 1$.

(i, j)	normalized eye image plane coordinates
(s, t)	normalized shadow map coordinates
(u, v)	normalized light image plane coordinates
$r_i \times r_j$	image resolution
$r_s \times r_t$	shadow map resolution
θ	frustum field of view
ϕ	angle between beam and image plane normal
ψ	angle between beam and surface normal
β	angle between image plane and surface normals
F, G	shadow map parameterization and its inverse
δ	spacing distribution function
$\tilde{\delta}_e$	perspective factor
d_l, d_e	distance from light and eye to a point in the scene
n_l, n_e	distance from light and eye to respective image plane
$\hat{\mathbf{n}}$	surface normal
L, E	lines through the light and eye
$\boldsymbol{\pi}_l, \boldsymbol{\pi}_e$	light and eye image planes
$\boldsymbol{\pi}$	a planar surface in the scene
$\mathbf{l}, \mathbf{e}$	light and eye positions
$\mathbf{p}_{l0}, \mathbf{p}_{e0}$	origins of coordinate systems on the light and eye image planes
$\mathbf{p}_l, \mathbf{p}_e$	points on the light and eye image planes
$(\Delta v)_l, (\Delta v)_e$	sample spacing in v on light image plane
w_l, w_e	widths of light and eye beams
w'_l, w'_e	widths of light and eye beams projected on a surface
$w_e^{\perp \hat{\mathbf{z}}_e}$	width of the eye beam measured perpendicular to $\hat{\mathbf{z}}_e$
W_l, W_e	widths of the light and eye image planes
$\hat{\mathbf{x}}_l, \hat{\mathbf{y}}_l, \hat{\mathbf{z}}_l$	light space coordinate vectors
$\hat{\mathbf{x}}_e, \hat{\mathbf{y}}_e, \hat{\mathbf{z}}_e$	eye space coordinate vectors

Table 3.1: ***Symbols used in this chapter.***

3.2 Deriving the aliasing error equation

To derive $\mathrm{d}j/\mathrm{d}t$ in Equation 3.1 we compute the function $j(t)$ from Figure 3.1 by tracing a sample from the shadow map to its corresponding location in the image. We begin by introducing our notation. A point $\mathbf{p}$ and a vector $\vec{\mathbf{v}}$ are expressed as column vectors in affine coordinates with the last entry equal to 1 for a point and 0 for a vector. Simpler formulas might be obtained by using full homogeneous coordinates, but we are interested in an intuitive definition of aliasing in terms of distances and angles, which are easier to compute with affine coordinates. $\hat{\mathbf{v}}$ is a normalized vector. A plane (or line in 2D) $\boldsymbol{\pi}$ is a row vector $(\hat{\mathbf{n}}^\top, -D)$,

where D is the distance to the plane (line) from the origin along the normal $\hat{\mathbf{n}}$. The dot product $\boldsymbol{\pi}\mathbf{p}$ gives the signed distance of $\mathbf{p}$ from the plane and $\boldsymbol{\pi}\hat{\mathbf{v}}$ gives $\hat{\mathbf{n}} \cdot \hat{\mathbf{v}} = \cos\theta$, where θ is the angle between $\hat{\mathbf{n}}$ and $\hat{\mathbf{v}}$.

A function G, the inverse of the shadow map parameterization F, maps a point t in the shadow map to the normalized light plane coordinate $v \in [0, 1]$. The point $\mathbf{p}_l$ on the light image plane $\boldsymbol{\pi}_l$ corresponding to v is given by:

$$\mathbf{p}_l = \mathbf{p}_{l0} + vW_l\hat{\mathbf{y}}_l \tag{3.2}$$

where W_l is the width of the portion of $\boldsymbol{\pi}_l$ covered by the shadow map. Projecting $\mathbf{p}_l$ onto a planar surface $\boldsymbol{\pi}$ along the line L through the light position $\mathbf{l}$ yields:

$$\mathbf{p} = \mathbf{p}_l - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}(\mathbf{p}_l - \mathbf{l})}(\mathbf{p}_l - \mathbf{l}). \tag{3.3}$$

This point is then projected onto the eye image plane $\boldsymbol{\pi}_e$ along the line E through the eye position $\mathbf{e}$:

$$\mathbf{p}_e = \mathbf{p} - \frac{\boldsymbol{\pi}_e\mathbf{p}}{\boldsymbol{\pi}_e(\mathbf{e} - \mathbf{p})}(\mathbf{e} - \mathbf{p}). \tag{3.4}$$

The eye image plane is parameterized by j in the same way as the light image plane is parameterized by v. The j coordinate can be computed from $\mathbf{p}_e$ as:

$$j = \frac{\hat{\mathbf{y}}_e \cdot (\mathbf{p}_e - \mathbf{p}_{e0})}{W_e} = \frac{\hat{\mathbf{y}}_e^\top (\mathbf{p}_e - \mathbf{p}_{e0})}{W_e} \tag{3.5}$$

To compute $\mathrm{d}j/\mathrm{d}t$ we use the chain rule:

$$\frac{\mathrm{d}j}{\mathrm{d}t} = \frac{\partial j}{\partial \mathbf{p}_e}\ \frac{\partial \mathbf{p}_e}{\partial \mathbf{p}}\ \frac{\partial \mathbf{p}}{\partial \mathbf{p}_l}\ \frac{\partial \mathbf{p}_l}{\partial v}\ \frac{\mathrm{d}v}{\mathrm{d}t} \tag{3.6}$$

$$\frac{\partial j}{\partial \mathbf{p}_e} = \frac{\hat{\mathbf{y}}_e^{\top}}{W_e} \tag{3.7}$$

$$\frac{\partial \mathbf{p}_e}{\partial \mathbf{p}} = \mathbf{I} + \frac{\boldsymbol{\pi}_e \mathbf{p}}{\boldsymbol{\pi}_e(\mathbf{e}-\mathbf{p})}\mathbf{I} - \frac{\boldsymbol{\pi}_e \mathbf{e}}{(\boldsymbol{\pi}_e(\mathbf{c_e}-\mathbf{p}))^2}\left((\mathbf{e}-\mathbf{p})\boldsymbol{\pi}_e\right) \tag{3.8}$$

$$\frac{\partial \mathbf{p}}{\partial \mathbf{p}_l} = \mathbf{I} - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}\mathbf{I} + \frac{\boldsymbol{\pi}\mathbf{l}}{(\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l}))^2}\left((\mathbf{p}_l-\mathbf{l})\boldsymbol{\pi}\right) \tag{3.9}$$

$$\frac{\partial \mathbf{p}_l}{\partial v} = W_l \hat{\mathbf{y}}_l \tag{3.10}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}G}{\mathrm{d}t}. \tag{3.11}$$

Equations 3.7 – 3.10 are expressions for Jacobian matrices. $\mathbf{I}$ is the identity matrix. The derivative $\mathrm{d}j/\mathrm{d}t$ can be reduced to a simpler form. We first multiply together Equations 3.9–3.11:

$$\begin{aligned}\frac{\partial \mathbf{p}}{\partial t} &= \frac{\partial \mathbf{p}}{\partial \mathbf{p}_l}\ \frac{\partial \mathbf{p}_l}{\partial v}\ \frac{\partial v}{\partial t} \\ &= W_l \frac{\mathrm{d}G}{\mathrm{d}t}\frac{\boldsymbol{\pi}\mathbf{l}}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}\left(\hat{\mathbf{y}}_l - \frac{\boldsymbol{\pi}\hat{\mathbf{y}}_l}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}(\mathbf{p}_l-\mathbf{l})\right).\end{aligned} \tag{3.12}$$

The $\boldsymbol{\pi}\mathbf{l}$ and $\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})$ terms are proportional to d_l and n_l, respectively, so their ratio can be replaced with d_l/n_l. The angle between $\hat{\mathbf{z}}_l$ and $\hat{\mathbf{n}}$ is β_l. Therefore $\boldsymbol{\pi}\hat{\mathbf{y}}_l = -\sin\beta_l$. We substitute $(\mathbf{p}_l-\mathbf{l}) = ||\mathbf{p}_l-\mathbf{l}||\hat{\mathbf{v}}_l$, where $-\hat{\mathbf{v}}$ is the light direction vector. The $||\mathbf{p}_l-\mathbf{l}||$ terms cancel leaving only $\hat{\mathbf{v}}$. We then replace $\boldsymbol{\pi}\hat{\mathbf{v}}$ with $-\cos\psi_l$:

$$\frac{\partial \mathbf{p}}{\partial t} = W_l \frac{\mathrm{d}G}{\mathrm{d}t}\frac{d_l}{n_l}\left(\hat{\mathbf{y}}_l - \frac{\sin\beta_l}{\cos\psi_l}\hat{\mathbf{v}}_l)\right). \tag{3.13}$$

We then expand $\hat{\mathbf{y}}_l$ an $\hat{\mathbf{v}}_l$ in terms of $\hat{\mathbf{n}}$ and $\hat{\mathbf{b}}$:

$$\hat{\mathbf{y}}_l = \cos\beta_l\hat{\mathbf{b}} - \sin\beta_l\hat{\mathbf{n}} \tag{3.14}$$

$$\hat{\mathbf{v}}_l = -\sin\psi_l\hat{\mathbf{b}} - \cos\psi_l\hat{\mathbf{n}}. \tag{3.15}$$

Substituting these equations into Equation 3.13 and utilizing the fact that $\psi_l - \beta_l = \phi_l$ yields:

$$\begin{aligned}\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} &= W_l \frac{\mathrm{d}G}{\mathrm{d}t} \frac{d_l}{n_l} \frac{(\cos\psi_l \cos\beta_l + \sin\psi_l \sin\beta_l)}{\cos\psi_l} \hat{\mathbf{b}} \\ &= W_l \frac{\mathrm{d}G}{\mathrm{d}t} \frac{d_l}{n_l} \frac{\cos(\psi_l - \beta_l)}{\cos\psi_l} \hat{\mathbf{b}} \\ &= W_l \frac{\mathrm{d}G}{\mathrm{d}t} \frac{d_l}{n_l} \frac{\cos\phi_l}{\cos\psi_l} \hat{\mathbf{b}}. \end{aligned} \tag{3.16}$$

Now we multiply together Equations 3.7, 3.8, and 3.16 and substitute $\boldsymbol{\pi}_e \mathbf{e} = n_e$, $\boldsymbol{\pi}_e(\mathbf{e} - \mathbf{p}) = d_e$, and $(\mathbf{e} - \mathbf{p}) = ||\mathbf{e} - \mathbf{p}|| \hat{\mathbf{v}}_e$:

$$\frac{\mathrm{d}j}{\mathrm{d}t} = \frac{W_l}{W_e} \frac{n_e}{n_l} \frac{d_l}{d_e} \left(\hat{\mathbf{y}}_e^\top \hat{\mathbf{b}} - \frac{(\hat{\mathbf{y}}_e^\top \hat{\mathbf{v}}_e)(\boldsymbol{\pi}_e \hat{\mathbf{b}})}{\boldsymbol{\pi}_e \hat{\mathbf{v}}_e} \right). \tag{3.17}$$

Substituting $\hat{\mathbf{y}}_e^\top \hat{\mathbf{b}} = \cos\beta_e$, $\hat{\mathbf{y}}_e^\top \hat{\mathbf{v}}_e = \sin\phi_e$, $\boldsymbol{\pi}_e \hat{\mathbf{b}} = -\sin\beta_e$, and $\boldsymbol{\pi}_e \hat{\mathbf{v}}_e = \cos\phi_e$ and utilizing the fact that $\beta_e - \phi_e = \psi_e$ yields the simplified version of $\mathrm{d}j/\mathrm{d}t$:

$$\begin{aligned}\frac{\mathrm{d}j}{\mathrm{d}t} &= \frac{W_l}{W_e} \frac{n_e}{n_l} \frac{d_l}{d_e} \frac{(\cos\phi_e \cos\beta_e + \sin\phi_e \sin\beta_e)}{\cos\phi_e} \\ &= \frac{W_l}{W_e} \frac{n_e}{n_l} \frac{d_l}{d_e} \frac{\cos(\beta_e - \phi_e)}{\cos\phi_e} \\ &= \frac{W_l}{W_e} \frac{n_e}{n_l} \frac{d_l}{d_e} \frac{\cos\psi_e}{\cos\phi_e}. \end{aligned} \tag{3.18}$$

Plugging $\mathrm{d}j/\mathrm{d}t$ into Equation 3.1 yields the final expression for aliasing error:

$$m = \frac{r_j}{r_t} \frac{\mathrm{d}G}{\mathrm{d}t} \frac{W_l}{W_e} \frac{n_e}{n_l} \frac{d_l}{d_e} \frac{\cos\phi_l}{\cos\phi_e} \frac{\cos\psi_e}{\cos\psi_l}. \tag{3.19}$$

3.2.1 Intuitive derivation.

Some basic intuition for the terms in Equation 3.19 can be obtained by considering an equivalent but less rigorous derivation of m from the relative sample spacing on a surface in the scene (see Figure 3.1). A beam from the light through a region on the light image plane corresponding to a single shadow map texel projects onto the surface with width w_l'. A beam from the eye through a pixel also projects onto the surface with width w_e'. The aliasing error

on the surface is given by the ratio of the projected beam widths:

$$m = \frac{w'_l}{w'_e}. \tag{3.20}$$

By similar triangles, the width of a pixel on $\boldsymbol{\pi}_e$ at a distance of n_e from the eye becomes $w_e^{\perp\hat{\mathbf{z}}_e}$ at a distance of d_e, where the beam intersects the surface:

$$w_e^{\perp\hat{\mathbf{z}}_e} = \frac{W_e}{r_j}\frac{d_e}{n_e}. \tag{3.21}$$

For a narrow beam, we can assume that the sides of the beam are essentially parallel. From Figure 3.2 we see that multiplying $w_e^{\perp\hat{\mathbf{z}}_e}$ by $\cos\phi_e$ gives the actual width of the beam w_e and dividing by $\cos\psi_e$ gives the width of its projection w'_e:

$$w_e = w_e^{\perp\hat{\mathbf{z}}_e}\cos\phi_e \tag{3.22}$$

$$w'_e = \frac{w_e}{\cos\psi_e} = \frac{W_e}{r_j}\frac{d_e}{n_e}\frac{\cos\phi_e}{\cos\psi_e}. \tag{3.23}$$

Similarly, a shadow map texel maps to a segment of width $(W_l/r_t)(\mathrm{d}G/\mathrm{d}t)$ on the light image plane producing a projected light beam width on the surface of:

$$w'_l = \frac{W_l}{r_t}\frac{\mathrm{d}G}{\mathrm{d}t}\frac{d_l}{n_l}\frac{\cos\phi_l}{\cos\psi_l}. \tag{3.24}$$

Plugging Equations 3.23 and 3.24 into Equation 3.20 yields Equation 3.19.

3.2.2 Directional lights.

As a point light at $\mathbf{l}$ moves away towards infinity along a direction $\hat{\mathbf{l}}$, it becomes a directional light. Equation 3.3 converges to:

$$\mathbf{p} = \mathbf{p}_l - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}\hat{\mathbf{l}}}\hat{\mathbf{l}}. \tag{3.25}$$

Equation 3.9 becomes:

$$\frac{\partial \mathbf{p}}{\partial \mathbf{p}_l} = \mathbf{I} - \frac{1}{\boldsymbol{\pi}\hat{\mathbf{l}}}\,\hat{\mathbf{l}}\boldsymbol{\pi}. \tag{3.26}$$

In Equation 3.19, the n_l/d_l term converges to 1 and the $\cos\phi_l$ term becomes constant.

The formulation for m in Equation 3.19 is similar to those used for aliasing error in previous work (Stamminger & Drettakis, 2002; Wimmer et al., 2004; Zhang et al., 2006b). However, our formulation is more general because it is valid for both point and directional lights and it takes into account the variation of eye and light beam widths as a function of ϕ_e and ϕ_l, respectively.

3.3 Factoring aliasing error

Our goal is to compute tight bounds on the aliasing error m, which we can then use to formulate a low-error shadow map parameterization F. For convenience we split m into two main parts:

$$m = \frac{(\Delta v)_l}{(\Delta v)_e} \tag{3.27}$$

$$(\Delta v)_l = \frac{1}{r_t}\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{r_t}\delta_l$$

$$(\Delta v)_e = \frac{1}{r_j}\frac{\mathrm{d}v}{\mathrm{d}j} = \frac{1}{r_j}\delta_e$$

$$\delta_l = \frac{\mathrm{d}G}{\mathrm{d}t} = \left(\frac{\mathrm{d}F}{\mathrm{d}v}\right)^{-1} \tag{3.28}$$

$$\delta_e = \tilde{\delta}_e \frac{\cos\psi_l}{\cos\psi_e} \tag{3.29}$$

$$\tilde{\delta}_e = \frac{W_e}{W_l}\frac{n_l}{n_e}\frac{d_e}{d_l}\frac{\cos\phi_e}{\cos\phi_l}. \tag{3.30}$$

Intuitively, $(\Delta v)_l$ and $(\Delta v)_e$ are the spacing in v between the corresponding light and eye samples (see Figure 3.1). This factorization is convenient because $(\Delta v)_e$ encapsulates all of the geometric terms fixed by the light, camera, and scene configuration, while $(\Delta v)_l$ encapsulates the two factors that can be manipulated to control aliasing – the parameterization, which determines the aliasing distribution, and the shadow map resolution that controls the overall

scale. For a point light, the orientation of the light image plane also affects the distribution of light samples and thus the aliasing error. We make a distinction between the light image plane and the near plane of the light frustum used to render the shadow map. The near plane is typically chosen based on other considerations, such as properly enclosing the scene geometry in the light frustum. Because one light image plane is related to another by a projective transformation that can be absorbed into the parameterization (see Appendix A), we can use a "standard" light image plane that is convenient for calculation and another image plane for actually rendering the shadow map. The spacing functions depend on a resolution factor and the *spacing distribution functions*, δ_l and δ_e, which are simply the derivatives $\mathrm{d}v/\mathrm{d}t$ and $\mathrm{d}v/\mathrm{d}j$, respectively. In the next chapter we will derive the parameterization F that produces a δ_l that approximates δ_e in order to minimize aliasing error. For this reason we have expressed δ_l in terms of F instead of its inverse.

Following Stamminger and Drettakis (2002), δ_e can be factored into two components – a *perspective factor*, $\tilde{\delta}_e$, and a *projection factor*, $\cos\psi_e/\cos\psi_l$. The perspective factor depends only on the position of the light and eye relative to a point. Over all points in the view frustum, the perspective factor is bounded and varies relatively smoothly. The projection factor, on the other hand, depends on the orientation of the surfaces in the scene, is potentially unbounded, and depending on the particular scene, can vary dramatically in complex ways. In order to obtain a simple parameterization amenable to real-time rendering without incurring the cost of a complex, scene-dependent analysis, many algorithms ignore the projective factor and address only perspective aliasing error $\tilde{m}$:

$$\tilde{m} = \frac{r_j}{r_t}\frac{\delta_l}{\tilde{\delta}_e} = \frac{w_e}{w_l}. \tag{3.31}$$

$\tilde{m}$ can be thought of as measuring the ratio of the widths of the light and eye beam at a point. Alternatively, $\tilde{m}$ can be thought of as the aliasing error on a surface with a normal that is located half-way between the eye and light directions $\hat{\mathbf{v}}_e$ and $-\hat{\mathbf{v}}_l$. For such a surface the projection factor is 1. We can use this fact to visualize perspective aliasing error in the image. Throughout this thesis we project the grid lines from the shadow map onto the scene in order to better visualize aliasing error. What we see, though, is a combination of projection

and perspective aliasing. By moving around a properly oriented planar "probe" surface in the scene we can visualize perspective aliasing directly for different regions of the view frustum.

3.4 Aliasing error in 3D

So far we have only analyzed aliasing error in 2D. In 3D we parameterize the eye's image plane, the light's image plane, and the shadow map by the 2D coordinates $\mathbf{i} = (i, j)^\top$, $\mathbf{u} = (u, v)^\top$, and $\mathbf{s} = (s, t)^\top$, respectively. Each coordinate is in the range $[0, 1]$. Equation 3.2 becomes:

$$\mathbf{p}_l = \mathbf{p}_{l0} + uW_{lx}\hat{\mathbf{x}}_l + vW_{ly}\hat{\mathbf{y}}_l. \tag{3.32}$$

Equations 3.3 and 3.4 remain the same. We replace W_e in Equation 3.5 with the direction-specific W_{ey} and add the equation to compute i:

$$i = \frac{\hat{\mathbf{x}}_e^\top(\mathbf{p}_e - \mathbf{p}_{e0})}{W_{ex}}. \tag{3.33}$$

The light image plane parameterization is now a 2D function $\mathbf{u} = \mathbf{G}(\mathbf{s})$. With these changes the projection of a shadow map texel in the image is now described by a 2×2 aliasing matrix $\mathbf{M}_a$:

$$\mathbf{M}_a = \begin{bmatrix} r_i & 0 \\ 0 & r_j \end{bmatrix} \frac{\partial \mathbf{i}}{\partial \mathbf{s}} \begin{bmatrix} \frac{1}{r_s} & 0 \\ 0 & \frac{1}{r_t} \end{bmatrix} \tag{3.34}$$

$$\frac{\partial \mathbf{i}}{\partial \mathbf{s}} = \begin{bmatrix} \frac{\partial i}{\partial s} & \frac{\partial i}{\partial t} \\ \frac{\partial j}{\partial s} & \frac{\partial j}{\partial t} \end{bmatrix} = \frac{\partial \mathbf{i}}{\partial \mathbf{p}_e} \frac{\partial \mathbf{p}_e}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{p}_l} \frac{\partial \mathbf{p}_l}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{s}} \tag{3.35}$$

$$\frac{\partial \mathbf{i}}{\partial \mathbf{p}_e} = \begin{bmatrix} \frac{\hat{\mathbf{x}}_e^\top}{W_{ex}} \\ \frac{\hat{\mathbf{y}}_e^\top}{W_{ey}} \end{bmatrix} \tag{3.36}$$

$$\frac{\partial \mathbf{p}_l}{\partial \mathbf{u}} = \begin{bmatrix} W_{lx}\hat{\mathbf{x}}_l & W_{ly}\hat{\mathbf{y}}_l \end{bmatrix} \tag{3.37}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{s}} = \frac{\partial \mathbf{G}}{\partial \mathbf{s}}. \tag{3.38}$$

To obtain a scalar measure of aliasing error it is necessary to define a metric h that is a function of the elements of $\mathbf{M}_a$. Some possibilities for h include a matrix norm, such as a p-norm or the Frobenius norm, or the determinant, which approximates the area of the projected shadow map texel in the image.

In 3D the spacing distribution functions $\boldsymbol{\delta}_l$ and $\boldsymbol{\delta}_e$ are 2×2 Jacobian matrices:

$$\boldsymbol{\delta}_l = \frac{\partial \mathbf{G}}{\partial \mathbf{s}} = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{u}} \right)^{-1} \tag{3.39}$$

$$\boldsymbol{\delta}_e = \left(\frac{\partial \mathbf{i}}{\partial \mathbf{u}} \right)^{-1}. \tag{3.40}$$

The factorization of $\boldsymbol{\delta}_e$ into perspective and projective factors is not as straightforward as in 2D. Several possibilities exist. Based on the intuition of the 2D perspective error, one approach might be to compute the differential cross sections $\mathbf{X}_e$ and $\mathbf{X}_l$ of the light and eye beam at a point and define perspective aliasing error as the ratio $\tilde{m} = h(\mathbf{X}_l)/h(\mathbf{X}_e)$. Another possibility would be to take $\tilde{m} = h(\mathbf{M}_a)$, where the normal of the planar surface used to compute $\mathbf{M}_a$ is oriented half-way between $\hat{\mathbf{v}}_e$ and $-\hat{\mathbf{v}}_l$. But unlike the 2D case, these two approaches are not guaranteed to be equivalent.

3.5 Summary

In this chapter we have shown how to compute aliasing error in both 2D and 3D for both point and directional light sources. In the next chapter we will use these equations to derive tight bounds on aliasing error in order to produce a parameterization that minimizes the error. In Chapter 5, these bounds are used to analyze various parameterizations.

CHAPTER 4

Minimizing aliasing error

In Chapter 3 we discussed how to quantify aliasing error. In this chapter we derive tight bounds on the aliasing error that can be used to derive a parameterization that minimizes the error. We first discuss how to compute an ideal parameterization in 2D that completely eliminates aliasing error. Unfortunately, this is only of theoretical interest due to its complexity. Departing instead from tight bounds on perspective aliasing error, we compute a spacing distribution δ_b that produces a near-optimal parameterization. This naturally leads to a partitioning of the shadow map into regions corresponding to the view frustum faces. We extend the analysis to 3D and analyze the parameterizations derived from δ_b. These parameterizations are also too complicated for practical use, but they provide a good baseline for evaluating simpler parameterizations in the next chapter.

4.1 The ideal parameterization

Aliasing error m can be controlled by choosing the appropriate sample spacing distribution δ_l on the light image plane, and using sufficient shadow map resolution r_t. Ideally, we would like to choose δ_l and r_t such that $m = 1$ for all visible points of a particular view. This condition ensures that aliasing is eliminated while using minimal shadow map resolution (recall that we are only handling resampling aliasing. More shadow map resolution may be needed to handle intial sampling aliasing). From Equation 3.27 we can see that $m = 1$ implies that $\delta_l / r_t = \delta_e / r_j$, which in turn implies:

$$\frac{1}{r_t}\frac{\mathrm{d}F}{\mathrm{d}v} = \frac{1}{r_j}\frac{\mathrm{d}j}{\mathrm{d}v}. \tag{4.1}$$

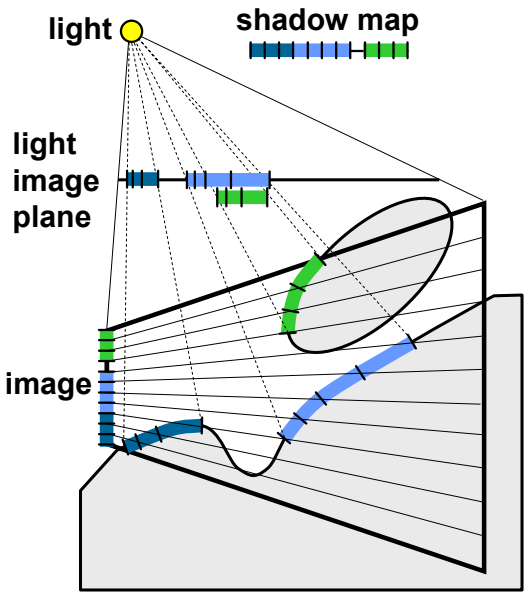

Figure 4.1: ***Ideal parameterization.*** *The partitions corresponding to continuously visible regions that face the light are color coded. The locations corresponding to eye samples are nonuniform on the light image plane. The ideal shadow map parameterization for each partition takes these samples back to their corresponding locations in the image.*

This equation is satisfied when $F(v) = j(v)$ and $r_t = r_j$. Thus the ideal parameterization is the function $j(v)$ that maps the location of a shadow map query on the light image plane back to the pixel in the image from which it originated. The ideal shadow map resolution is the same as that of image.

For a scene consisting of a single plane, $j(v)$ consists of a single projective transformation. The presence of multiple or non-convex surfaces requires partitioning to ensure that $j(v)$ is invertible. The partitions are defined by regions that are continuous, visible to the eye, and face towards the light (see Figure 4.1). Visible regions not facing the light are guaranteed to lie in shadow and need not be considered. The resolution of the shadow map needs be no larger than that of the image since each texel is mapped to a pixel in the image. In fact, the ideal parameterization gives us the same sample positions as used by the irregular z-buffer. The ideal parameterization is a generalization of the algorithm of Chong and Gortler (2004). They establish the one-to-one correspondence between shadow map texels and image pixels on a few important planes in the scene. For a scene consisting entirely of polygonal surfaces, the ideal parameterization would essentially be the result of applying this algorithm to the visible

portions of each polygon. The ideal parameterization is more general, however, because it can also be applied to higher-order surfaces.

Unfortunately, the ideal parameterization is completely impractical. The number of partitions required for a complex scene can be very large. Computing the regions corresponding to the partitions would be expensive. In addition, the warping functions for an arbitrary surface could be quite complex or may not be expressible in closed form.

4.2 Minimizing perspective aliasing error

The potentially large number of partitions required by the ideal parameterization arises from trying to match $\delta_l(v)/r_t$ to δ_e/r_j over all points that map to v on the light image. A simpler approach might be to match δ_l to the lower bound on δ_e over these points:

$$\delta_{e,\min}(v) = \min_{\mathbf{p}\in\mathcal{P}(v)}(\delta_e(\mathbf{p})). \tag{4.2}$$

$\mathcal{P}(v)$ is defined as the set of intersection points of the surfaces inside the view frustum with the line $\mathsf{L}(v)$ that passes through both the light and the point $\mathbf{p}_l(v)$ on the light image plane. Because $\delta_{e,\min}$ depends on scene geometry it can be arbitrarily complex. A lower bound on the perspective factor $\tilde{\delta}_e$ yields a simpler, scene-independent function:

$$\delta_b = \min_{\mathbf{p}\in\{\mathsf{L}(v)\cap\mathsf{V}\}}(\tilde{\delta}_e(\mathbf{p})). \tag{4.3}$$

V is the set of all points inside the view frustum. Using δ_b we can define a tight upper bound on the perspective aliasing error $\widetilde{M}$ expressed as a function of v:

$$\widetilde{M}\,(v) = \max_{\mathbf{p}\in\{\mathsf{L}(v)\cap\mathsf{V}\}}(\tilde{m}(\mathbf{p})) = \frac{r_j}{r_t}\frac{\delta_l(v)}{\delta_b(v)}. \tag{4.4}$$

For a given $\delta_l(v)$, the lowest resolution required to guarantee that there is no perspective aliasing in the view frustum, i.e. $\max_v(\widetilde{M}\,) = 1$, is $r_t = R_t r_j$, where R_t is given by:

$$R_t = \max_v\left(\frac{\delta_l(v)}{\delta_b(v)}\right). \tag{4.5}$$

We call R_t the *critical resolution factor*. R_t is smallest when δ_l is proportional to δ_b, in which case $R_t = \rho_b$, where ρ_b is the constant of proportionality. In general, we cannot choose δ_l to be exactly equal to δ_b because a constant scale factor is typically introduced when the parameterization which generates δ_l is normalized to the range $[0, 1]$.

Given an arbitrary spacing distribution function $\delta(v)$ the parameterization F that produces $\delta_l \sim \delta(v)$ can be computed as follows:

$$\left(\frac{\mathrm{d}F}{\mathrm{d}v}\right)^{-1} \sim \delta(v)$$

$$\frac{\mathrm{d}F}{\mathrm{d}v} = \frac{1}{\rho\delta(v)} \tag{4.6}$$

$$F(v) = \frac{1}{\rho}\int_0^v \frac{1}{\delta(\nu)}\mathrm{d}\nu. \tag{4.7}$$

Normalizing F to the range $[0, 1]$ gives the constant of proportionality.

$$\rho = \int_0^1 \frac{1}{\delta(v)}\mathrm{d}v. \tag{4.8}$$

This process will work with any valid $\delta(v)$ that is positive over the domain of integration.

4.2.1 Computing δ_b

The points inside the view frustum along a line $\mathsf{L}(v)$ can be parameterized as shown in Figure 4.2a:

$$\mathbf{p}(\mu) = \mathbf{l} + \mu\hat{\mathbf{v}}_l(v). \tag{4.9}$$

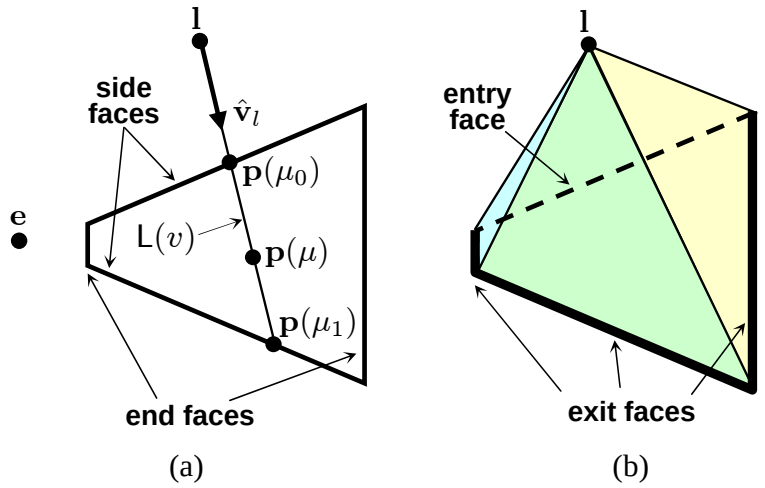

Figure 4.2: ***Face partitioning.*** *(a) Along a line segment* L *through the view frustum, a tight lower bound on the perspective factor* $\tilde{\delta}_e$ *attains its minimum value at the faces where* L *either enters or exits the frustum, depending on the position of the light. (b) Based on this observation we partition the shadow map, using a separate shadow map for the region corresponding to each face. For this light position, we partition using the exit faces. By using parameterizations that minimize the error along these faces, we can minimize the error over the whole frustum.*

From Equation 3.30 we compute $\tilde{\delta}_e(\mu)$:

$$\tilde{\delta}_e(\mu) = c\frac{d_e(\mu)}{d_l(\mu)}\cos\phi_e(\mu) \tag{4.10}$$

$$d_e(\mu) = -\hat{\mathbf{z}}_e \cdot (\mathbf{p}(\mu) - \mathbf{e}) = -\mu(\hat{\mathbf{z}}_e \cdot \hat{\mathbf{v}}_l) - \hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e}) \tag{4.11}$$

$$d_l(\mu) = -\hat{\mathbf{z}}_l \cdot (\mathbf{p}(\mu) - \mathbf{l}) = -\mu(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l) \tag{4.12}$$

$$\cos\phi_e(\mu) = \frac{d_e(\mu)}{||\mathbf{p}(\mu) - \mathbf{e}||}$$

$$c = \frac{W_e}{W_l}\frac{n_l}{n_e}\frac{1}{\cos\phi_l}.$$

The value of c is constant on L. Note that since δ_l is also constant along the line, $\max(\tilde{m})$ for any parameterization occurs at the same location as $\min(\tilde{\delta}_e)$. We want to find $\min_\mu(\tilde{\delta}_e(\mu))$ on the interval $\mu \in [\mu_0, \mu_1]$ inside the view frustum. To simplify the analysis we assume a symmetric view frustum and bound $\tilde{\delta}_e(\mu)$ from below by replacing the $\cos\phi_e$ term with its

smallest possible value, $\cos\theta$:

$$B(\mu) = c\frac{d_e(\mu)}{d_l(\mu)}\cos\theta \leq \tilde{\delta}_e(\mu). \tag{4.13}$$

$\tilde{\delta}_e$ can be at most $1/\cos\theta$ times larger than B, i.e. when $\cos\theta_e = 1$. For typical fields of view, B is a fairly tight lower bound. For example, with $\theta = 30°$, $1/\cos\theta$ is only about 1.15.

We take the derivative of $B(\mu)$ to determine where it reaches its minimum value:

$$\frac{\mathrm{d}B}{\mathrm{d}\mu} = (c\cos\theta)\ \frac{\hat{\mathbf{z}}_e \cdot (\mathbf{l}-\mathbf{e})(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l)}{(\mu(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l))^2}. \tag{4.14}$$

The first term and the denominator of the second term are strictly positive, and the $(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}})$ term in the numerator is strictly negative. Therefore, the sign of $\mathrm{d}B/\mathrm{d}\mu$ depends only on $\hat{\mathbf{z}}_e \cdot (\mathbf{l}-\mathbf{e})$. Since $\hat{\mathbf{z}}_e \cdot (\mathbf{l}-\mathbf{e})$ is constant for all μ, the location of the minimum B value, $\mu^B_{\min}$, must be at one of the boundaries of the interval:

$$\mu^B_{\min} = \underset{\mu\in[\mu_0,\mu_1]}{\operatorname{argmin}}\,(B(\mu)) = \begin{cases} \mu_0 & \hat{\mathbf{z}}_e \cdot (\mathbf{l}-\mathbf{e}) > 0, \\ \mu_1 & \hat{\mathbf{z}}_e \cdot (\mathbf{l}-\mathbf{e}) < 0. \end{cases} \tag{4.15}$$

When $\hat{\mathbf{z}}_e \cdot (\mathbf{l}-\mathbf{e}) = 0$, B is constant over the entire interval. For directional lights, the $(\mathbf{l}-\mathbf{e})$ term is replaced by the light direction $\hat{\mathbf{l}}$. Equation 4.15 shows that $\min(B)$ occurs on the faces where $\mathsf{L}(v)$ either enters or exits the view frustum, depending on the position of the light relative to the eye. When $\mu^B_{\min}$ is on a side face, $B(\mu^B_{\min})$ is the actual minimum for $\tilde{\delta}_e$. This can be seen from the fact that along a side face $\cos\phi_e = \cos\theta$ so $\tilde{\delta}_e = B$. Because B is never greater than $\tilde{\delta}_e$, this means that the actual minimum of $\tilde{\delta}_e$ over the interval cannot be smaller than the minimum B and must therefore be the same. Based on this observation we choose $\min(B)$ for δ_b:

$$\begin{aligned} \delta_b &= \min(B) = B(\mu^B_{\min}) \\ &= \frac{W_e}{W_l}\frac{n_l}{n_e}\frac{d_e}{d_l}\frac{\cos\theta}{\cos\phi_l}, \end{aligned} \tag{4.16}$$

where the d_e and d_l values are for points along the appropriate faces. At light positions where

$\mu^B_{\min}$ transitions from entry to exit faces or vice versa, $\min(B)$ is the same on both sets of faces. Thus the abrupt transitions in $\mu^B_{\min}$ as the light and camera move around do not cause temporal discontinuities in δ_b.

Choosing $\delta_l(v)/r_t = \rho_b\delta_b(v)/r_j$ we can guarantee that $\widetilde{M}(v) = 1$ for all v. Intuitively, this can be thought of as choosing δ_l and r_t such that no light beam is wider than the lower bound on the width of any eye beam that it intersects.

4.3 Computing a parameterization in 3D

One approach to computing a parameterization in 3D is to simply follow the same integration formulation that we used for 2D. We start from a $\boldsymbol{\delta}(u,v)$ that is a 2×2 matrix that describes the sample spacing distribution on the light image plane, invert $\boldsymbol{\delta}$, and integrate to get $\mathbf{F}(u,v)$. However, this approach has several complications. First, it is not clear how to compute a $\boldsymbol{\delta}$ that is lower bound on $\tilde{\boldsymbol{\delta}}_e$. The main problem is that $\boldsymbol{\delta}$ now contains information about orientation, whereas in 2D it was simply a scalar. $\boldsymbol{\delta}$ must also be invertible. Second, even if we come up with an invertible $\boldsymbol{\delta}$, there is no guarantee that we can integrate it to obtain $\mathbf{F}$. The rows of $\partial\mathbf{F}/\partial\mathbf{u}$ are the gradients of multivariable functions $s(u,v)$ and $t(u,v)$. Thus the mixed partials of its row entries must be equivalent, i.e.:

$$\frac{\partial^2 s}{\partial u\partial v} = \frac{\partial^2 s}{\partial v\partial u} \quad \text{and} \quad \frac{\partial^2 t}{\partial u\partial v} = \frac{\partial^2 t}{\partial v\partial u}. \tag{4.17}$$

If this property does not hold for a particular $\boldsymbol{\delta}^{-1}(u,v)$ then it cannot be the gradient of a function $\mathbf{F}(u,v)$. Finally, even if $\boldsymbol{\delta}^{-1}(u,v)$ is integrable, it is not guaranteed to be one-to-one over the entire domain covered by the shadow map.

Our approach is to treat the parameterizations s and t as essentially two instances of the simpler 2D problem. We choose scalar functions $\delta_{b,s}$ and $\delta_{b,t}$ derived from Equation 4.16 and integrate their multiplicative inverses w.r.t. u and v, respectively, to obtain s and t. We choose our coordinate system on each face as shown in Figure 4.3. For the W_l term we use the width of the parameterized region in the appropriate direction and for $\cos\phi_l$ we use $\cos\phi_{lx}$ or $\cos\phi_{ly}$, the angles between $\hat{\mathbf{z}}_l$ and the projection of $\hat{\mathbf{v}}_l$ in the XZ or YZ planes,

respectively. To obtain a scalar for W_e and $\cos\theta$, we assume that the fields of view are the same in both directions. If they are not, we can always choose conservative bounds, i.e. $W_e = \min(W_{ex}, W_{ey})$ and $\theta = \min(\theta_x, \theta_y)$. To simplify the parameterization we also choose the light image plane to coincide with the face, thus eliminating the d_l/n_l term. We will now discuss how we compute $\delta_{b,s}$ and $\delta_{b,t}$ for each type of face and then derive the corresponding parameterizations. If we assume a symmetric view frustum we can use the following equation to expand W_e when necessary:

$$W_e = 2n_e \tan\theta. \tag{4.18}$$

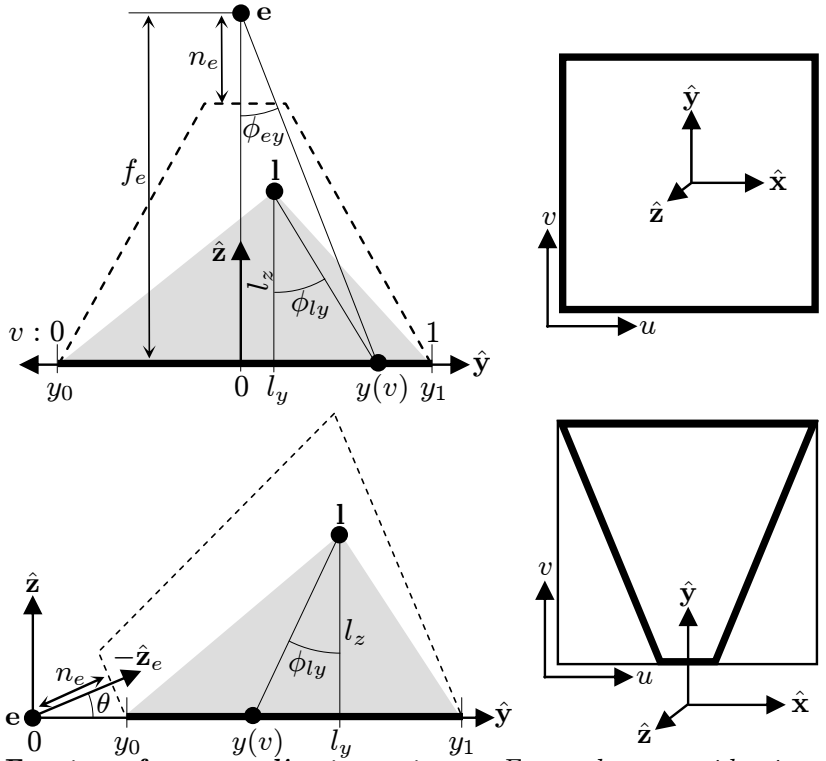

Figure 4.3: ***Frustum face coordinate systems.*** *For each row a side view appears on the left and a view from above the face on the right. (Top row) End face. The far face is shown here. The coordinate axes are aligned with the eye space coordinate axes. The origin is at the center of the face. (Bottom row) Side face. The y axis is aligned with the center line of the face with the z axis aligned with the face normal. The origin is at the eye.*

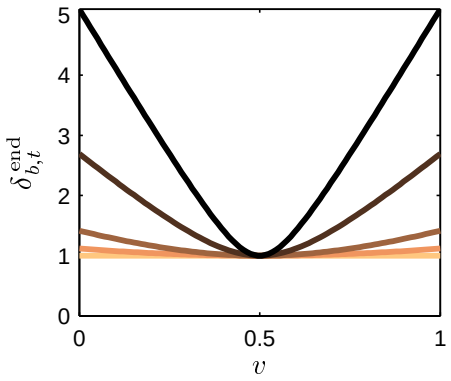

Figure 4.4: δ_b ***on an end face.*** *This graph shows* $\delta^{end}_{b,t}$ *for the light at* $l_y = 0$ *and* $l_z = \sigma(y_1 - y_0)$*. From dark to light,* $\sigma = 0.1, 0.2, 0.5, 1, \infty$*. The* $\cos\theta$ *term has been factored out.*

4.3.1 δ_b for end faces of the view frustum

We will first look at $\delta^{\text{end}}_{b,t}$. On the near face $d_e = n_e$, and on the far face $d_e = f_e$. The width of the v domain in y is the same as the width of the face $W_{ly} = (W_e d_e)/n_e$ so we can parameterize positions on the face as:

$$y(v) = (y_1 - y_0)v + y_0$$
$$y_0 = -\frac{W_e}{2}\frac{d_e}{n_e}, \quad y_1 = \frac{W_e}{2}\frac{d_e}{n_e}.$$

$\cos\phi_{ly}$ in terms of v is:

$$\cos\phi_{ly}(v) = \frac{l_z}{\sqrt{(y(v) - l_y)^2 + l_z^2}}. \tag{4.19}$$

Plugging these equations into Equation 4.16 we get:

$$\delta^{\text{end}}_{b,t}(v) = \frac{\cos\theta}{\cos\phi_{ly}(v)}. \tag{4.20}$$

$\delta^{\text{end}}_{b,s}$ is the same as $\delta^{\text{end}}_{b,t}$ except that quantities in y and v are exchanged for the corresponding quantities in x and u.

Figure 4.4 shows how $\delta^{\text{end}}_{b,t}$ on the far face changes with the light position. The $\cos\phi_{ly}$ term is responsible for the variation in the shape of the function. Close to the face, $\delta^{\text{end}}_{b,t}$ resembles an absolute value function with a rounded tip. As $l_z \to \infty$, $\cos\phi_{ly}$ converges to a constant

and the spacing becomes uniform. Moving the light in y simply translates the function along the v-axis.

4.3.2 δ_b for side faces of the view frustum

The end points of the side face are related to n_e and f_e:

$$y_0 = \frac{n_e}{\cos\theta}, \quad y_1 = \frac{f_e}{\cos\theta}.$$

On a side face, d_e is constant in x but increases with y. By similar triangles, the d_e/n_e term in δ_b can be expressed as:

$$\frac{d_e}{n_e} = \frac{y}{y_0}. \tag{4.21}$$

A side face does not cover the entire (u, v) domain. The extents of the face in x are:

$$x_0(y) = -\frac{W_e}{2}\frac{y}{y_0}, \quad x_1(y) = \frac{W_e}{2}\frac{y}{y_0}. \tag{4.22}$$

The width of the domain in x is the width of the face at y_1, $W_{lx} = (x_1(y_1) - x_0(y_1))$. The width in y, W_{ly}, is $(y_1 - y_0)$. We parameterize positions on the face as:

$$x(u) = (x_1(y_1) - x_0(y_1))u + x_0(y_1) \tag{4.23}$$

$$y(v) = (y_1 - y_0)v + y_0 \tag{4.24}$$

Putting all these equations together yields:

$$\delta^{\text{side}}_{b,s}(u, v) = \frac{y(v)}{y_1}\frac{\cos\theta}{\cos\phi_{lx}(u)} \tag{4.25}$$

$$= \frac{n_e + (f_e - n_e)v}{f_e}\frac{\cos\theta}{\cos\phi_{lx}(u)} \tag{4.26}$$

$$\delta^{\text{side}}_{b,t}(v) = \frac{W_e}{(y_1 - y_0)}\frac{y(v)}{y_0}\frac{\cos\theta}{\cos\phi_{ly}(v)} \tag{4.27}$$

$$= \left(v + \frac{n_e}{f_e - n_e}\right)\frac{2\tan\theta\cos^2\theta}{\cos\phi_{ly}(v)}. \tag{4.28}$$

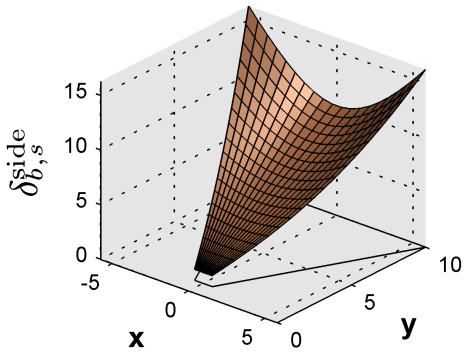

Figure 4.5: δ_b ***for*** s ***on a side face.*** *This graph shows* $\delta^{side}_{b,s}$ *with the light centered over the face.*

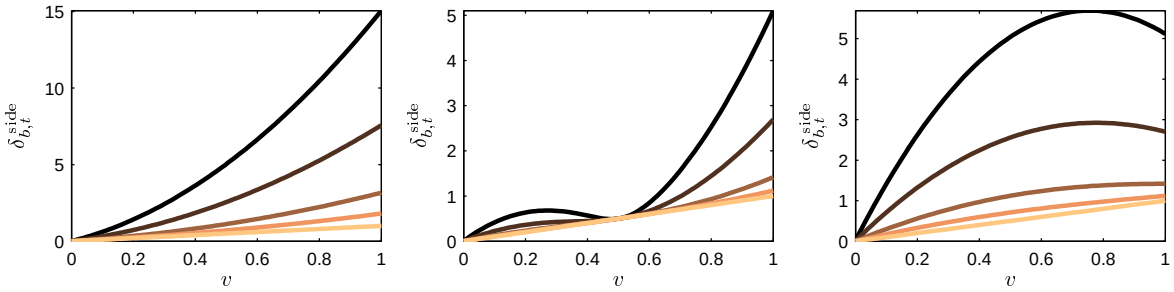

Figure 4.6: δ_b ***for*** t ***on a side face.*** *These graphs show* $\delta^{side}_{b,t}$ *for the light at* $l_y = \kappa(y_1 - y_0)$ *and* $l_z = \sigma(y_1 - y_0)$*. From left to right,* $\kappa = -0.5, 0.5, 1$*. From dark to light,* $\sigma = 0.1, 0.2, 0.5, 1, \infty$*. The frustum parameters are* $\theta = 45°$*,* $n_e = 1$*, and* $f_e = 1000$*.*

$\delta^{\text{side}}_{b,s}$ is a function of both u and v, so this is not strictly another instance of the 2D parameterization problem because the $\partial s/\partial v$ element of $\partial\mathbf{F}/\partial\mathbf{u}$ is nonzero. It is trivial to show, however, that when $s = \delta^{\text{side}}_{b,s}(u, v)$, the mixed partials are equal. The parameterizations $s(u, v)$ and $t(v)$ are invertible, and thus one-to-one, because we can always find v using $t^{-1}(v)$, plug this into $s(u, v)$, and invert to find u.

Figure 4.5 shows the shape of $\delta^{\text{side}}_{b,s}$ for a light centered over the face. Note that $\delta^{\text{side}}_{b,s}$ is essentially a linear ramp in y modulated by $\delta^{\text{end}}_{b,t}$ in x.

Figure 4.6 shows $\delta^{\text{side}}_{b,t}$ for the light at varying heights above the face. When the light is close to the face, the spacing function is an undulating curve with dense sampling near the light and the viewer. Translating the light in y produces a shifted and scaled version of the function. As the light moves away toward infinity, the spacing function converges to a linear ramp. Once again, it is the $\cos\phi_{ly}$ term that accounts for the variation in the shape of the

spacing function when the light is close to the face. When a directional light changes position, only the overall scale is affected.

4.3.3 δ_b parameterizations

If we let ζ_b be the indefinite integral of $1/\delta_b$, the parameterizations F_b and normalizing constants ρ_b for the three varieties of δ_b can be computed as follows:

$$F^{\text{end}}_{b,t}(v) = \frac{1}{\rho^{\text{end}}_{b,t}} \zeta^{\text{end}}_{b,t}\Big|_0^v \qquad \rho^{\text{end}}_{b,t} = \zeta^{\text{end}}_{b,t}\Big|_0^1 \tag{4.29}$$

$$F^{\text{side}}_{b,s}(u,v) = \frac{1}{\rho^{\text{side}}_{b,t}(v)} \zeta^{\text{side}}_{b,s}\Big|_{u_0(v)}^u \qquad \rho^{\text{side}}_{b,t}(v) = \zeta^{\text{side}}_{b,t}\Big|_{u_0(v)}^{u_1(v)} \tag{4.30}$$

$$F^{\text{side}}_{b,t}(v) = \frac{1}{\rho^{\text{side}}_{b,t}} \zeta^{\text{side}}_{b,t}\Big|_0^v \qquad \rho^{\text{side}}_{b,t} = \zeta^{\text{side}}_{b,t}\Big|_0^1 \tag{4.31}$$

$$\zeta^{\text{end}}_{b,t}(v) = C^{\text{end}}_{b,t}\ l_z \sinh^{-1}\left(\frac{y(v) - l_y}{l_z}\right) + C \tag{4.32}$$

$$\zeta^{\text{side}}_{b,s}(u,v) = C^{\text{side}}_{b,s}\ \frac{l_z}{y(v)} \sinh^{-1}\left(\frac{x(u) - l_x}{l_z}\right) + C \tag{4.33}$$

$$\begin{aligned}\zeta^{\text{side}}_{b,t}(v) = -C^{\text{side}}_{b,t}\ & \frac{l_z}{\sqrt{l_y^2 + l_z^2}} \\ & \log\left(\frac{2}{y(v)}\left(\frac{l_y(l_y - y(v)) + l_z^2}{l_z\sqrt{l_y^2 + l_z^2}} + \frac{1}{\cos\phi_{ly}(v)}\right)\right) \\ & + C.\end{aligned} \tag{4.34}$$

$$C^{\text{end}}_{b,t} = \frac{1}{(y_1 - y_0)\cos\theta} = \frac{1}{2d_e \tan\theta\cos\theta}$$

$$C^{\text{side}}_{b,s} = \frac{1}{(x_1(y_1) - x_0(y_1))\cos\theta} = \frac{1}{2f_e \tan\theta\cos\theta}$$

$$C^{\text{side}}_{b,t} = \frac{y_0}{W_e \cos\theta} = \frac{1}{2\tan\theta\cos^2\theta}$$

Note that for $F^{\text{side}}_{b,s}$ we integrate over $u \in [u_0(v), u_1(v)]$ corresponding to the part of the domain covered by the face. $u_0(v)$ and $u_1(v)$ can be found by solving $x(u) = x_0(y(v))$ and

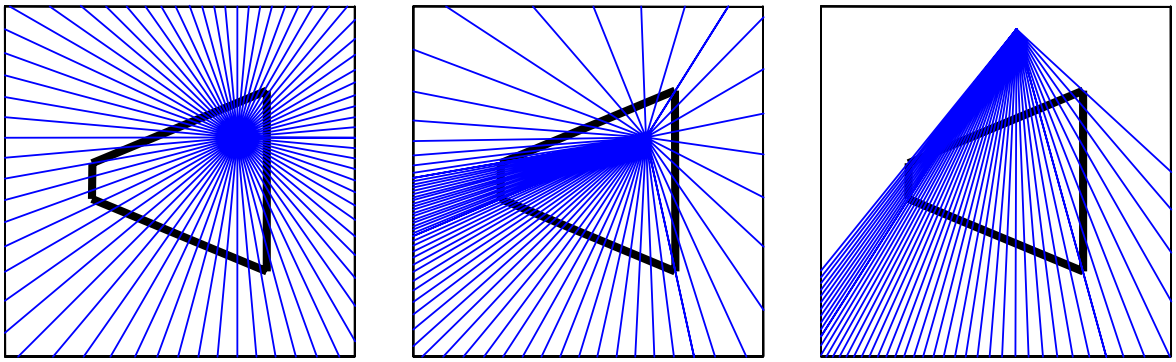

Figure 4.7: ***Visualization of light beams generated by*** δ_b ***parameterizations****. (Left) Uniform light beams from a point light source. (Middle and Right) Light beams distributed according to* δ_b *for a light inside and outside the view frustum. The width of the beams (measured perpendicular to the beam direction) matches that of the eye beams along the exit faces.*

$x(u) = x_1(y(v))$ for u:

$$u_0(v) = \frac{y_1 - y(v)}{2y_1}, \quad u_1(v) = \frac{y_1 + y(v)}{2y_1}. \tag{4.35}$$

Figure 4.7 shows what the light beams would look like using F_b to parameterize the faces. When the light is close to the face, less resolution is needed to cover the face because of the $\cos\phi_l$ factor.

4.4 Global error measures in 3D

We can compare parameterizations in 2D using the critical resolution factor R (Equation 4.5) as a error metric. R is the maximum perspective aliasing error over the view frustum that would occur if the image and shadow map resolution were the same. In 3D we combine the critical resolution factors for s and t:

$$S = R_s \times R_t. \tag{4.36}$$

We call S the *storage factor* because it represents the overall size in texels of a critical resolution shadow map relative to the size of the image in pixels. The definition of R_s^{side} must

be modified as follows:

$$R_s^{\text{side}} = \max_{(u,v)\in\mathcal{F}} \left(\frac{\delta_l(u,v)}{\delta_{b,s}^{\text{side}}(u,v)} \right), \tag{4.37}$$

where $\mathcal{F}$ is the set of points covered by the face. This is necessary because $\delta_{b,s}^{\text{side}}$ is a function of both u and v. It is also useful to define measures of maximum perspective error over the set of all possible light positions Ω:

$$\mathsf{R} = \max_{\Omega} R \tag{4.38}$$

$$\mathsf{S} = \max_{\Omega} S\,. \tag{4.39}$$

R and S can be evaluated simply by computing R and S for a light at infinity directly above the face. For this light position the $\cos\phi_l$ factor of δ_b is 1 over the entire face and R and S are maximized.

4.4.1 Balancing error between shadow map partitions and directions

In general, the position of the light relative to a face differs between faces. This causes S to also differ between faces. In order to minimize the maximum error over the entire frustum, the available shadow map resolution should be distributed between the faces in proportion to their maximum error. The fraction of the total number of texels T that should be allocated to each partition P_i is $\sigma_i T$:

$$\sigma_i = \frac{S_i}{\Sigma_{j=1}^{k} S_j}, \tag{4.40}$$

where k is the number of partitions and S_i is the storage factor of partition P_i. The texels allotted to a given partition can then be divided between the s and t directions of its shadow map in proportion to R_s and R_t in order to minimize the maximum error in either direction:

$$r_s = \sqrt{\frac{R_s}{R_t}\sigma T}, \qquad r_t = \sqrt{\frac{R_t}{R_s}\sigma T}. \tag{4.41}$$

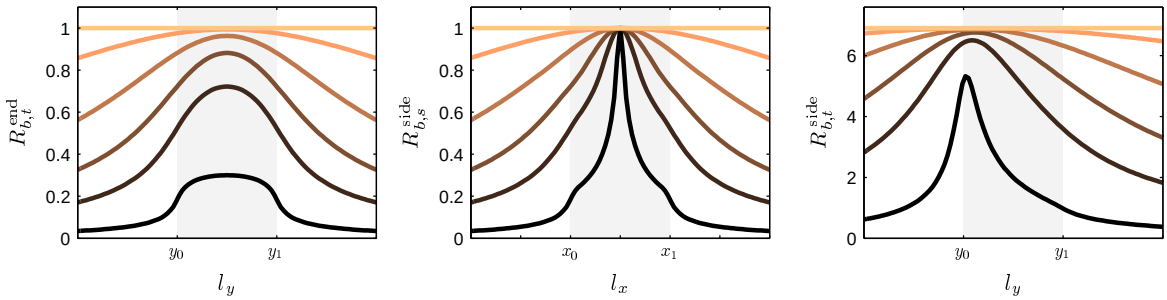

Figure 4.8: ***Critical resolution factor for*** δ_b ***parameterizations.*** *From left to right,* $R^{end}_{b,t}$, $R^{side}_{b,s}$, *and* $R^{side}_{b,t}$ *for the light at varying positions. The plots show* $l_z = \sigma(y_1 - y_0)$, *where from dark to light,* $\sigma = 0.1, 0.2, 0.5, 1, \infty$. *The grayed out region indicates light positions over the face. The frustum parameters are* $\theta = 45°$, $n_e = 1$, *and* $f_e = 1000$. *The* $\cos\theta$ *term has been factored out of* $R^{end}_{b,t}$ *and* $R^{side}_{b,s}$.

4.5 Maximum error of δ_b parameterizations

We can now compute the maximum error of each form of the δ_b parameterizations in terms of the critical resolution factor R and the storage factor S. As noted earlier, $R_b = \rho_b$. Figure 4.8 shows R_b for the three varieties of δ_b. In general, R_b is smaller when the light is close to the plane containing the face. It also falls off as the light moves to either side of the face. The maximum critical resolution factors R_b are:

$$\mathsf{R}^{\text{end}}_{b,t} = \frac{1}{\cos\theta} \tag{4.42}$$

$$\mathsf{R}^{\text{side}}_{b,s} = \frac{1}{\cos\theta} \tag{4.43}$$

$$\begin{aligned}\mathsf{R}^{\text{side}}_{b,t} &= \frac{y_0 \log(y_1/y_0)}{W_e \cos\theta} \\ &= \frac{\log(f_e/n_e)}{2\tan\theta\cos^2\theta},\end{aligned} \tag{4.44}$$

Multiplying together R for both the s and t directions (recall that $\delta^{\text{end}}_{b,s} = \delta^{\text{end}}_{b,t}$, so $\mathsf{R}^{\text{end}}_{b,s} = \mathsf{R}^{\text{end}}_{b,t}$) we see that S is largest for frusta with high far to near plane distance ratios and either very wide or narrow fields of view. Since the typical field of view falls somewhere in between, the (f_e/n_e) ratio is the dominating factor. The critical resolution factors are at most $O(1)$ for an end face and $O(\log(f_e/n_e))$ for a side face. Because these parameterizations are based on tight bounds of perspective aliasing, they are nearly optimal. These error bounds represent a baseline for the minimal maximum error.

In the next chapter we will show how the LogPSM parameterization has worst case error bounds that are identical to those of the near optimal δ_b parameterizations.

CHAPTER 5

Analysis of shadow map algorithms

In this chapter we analyze a number of shadow map algorithms using the error metrics developed in Chapter 4. This chapter consists of three major parts. First, we analyze parameterizations of the view frustum faces. Second, we extend this analysis to parameterizations of the entire view frustum. Third, we compare the various algorithms. We consider four different parameterizations:

- **Uniform.** Used by standard shadow maps.
- **Perspective.** Used by existing warping methods.
- **Logarithmic.** A parameterization suggested by several researchers as an improvement over a perspective parameterization.
- **Logarithmic+perspective.** The parameterization upon which LogPSMs are based.

We look at the overall behavior of these parameterizations as well as their maximum error. We show that the logarithmic+perspective parameterization produces worst-case maximum error that is on the same order as that produced by the near optimal parameterizations derived from tight bounds on perspective aliasing (Equations 4.42–4.44). We also examine the effects of applying z-partitioning to the faces, and attempt to address shearing artifacts that can arise with face partitioning. We also propose a new z-partitioning scheme.

We extend the analysis of face parameterization to the more complex case of parameterizing the entire frustum. Frustum parameterizations must converge to a uniform parameterization as the light direction approaches the view direction. We analyze the error over a range of light directions to formulate improved algorithms for choosing the warping parameters.

We conclude the chapter by comparing the various algorithms. We present experimental results and additional analysis on how to obtain low error shadow maps when a logarithmic perspective parameterization is not available.

5.1 Face parameterization

In Chapter 4 we saw that δ_b has 3 different forms depending on the face and the direction along the face we are considering (Equations 4.20, 4.26, and 4.28). The perspective aliasing error bounds on the frustum faces for an arbitrary parameterization as a function of position on the light image plane can be computed by plugging these equations into Equation 4.4:

$$\widetilde{M}_t^{\text{end}}(v) = \frac{r_j}{r_t}\frac{\delta_l(v)}{\delta_{b,t}^{\text{end}}(v)} = \widetilde{\mathrm{M}}_t^{\text{end}}(v)\frac{r_j}{r_t}\frac{\cos\phi_{ly}(v)}{\cos\theta} \qquad \widetilde{\mathrm{M}}_t^{\text{end}}(v) = \delta_l(v) \tag{5.1}$$

$$\widetilde{M}_s^{\text{side}}(u,v) = \frac{r_i}{r_s}\frac{\delta_l(u,v)}{\delta_{b,s}^{\text{side}}(u,v)} = \widetilde{\mathrm{M}}_s^{\text{side}}(u,v)\frac{r_i}{r_s}\frac{\cos\phi_{lx}(u)}{\cos\theta} \qquad \widetilde{\mathrm{M}}_s^{\text{side}}(u,v) = \frac{\delta_l(u,v)f_e}{n_e + (f_e - n_e)v} \tag{5.2}$$

$$\widetilde{M}_t^{\text{side}}(v) = \frac{r_j}{r_t}\frac{\delta_l(v)}{\delta_{b,t}^{\text{side}}(v)} = \widetilde{\mathrm{M}}_t^{\text{side}}(v)\frac{r_j}{r_t}\frac{\cos\phi_{ly}(v)}{2\tan\theta\cos^2\theta} \qquad \widetilde{\mathrm{M}}_t^{\text{side}}(v) = \frac{\delta_l(v)}{v + \frac{n_e}{f_e - n_e}} \tag{5.3}$$

We call $\widetilde{\mathrm{M}}$ the base distribution of the perspective aliasing error. For the perspective, logarithmic, and logarithmic perspective parameterizations we will drop the superscripts since these parameterizations are applied only to side faces. The resolution and θ terms affect the overall scale of the perspective aliasing error. For a directional light, the $\cos\phi_l$ term is constant and thus affects only the overall scale, but for a point light $\cos\phi_l$ modulates the base error distribution. Figure 5.1 shows the $\cos\phi_l$ term for the light at varying heights above a face. Separating out the base error distribution will simplify some of our analysis. It also makes it easier to see the dependence of the error on n_e and f_e because only $\widetilde{\mathrm{M}}$ contains these terms. We also define a base critical resolution R and base storage factor S in terms of $\widetilde{\mathrm{M}}$:

$$\mathrm{R} = \max_v \left(\widetilde{\mathrm{M}}(v)\right) \tag{5.4}$$

$$\mathrm{S} = \mathrm{R}_s\mathrm{R}_t. \tag{5.5}$$

A summary of the various error metrics used in this thesis are found in Table 5.1.

5.1.1 Uniform parameterization.

A shadow map parameterization consists of functions for the s and t coordinates in the shadow map as a function of u and v on the light image plane. The simplest shadow map

m	aliasing error
$\tilde{m}$	perspective aliasing error
$\widetilde{M}$	maximum $\tilde{m}$ along a light beam inside the view frustum
$\widetilde{\mathrm{M}}$	base distribution of perspective aliasing error ($\widetilde{M}$ without ϕ, θ, and resolution terms)
R	critical resolution factor (maximum $\widetilde{M}$ over view frustum in one direction)
R	base critical resolution (R without ϕ and θ terms)
R	maximum R over all light positions
S	storage factor (combined maximum error over view frustum in both directions)
S	base storage factor (S without ϕ and θ terms)
S	maximum S over all light positions

Table 5.1: ***Summary of various error metrics.***

F_b	near optimal, error bound parameterization
F_{un}	uniform parameterization
F_p	perspective parameterization
$F_{\log}$	logarithmic parameterization
F_{lp}	logarithmic perspective parameterization
n', f'	near and far plane distances of the warping frustum
$\bar{n}'$	warping parameter (normalized near plane distance of warping frustum)
η	reparameterized warping parameter
γ	angle between light and view directions

Table 5.2: ***Important symbols used in this chapter.***

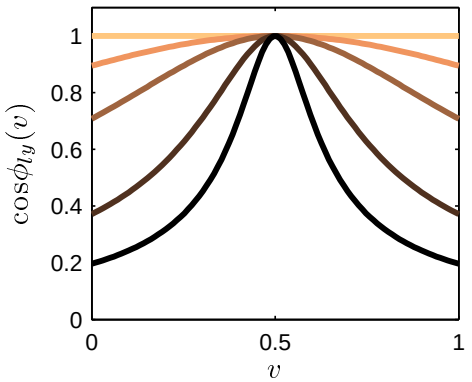

Figure 5.1: $\cos\phi_l$ ***factor.*** *The* $\cos\phi_l(v)$ *factor modulates the base error distribution* $\widetilde{\mathrm{M}}$*. For these graphs* $l_y = (y_0 + y_1)/2$ *and* $l_z = \sigma(y_1 - y_0)$*, where from dark to light, the plots are for* $\sigma = 0.1, 0.2, 0.5, 1, \infty$*.*

parameterization is the uniform parameterization:

$$\mathbf{F}_{un}(u,v) = \begin{bmatrix} s(u,v) \\ t(v) \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{5.6}$$

$$\delta_{un} = 1 \tag{5.7}$$

The critical resolution factors for the uniform parameterization can be computed from Equation 4.5. For example, on a side face we have:

$$R^{\text{side}}_{un,s} = \max\left(\widetilde{M}^{\text{side}}_{un,s}(u,v)\right) = \max\left(\widetilde{\mathrm{M}}^{\text{side}}_{un,s}(u,v)\frac{\cos\phi_{lx}(u)}{\cos\theta}\right) \tag{5.8}$$

$$\widetilde{\mathrm{M}}^{\text{side}}_{un,s} = \frac{f_e}{n_e + (f_e - n_e)v}. \tag{5.9}$$

Figure 5.2 shows R^{end}_{un} and R^{side}_{un} for various light positions. Note that each R_{un} reaches its maximum when the light position is over the face. That is because the $\cos\phi_{ly}$ factor reaches its maximum for points where $y(v) = l_y$. Comparing Figure 5.2 to Figure 4.8, we see that when the light is not directly over the face, the shapes of the graphs are nearly identical, but on the side face, the magnitudes of the plots differ significantly because $\widetilde{\mathrm{M}}^{\text{side}}_{un,s}$ and $\widetilde{\mathrm{M}}^{\text{side}}_{un,t}$ have a maximum value of f_e/n_e over the face.

5.1.1.1 Maximum error

In Section 4.4 we introduced the error metric R, which is the maximum perspective aliasing error over all points in the view frustum and over all light directions using a shadow map with the same resolution as the image. R_{un} can be computed by setting $\cos\phi_l = 1$ in R_{un}. This makes $\mathsf{R}^{\text{end}}_{un,t}$ constant. For $\mathsf{R}^{\text{side}}_{un,s}$ and $\mathsf{R}^{\text{side}}_{un,t}$, the $\widetilde{\mathrm{M}}^{\text{side}}_{un,s}$ and $\widetilde{\mathrm{M}}^{\text{side}}_{un,t}$ terms have their maximum

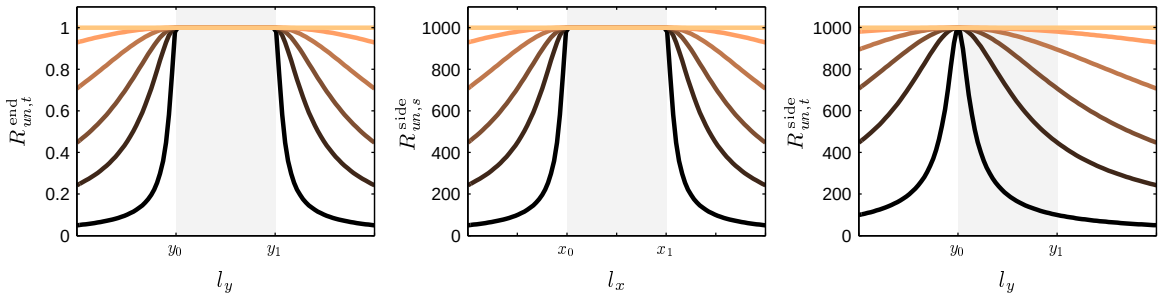

Figure 5.2: ***Critical resolution factor for uniform parameterization.*** *From left to right, $R^{end}_{un,t}$, $R^{side}_{un,s}$, and $R^{side}_{un,t}$ for various light positions. For all graphs $l_z = \sigma(y_1 - y_0)$, where from dark to light, the plots are for $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The grayed out region indicates light positions over the face. The frustum parameters are $\theta = 45°$, $n_e = 1$, and $f_e = 1000$. The $\cos\theta$ terms have been factored out of $R^{end}_{un,t}$ and $R^{side}_{un,s}$.*

at $v = 0$. Thus for maximum error with a uniform parameterization we get:

$$\mathsf{R}^{\text{end}}_{un,t} = \frac{1}{\cos\theta} \tag{5.10}$$

$$\mathsf{R}^{\text{side}}_{un,s} = \frac{f_e}{n_e \cos\theta} \tag{5.11}$$

$$\mathsf{R}^{\text{side}}_{un,t} = \frac{f_e - n_e}{2n_e \tan\theta \cos^2\theta} \tag{5.12}$$

$$\mathsf{S}^{\text{side}}_{un} = \mathsf{R}^{\text{side}}_{un,s}\mathsf{R}^{\text{side}}_{un,t} = (f_e/n_e)\frac{((f_e/n_e) - 1)}{2\tan\theta\cos^3\theta}. \tag{5.13}$$

The maximum critical resolution for the uniform parameterization is identical to that of F^{end}_b. For a side face, the maximum error in each direction is $O(f_e/n_e)$ while the error combining both directions is $O((f_e/n_e)^2)$.

5.1.2 Perspective parameterization.

The perspective parameterization places a trapezoidal warping frustum around the side face as shown in Figure 5.3. The parameterization maps the warping frustum to the square shadow map. The strength of the warping is controlled by n', the near plane distance of the warping frustum.

Figure 5.3 shows three different sets of coordinates that can be used for analyzing the perspective parameterization. The normalized light plane coordinates (v) are convenient because they are independent of the overall scale of the view frustum. $\bar{n}'$ is the normalized

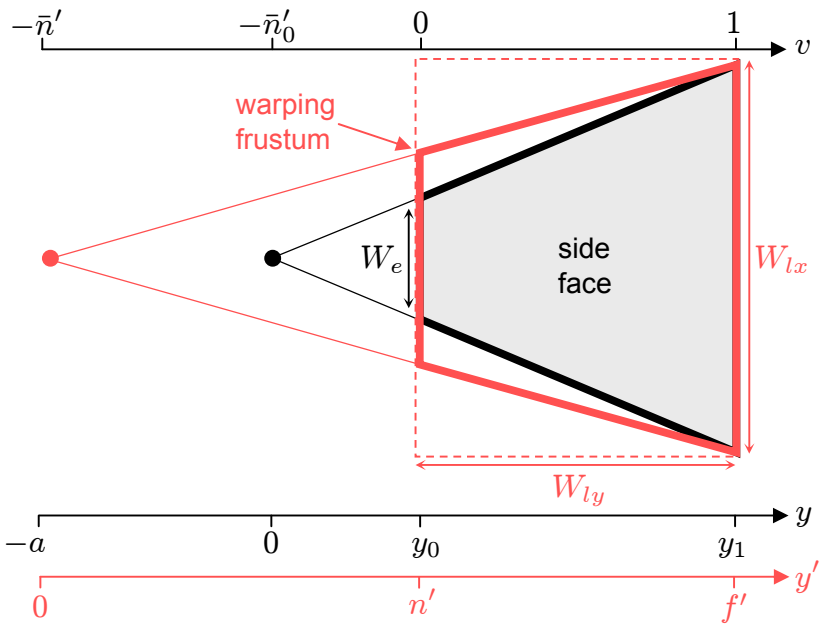

Figure 5.3: ***Perspective warping.*** *The perspective parameterization uses a second frustum placed around a side face to warp the shadow map. The near plane distance of warping frustum n', controls the strength of the warping. Also shown are quantities in several coordinate systems: normalized light image plane coordinates (v), face coordinates (y), and warping frustum coordinates (y').*

warping parameter. $\bar{n}'_0$ is the normalized position of the eye. The face coordinate system (y) introduced in Chapter 4 is the most natural for specifying positions relative to the face. The origin of the face coordinate system is at the eye. The warping frustum coordinates (y') are the same as face coordinates except that the origin is shifted back along y by a distance $a = n' - y_0$. Warping frustum coordinates can be expressed by quantities in normalized coordinates using the following simple relations:

$$y' = W_{ly}(v + \bar{n}') \tag{5.14}$$

$$n' = W_{ly}\bar{n}' \tag{5.15}$$

$$f' = W_{ly}(\bar{n}' + 1). \tag{5.16}$$

The dimensions of the rectangular region of the light image plane covered by the warping

frustum can be computed as:

$$W_{lx} = W_e \frac{y_1}{y_0} = 2 \tan\theta f_e \tag{5.17}$$

$$W_{ly} = f' - n' = (f_e - n_e)\cos\theta \tag{5.18}$$

The perspective parameterization is most conveniently specified in warping frustum coordinates using a 2D projective matrix:

$$\mathbf{F}_p = \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} x'_p/w'_p \\ y'_p/w'_p \end{bmatrix} \tag{5.19}$$

$$\begin{bmatrix} x'_p \\ y'_p \\ w'_p \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & 0 \\ 0 & p_2 & p_3 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{5.20}$$

$$\begin{aligned} p_0 &= \frac{f'}{W_{lx}} \\ p_1 &= \frac{1}{2} \\ p_2 &= \frac{f'}{f' - n'} \\ p_3 &= -p_2 n'. \end{aligned} \tag{5.21}$$

To obtain the spacing distribution functions generated by $\mathbf{F}_p$ we take the reciprocals of the derivatives of $F_{p,s}$ and $F_{p,t}$ w.r.t. x' and y' and scale by $1/W_{lx}$ and $1/W_{ly}$, respectively:

$$\delta_{p,s} = \frac{y'}{f'} = \frac{v + \bar{n}'}{\bar{n}' + 1} \tag{5.22}$$

$$\delta_{p,t} = \frac{(y')^2}{n'f'} = \frac{(v + \bar{n}')^2}{\bar{n}'(\bar{n}' + 1)}. \tag{5.23}$$

$\delta_{p,s}$ is constant in u and increases linearly in v. $\delta_{p,t}$ is quadratic in v. Because the light sample spacing increases quadratically and the eye sample spacing increases linearly, the perspective parameterization cannot remove perspective aliasing error. Interestingly, however, it can remove the combination of perspective *and* projection aliasing error for a single plane (Chong & Gortler, 2004).

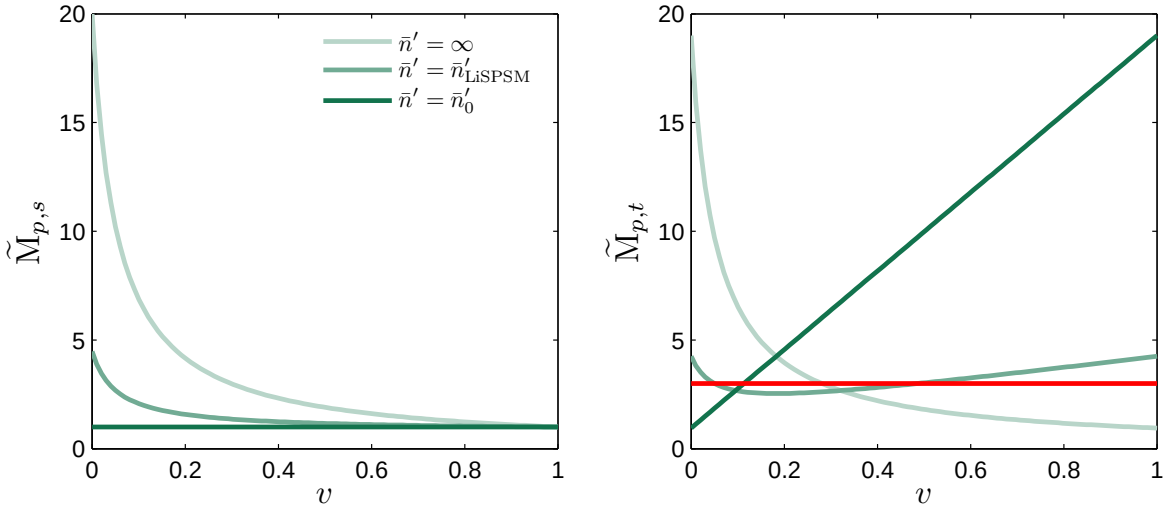

Figure 5.4: ***Base error distribution for perspective parameterization.*** *These plots show* $\widetilde{\mathrm{M}}_{p,s}$ *(left) and* $\widetilde{\mathrm{M}}_{p,t}$ *(right) for various values of* $\bar{n}'$*. For comparison, the base error distribution of a logarithmic perspective parameterization,* $\widetilde{\mathrm{M}}_{lp,t}$*, is shown in red on the right.* $\widetilde{\mathrm{M}}_{lp,s}$ *is the same as* $\widetilde{\mathrm{M}}_{p,s}$ *with* $\bar{n}' = \bar{n}'_0$*. Even for the very low* f_e/n_e *ratio shown in this graph, the difference in maximum error between the perspective and logarithmic perspective parameterizations is noticeable. (*$n_e = 1$ *and* $f_e = 20$*).*

The parts of δ_b^{side} (Equations 4.26 and 4.28) that depend on n_e and f_e can be rewritten in terms of $\bar{n}'_0$:

$$\frac{n_e + (f_e - n_e)v}{f_e} = \frac{v + \bar{n}'_0}{\bar{n}'_0 + 1} \tag{5.24}$$

$$v + \frac{n_e}{f_e - n_e} = v + \bar{n}'_0 \tag{5.25}$$

$$\bar{n}'_0 = \frac{n_e}{f_e - n_e} \tag{5.26}$$

This allows us to write the base error distributions for the perspective parameterization as:

$$\widetilde{\mathrm{M}}_{p,s} = \frac{(v + \bar{n}')(\bar{n}'_0 + 1)}{(v + \bar{n}'_0)(\bar{n}' + 1)} \tag{5.27}$$

$$\widetilde{\mathrm{M}}_{p,t} = \frac{(v + \bar{n}')^2}{\bar{n}'(\bar{n}' + 1)(v + \bar{n}'_0)}. \tag{5.28}$$

Figure 5.4 shows $\widetilde{\mathrm{M}}_{p,s}$ and $\widetilde{\mathrm{M}}_{p,t}$ for several values of $\bar{n}'$. At $\bar{n}' = \bar{n}'_0$ the error in s is uniform and minimal, while the error in t increases linearly. This parameter value corresponds to that used by PSMs for an overhead directional light. At $\bar{n}' = \bar{n}'_{\mathrm{LiSPSM}}$, the maximum error in t is

minimized, but the error in s goes up. $\bar{n}'_{\text{LiSPSM}}$ is the normalized warping parameter used by LiSPSMs (Wimmer et al., 2004):

$$\bar{n}'_{\text{LiSPSM}} = \frac{n_e + \sqrt{n_e f_e}}{f_e - n_e}. \tag{5.29}$$

As $\bar{n}' \to \infty$, the perspective parameterization converges to a uniform parameterization. The maximum error is high for both $\widetilde{\mathrm{M}}_{p,s}$ and $\widetilde{\mathrm{M}}_{p,t}$.

5.1.2.1 Parameterizing the warping parameter

The warping parameter $\bar{n}'$ is inconvenient to work with directly. The semi-infinite range of $\bar{n}' \in [\bar{n}'_0, \infty)$ is difficult to graph. Also the correlation between a change in $\bar{n}'$ and the change in error is nonlinear and not very intuitive. We have found it useful to parameterize $\bar{n}'$ directly in terms of the maximum error R.

Our first parameterization is based on $\mathrm{R}_{p,s} = \max(\widetilde{\mathrm{M}}_{p,s})$. As can be seen in Figure 5.4, $\mathrm{R}_{p,s} = \widetilde{\mathrm{M}}_{p,s}(0)$ for $\bar{n}' \in [\bar{n}'_0, \infty]$ because the maximum value of $\widetilde{\mathrm{M}}_{p,s}$ always occurs at $v = 0$ in this range. The value of $\mathrm{R}_{p,s}$ decreases from its maximum at $\bar{n}' = \infty$ (no warping) to its minimum at $\bar{n}' = 0$ (maximum warping). Based on this observation we define a parameter $\omega \in [0, 1]$ that can be thought of as the "amount of warping":

$$\begin{aligned} \omega(\bar{n}') &= \frac{\mathrm{R}_{p,s}(\bar{n}') - \mathrm{R}_{p,s}(\bar{n}'_0)}{\mathrm{R}_{p,s}(\infty) - \mathrm{R}_{p,s}(\bar{n}'_0)} \\ &= \frac{\bar{n}'_0 + 1}{(\bar{n}' + 1)} \end{aligned} \tag{5.30}$$

$$\bar{n}'(\omega) = (\bar{n}'_0 + 1)\left(\frac{1}{\omega} - 1\right), \tag{5.31}$$

where the argument to $\mathrm{R}_{p,s}$ is the warping parameter. As ω varies linearly from 0 to 1, $\bar{n}'$ varies (nonlinearly) from ∞ down to $\bar{n}'_0$, increasing the warping strength, and the value of $\mathrm{R}_{p,s}$ varies linearly from its maximum to minimum value. The width of the near plane of the warping frustum also varies linearly from W_{lx} down to W_e. Thus ω is also an intuitive control for the shape of the warping frustum.

Our second parameterization is based on $\mathrm{R}_{p,t}$. The behavior of $\mathrm{R}_{p,t}$ over the range of

warping parameters is slightly more involved than that of $\mathrm{R}_{p,s}$. The maximum error occurs at the far plane ($v = 1$) for $\bar{n}' \in [\bar{n}'_0, \bar{n}'_{\mathrm{LiSPSM}})$ and at the near plane ($v = 0$) for $\bar{n}' \in [\bar{n}'_{\mathrm{LiSPSM}}, \infty)$. In much the same way as we defined ω, we define a parameter $\eta \in [-1, 1]$, where $\eta = -1$ corresponds to maximum error at the near plane, which decreases linearly to its minimum over the range $\eta \in [-1, 0]$, and then increases linearly again to the same maximum on the far plane with $\eta \in [0, 1]$:

$$\begin{aligned}
\eta(\bar{n}') &= \begin{cases} \frac{\mathrm{R}_{p,t}(\bar{n}'_{\mathrm{LiSPSM}}) - \mathrm{R}_{p,t}(\bar{n}')}{\mathrm{R}_{p,t}(\infty) - \mathrm{R}_{p,t}(\bar{n}'_{\mathrm{LiSPSM}}} & \bar{n}' \geq \bar{n}'_{\mathrm{LiSPSM}} \\ \frac{\mathrm{R}_{p,t}(\bar{n}') - \mathrm{R}_{p,t}(\bar{n}'_{\mathrm{LiSPSM}})}{\mathrm{R}_{p,t}(\bar{n}'_0) - \mathrm{R}_{p,t}(\bar{n}'_{\mathrm{LiSPSM}})} & \bar{n}' < \bar{n}'_{\mathrm{LiSPSM}} \end{cases} \\
&= \begin{cases} \frac{1+\sqrt{\alpha}-\bar{n}'(\alpha-1)}{(\alpha-1)(\bar{n}'+1)} & \bar{n}' \geq \bar{n}'_{\mathrm{LiSPSM}} \\ \frac{\bar{n}'(1-\sqrt{\alpha})+1}{\bar{n}'(\alpha-\sqrt{\alpha})} & \bar{n}' < \bar{n}'_{\mathrm{LiSPSM}} \end{cases} \qquad (5.32)
\end{aligned}$$

$$\bar{n}'(\eta) = \begin{cases} \frac{1+\sqrt{\alpha}-\eta(\alpha-1)}{(\alpha-1)(\eta+1)} & -1 \leq \eta < 0 \\ \frac{1}{(\sqrt{\alpha}-1)(\eta\sqrt{\alpha}+1)} & 0 \leq \eta \leq 1 \end{cases} \qquad (5.33)$$

$$\alpha = \frac{f_e}{n_e} \qquad (5.34)$$

For an overhead directional light, $\eta = -1$ corresponds to a standard shadow map, $\eta = 0$ corresponds to the warping parameter of a LiSPSM, and $\eta = 1$ corresponds to the warping parameter of a PSM.

5.1.2.2 Behavior over the range of warping parameters

Figure 5.5 shows $\widetilde{\mathrm{M}}_{p,s}$ and $\widetilde{\mathrm{M}}_{p,t}$ over all valid values of v and $\bar{n}'$, with $\bar{n}'$ parameterized by η. $\widetilde{\mathrm{M}}_{p,s}$ decreases monotonically over v for all η. $\widetilde{\mathrm{M}}_{p,t}(v)$ has an internal minimum at $v = \bar{n}' - \bar{n}'_0$, which is shown on the graph. $\widetilde{\mathrm{M}}_{p,t}(v)$ increases monotonically when $\bar{n}' < \bar{n}'_0$ and decreases monotonically when $\bar{n}' > \bar{n}'_0 + 1$. The maxima of $\widetilde{\mathrm{M}}_{p,s}$ and $\widetilde{\mathrm{M}}_{p,t}$ always occur at one of the boundaries. For a point light, the base error distributions are modulated by the $\cos\phi_l$ term (see Figure 5.1), which can introduce error maxima that do not occur at the boundaries.

Figure 5.6 abstracts some of the information in Figure 5.5 and makes it easier to see the behavior of the maximum error over the range of warping parameter values. The figure shows

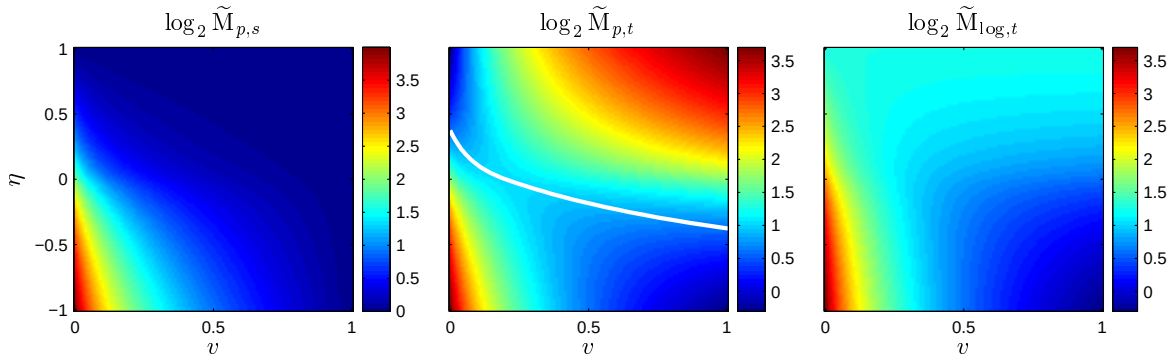

Figure 5.5: ***Error distribution for all warping parameter values.*** *These plots show* $\widetilde{\mathrm{M}}$ *for both perspective and logarithmic parameterizations over the entire range of warping parameter values. The white line in the plot for* $\widetilde{\mathrm{M}}_{p,t}$ *is the location of the minimum over* v*.* ($n_e = 1,\ f_e = 16$)

$\widetilde{\mathrm{M}}_{p,s}$ and $\widetilde{\mathrm{M}}_{p,t}$ for both $v = 0$ and $v = 1$. It also shows the minimum value of $\widetilde{\mathrm{M}}_{p,t}$. We plot the functions at these v values to capture the variation in error over all v. The maximum value of these functions is the base critical resolution factor R.

Figure 5.6 also shows the base storage factor S_p. It is interesting to note that there exists not just one warping parameter value for which S_p is minimal, but a range of values, i.e. $\eta \in [0, 1]$. This means that on a face, the warping parameter values for LiSPSMs and PSMs, which lie in this range, are equivalent in terms of storage factor, at least for a directional light, for which the $\cos\phi_l$ factor (not included in the graphs) is constant. Figure 5.7 shows that a range of optimal parameter values also exists for point lights. From Figure 5.6, however, we see that variation in error within the view frustum changes considerably within the equivalent range. Choosing the largest $\bar{n}'$ (smallest η) on the equivalent range insures that the distribution of error between the s and t directions is most balanced. For the smallest parameter in the range, $\bar{n}' = \bar{n}'_0$, almost all the error is concentrated in the t direction. Choosing the maximum parameter in the equivalent range also helps reduce the shearing artifacts described later in this section.

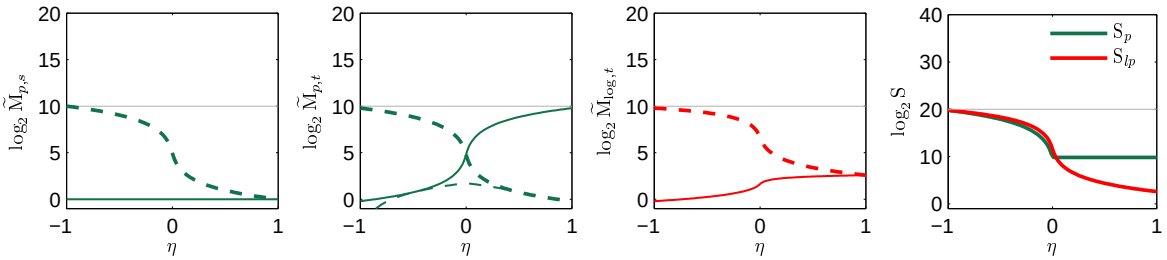

Figure 5.6: ***Maximum error over all warping parameter values.*** *These graphs plot the error at $v = 0$ (heavy dotted line), $v = 1$ (thin solid line), and for $\widetilde{\mathrm{M}}_{p,t}$ the minimum error over all v (thin dotted line). The maximum error of a uniform parameterization is also shown (gray line). S_p is constant and minimal for $\eta \in [0, 1]$. ($n_e = 1$ and $f_e = 1024$).*

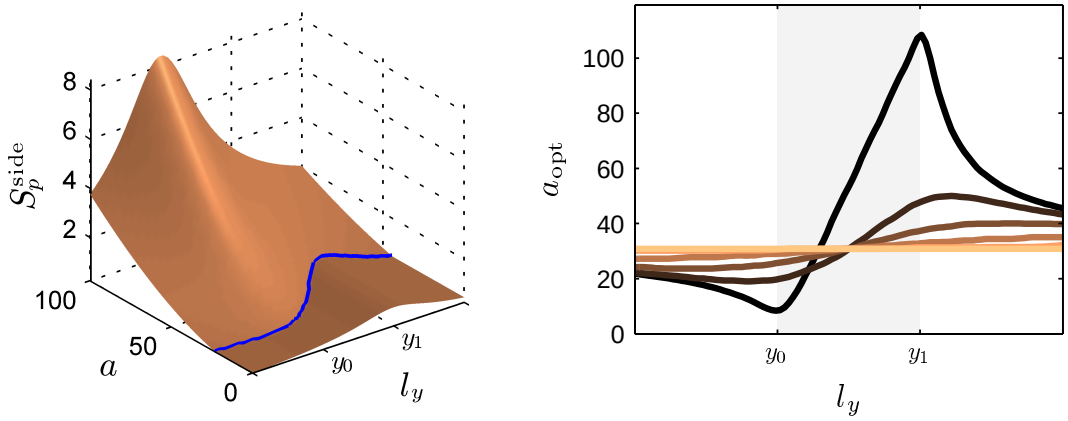

Figure 5.7: ***Optimal perspective warping parameter.*** (Left) *When the errors in s and t are combined, a range of warping parameter values (those below the curve) produce the same storage factor S_p^{side} for each light position ($l_z = 0.5(y_1 - y_0)$). The warping parameter is shown here in terms of $a = (\bar{n}' - \bar{n}'_0) \cdot (y_1 - y_0)$.* (Right) *The maximum parameter value in optimal range a_{opt} for $l_z = \sigma(y_1 - y_0)$, where from dark to light $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The frustum parameters are $\theta = 45°$, $n_e = 1$, and $f_e = 1000$.*

5.1.2.3 Maximum error

Figure 5.8 shows the optimal $R_{p,s}$, $R_{p,t}$, and S_p for varying light positions above the face. The optimal $R_{p,s}$ is computed as:

$$\min_{\bar{n}'} R_{p,s} = \min_{\bar{n}'} \max_{(u,v)\in\mathcal{F}} \left(\frac{\delta_{p,s}(u,v)}{\delta_{b,s}^{\mathrm{side}}(u,v)} \right). \tag{5.35}$$

The value of $R_{p,s}$ is that of $\widetilde{M}_{p,s}$ at $v = 0$. $R_{p,s}$ is minimized when $\bar{n}' = \bar{n}'_0$, which leaves only the $\cos\phi_{lx}/\cos\theta$ term. Geometrically, setting $\bar{n}' = \bar{n}'_0$ causes the face to be stretched out to

fill the entire shadow map, maximizing the use of the available resolution. With $\bar{n}' = \bar{n}'_0$, the $\widetilde{\mathrm{M}}_{p,s}$ term of $R_{p,s}$ becomes constant. As we saw in Figure 5.7, the optimal $\bar{n}'$ parameter for $R_{p,t}$ varies with the position of the light. For a directional light directly above the face, the optimal parameter value is $\bar{n}'_{\mathrm{LiSPSM}}$. The maximum occurs at either $v = 0$ or $v = 1$. Using these optimal parameters for s and t for an overhead directional light gives the optimal R_p:

$$\min_{\bar{n}'} \mathsf{R}_{p,s} = \frac{1}{\cos\theta} \tag{5.36}$$

$$\begin{aligned} \min_{\bar{n}'} \mathsf{R}_{p,t} &= \frac{(f_e - n_e)}{2\sqrt{n_e f_e}\tan\theta\cos^2\theta} \\ &\approx O(\sqrt{f_e/n_e}). \end{aligned} \tag{5.37}$$

If $\mathbf{F}_p$ is implemented with a projective matrix, the $\bar{n}'$ parameter can not be chosen independently for s and t. Rather $R_{p,s}$ and $R_{p,t}$ should be optimized simultaneously by optimizing the storage factor S_p. Using $\bar{n}' = \bar{n}'_{\mathrm{LiSPSM}}$ for an overhead directional light gives the optimal S_p:

$$\begin{aligned} \min_{\bar{n}'} \mathsf{S}_p &= \frac{(f_e - n_e)}{2n_e\tan\theta\cos^3\theta} \\ &= \frac{(f_e/n_e - 1)}{2\tan\theta\cos^3\theta} \\ &\approx O(f_e/n_e). \end{aligned} \tag{5.38}$$

The effect of optimal perspective warping is to remove the leading (f_e/n_e) from S_{un} on a side face (cf. Equation 5.13). Though the maximum error in s is the same as that of $F^{\mathrm{side}}_{b,s}$, when we consider both directions at the same time the $O(f_e/n_e)$ minimal maximum error of $\mathbf{F}^{\mathrm{side}}_p$ is still relatively high compared to the $O(\log(f_e/n_e))$ maximum error of $\mathbf{F}^{\mathrm{side}}_b$.

5.1.3 Logarithmic parameterization.

We have now shown that the uniform parameterization gives error for end faces that is on the same order as that of the near optimal δ_b parameterization. The perspective parameterization can do the same for the s direction on side faces. A logarithmic parameterization can give the same maximum error as the parameterization based on $\delta^{\mathrm{side}}_{b,t}$. The logarithmic

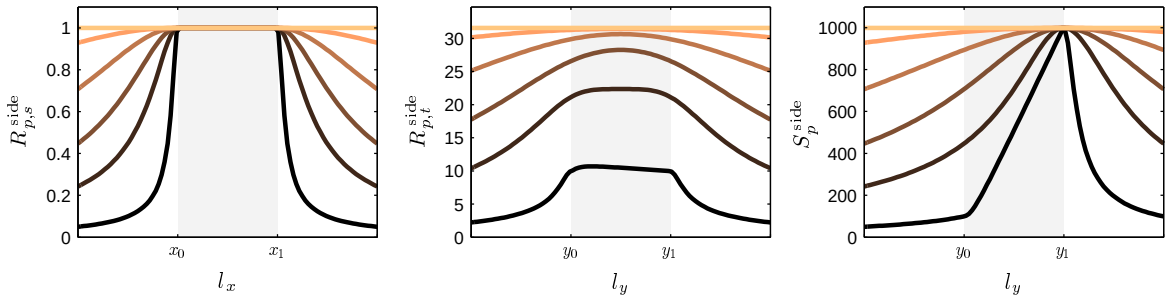

Figure 5.8: ***Optimal critical resolution factors for perspective parameterization.*** *From left to right $R^{side}_{p,s}$, $R^{side}_{p,t}$, and S^{side}_{p} for various light positions. When parameterizing with a perspective matrix, $\delta^{side}_{p,s}$ and $\delta^{side}_{p,t}$ are coupled. S^{side}_{p} combines the errors in both directions so that the optimal perspective parameter can be found considering both directions together. For all graphs $l_z = \sigma(y_1 - y_0)$, where from dark to light, the plots are for $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The grayed out region indicates light positions over the face. The frustum parameters are $\theta = 45°$, $n_e = 1$, and $f_e = 1000$. A $\cos\theta$ term have been factored out of $R^{side}_{p,s}$ and S^{side}_{p}.*

parameterization is given by:

$$F_{\log,t} = \frac{\log\left(y'/n'\right)}{\log\left(f'/n'\right)}. \tag{5.39}$$

This is a generalized version of a logarithmic parameterization with a free warping parameter n' similar to that of $\mathbf{F}_p$. The spacing distribution generated by $F_{\log,t}$ is:

$$\delta_{\log,t} = \log\left(f'/n'\right)\frac{y'}{f'-n'} = \log\left(\frac{\bar{n}'+1}{\bar{n}'}\right)(v+\bar{n}') \tag{5.40}$$

$\widetilde{\mathrm{M}}_{\log,t}$ can be computed just as for $\widetilde{\mathrm{M}}_{p,t}$:

$$\widetilde{\mathrm{M}}_{\log,t} = \log\left(\frac{\bar{n}'+1}{\bar{n}'}\right)\frac{v+\bar{n}'}{v+\bar{n}'_0}. \tag{5.41}$$

From Figure 5.5 we can see the maximum $\widetilde{\mathrm{M}}_{\log,t}$ occurs at $v = 0$ for all parameter values. At $\eta = 1$, $\max_v(\widetilde{\mathrm{M}}_{\log,t}(v))$ reaches its minimum and is constant over v.

5.1.3.1 Maximum error

Figure 5.9 shows $R_{\log,t}$. Because $\delta_{\log,t}$ is not the best match for the $\cos\phi_l$ term, $R_{\log,t}$ is not as low as $R^{\text{side}}_{b,t}$ when the light is close to the face, but the range of values is on the same order

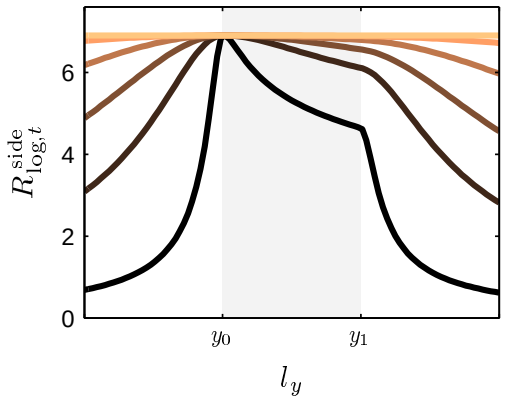

Figure 5.9: ***Critical resolution factor for logarithmic parameterization.*** *For all graphs $l_z = \sigma(y_1 - y_0)$, where from dark to light, the plots are for $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The grayed out region indicates light positions over the face. The frustum parameters are $\theta = 45°$, $n_e = 1$, and $f_e = 1000$.*

Parameterization	$\mathsf{R}_s^{\text{end}}$	$\mathsf{R}_s^{\text{side}}$	$\mathsf{R}_t^{\text{side}}$	S^{side}
Near optimal (F_b)	$O(1)$	$O(1)$	$O\left(\log\left(\frac{f_e}{n_e}\right)\right)$	$O\left(\log\left(\frac{f_e}{n_e}\right)\right)$
Uniform	$O(1)$	$O\left(\frac{f_e}{n_e}\right)$	$O\left(\frac{f_e}{n_e}\right)$	$O\left(\left(\frac{f_e}{n_e}\right)^2\right)$
Perspective	–	$O(1)$	$O\left(\sqrt{\frac{f_e}{n_e}}\right)$	$O\left(\frac{f_e}{n_e}\right)$
Logarithmic	–	–	$O\left(\log\left(\frac{f_e}{n_e}\right)\right)$	–

Table 5.3: ***Maximum error.*** *This table shows R and S, measures of the maximum error over all light directions, for several parameterizations, including the near optimal parameterization F_b based on δ_b. The perspective and logarithmic parameterizations have error that is on the same order as $F_{b,s}$ and $F_{b,t}$ for side faces, respectively. A uniform parameterization is sufficient for end faces.*

as that of $R^{\text{side}}_{b,t}$. With an overhead directional light and the optimal warping parameter value of $\bar{n}' = \bar{n}'_0$ we get:

$$\min_{\bar{n}'} \mathsf{R}_{\log,t} = \frac{\log(f_e/n_e)}{2\tan\theta\cos^2\theta}, \tag{5.42}$$

which is the same as $\mathsf{R}^{\text{side}}_{b,t}$.

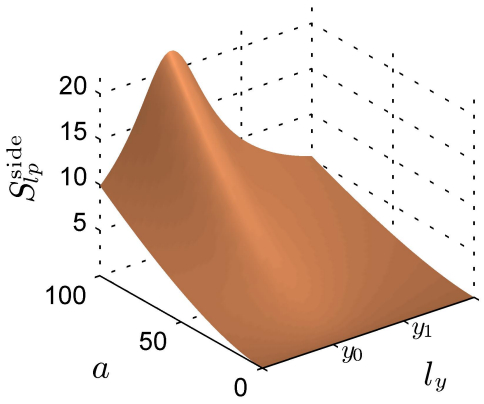

Figure 5.10: ***Storage factor for logarithmic perspective parameterization.*** *S_{lp} over a range of warping parameter values and light positions. The warping parameter is specified here in terms of $a = (\bar{n}' - \bar{n}'_0) \cdot (y_1 - y_0)$. The optimal warping parameter is $a = 0$. The frustum parameters are $n_e = 1$ and $f_e = 1000$. The height of the light above the face is $l_z = 0.5(y_1 - y_0)$.*

5.1.4 Logarithmic + perspective parameterization

Table 5.3 summarizes the error for various parameterizations, including the F_b parameterizations derived in the last chapter. The table shows that error on the same order as that of F_b can be obtained by using a uniform parameterization for the end faces, a perspective projection for the s direction on a side face, and a logarithmic parameterization for t. Our LogPSM parameterization combines a logarithmic transformation with a perspective projection to get good error bounds for both s and t on a side face.

We start with $\mathbf{F}_p$. We multiply $F_{p,t}$ by one constant and add another to transform it into n'/y'. Taking the logarithm and scaling by a third constant yields $F_{\log,t}$. Our logarithmic perspective parameterization is given by:

$$\mathbf{F}_{lp} = \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} F_{p,s} \\ c_0 \log(c_1 F_{p,t} + c_2) \end{bmatrix} \tag{5.43}$$

$$c_0 = \frac{-1}{\log\left(f'/n'\right)}$$

$$c_1 = \frac{n'}{p_3} = -\frac{f' - n'}{f'}$$

$$c_2 = -n'\frac{p_2}{p_3} = 1.$$

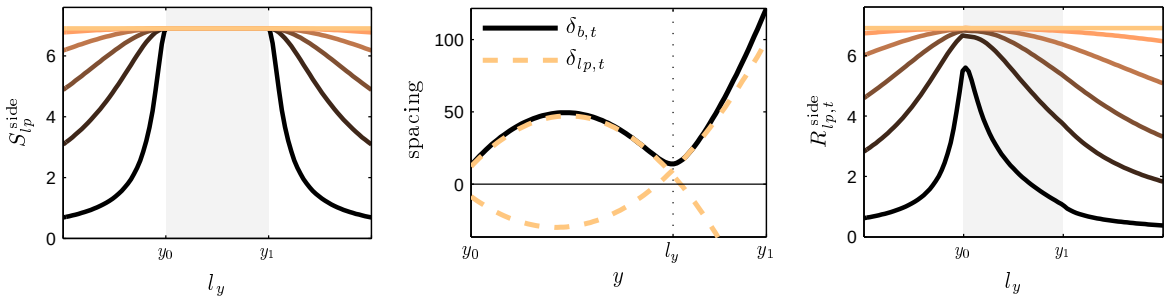

Figure 5.11: ***Critical resolution and storage factors for logarithmic+perspective parameterizations.*** (Left) *Storage factor for the linear form of* $\delta_{lp,t}^{side}$ *that we use for Log-PSMs.* (Middle) *The logarithmic + perspective parameterization can also generate parabolic spacing functions that can give a good fit to* $\delta_{b,t}^{side}$. (Right) *Using two parabolas gives a critical resolution that approaches that of* $\delta_{b,t}^{side}$ *(cf. Figure 4.8). For the plots on the left and the right,* $l_x = 0$ *and* $z = \sigma y_1$*, where from dark to light* $\sigma = 0.1, 0.2, 0.5, 1, \infty$*. The grayed out region indicates light positions over the face. The frustum parameters are* $\theta = 45°$*,* $n_e = 1$*, and* $f_e = 1000$*.*

With $\bar{n}' = \bar{n}'_0$ this parameterization provides the same good error bounds in the s and t directions as the perspective and logarithmic parameterizations, respectively. As seen in Figure 5.6, the optimal S_{lp} for a directional light is much lower than the optimal S_p. Unlike the perspective parameterization, the optimal parameter values are the same for both directions.

The spacing distribution for $F_{lp,t}$ is more general than for $F_{\log,t}$:

$$\delta_{lp,t} = \frac{y'\Big(y'(c_1p_2 + c_2) + c_1p_3\Big)(f' - n')}{c_0c_1p_3} \tag{5.44}$$

$$= \frac{(v + \bar{n}')\Big((v + \bar{n}')(c_1p_2 + c_2) + c_1p_3\Big)W_{ly}}{c_0c_1p_3}. \tag{5.45}$$

$\delta_{lp,t}$ has two real roots:

$$\lambda_0 = -\bar{n}', \qquad \lambda_1 = -\left(\bar{n}' + \frac{c_1p_3}{W_{ly}(c_1p_2 + c_2)}\right). \tag{5.46}$$

If we choose the constants c_1 and c_2 such that $\lambda_1 = -b$, we can write $\delta_{lp,t}$ as:

$$\delta_{lp,t} \sim (v + \bar{n}')(v + b). \tag{5.47}$$

To compute c_1 and c_2 we need an additional constraint, which comes from the requirement that

$F_{lp,t}(0) = 0$. This constraint is satisfied when the argument of the logarithm, $c_1 F_{p,t}(v) + c_2$, is 1. Solving for c_1 and c_2 we get:

$$c_1 = -W_{ly} \frac{\bar{n}'(b - \bar{n}')}{\bar{n}' p_3} \tag{5.48}$$

$$c_2 = W_{ly} \frac{\bar{n}'(p_2(\bar{n}' - b) + p_3)}{b p_3}. \tag{5.49}$$

Plugging these values into Equation 5.45 gives us the normalized spacing distribution:

$$\delta_{lp,t} = \frac{(v + \bar{n}')(v + b)}{c_0(\bar{n}' - b)}. \tag{5.50}$$

The constant c_0 can be computed from the constraint that $F_{lp,t}(1) = 1$:

$$c_0 = \frac{1}{\log\left(\frac{\bar{n}'(b+1)}{b(\bar{n}'+1)}\right)}. \tag{5.51}$$

When $b \to \infty$ the constants converge to those shown for Equation 5.43 and $\delta_{lp,t}$ converges to a linear spacing distribution. Figure 5.11 shows the minimal S_{lp} for varying light positions and $b = \infty$. The $\bar{n}'$ parameter also effects $R_{lp,s}$ and thus must be optimized simultaneously with $R_{lp,t}$ by optimizing S_{lp}. We find the optimal $\bar{n}'$ parameter numerically. Testing a wide range of frustum parameter values and light positions, it appears that the optimal parameter value is always $\bar{n}' = \bar{n}'_0$ (see Figure 5.10 for S_{lp} over a subset of light positions and $\bar{n}'$ values). Note that when the light is over the face, $R_{lp,t}$ is higher with this parameter value than the optimal $R_{\log,t}$. However $\mathsf{R}_{lp,t}$ and $\mathsf{R}_{\log,t}$ are still the same. Multiplying $\mathsf{R}_{lp,s}$ (which is the same as $\mathsf{R}_{p,s}$) with $\mathsf{R}_{lp,t}$ and using the optimal warping parameter $\bar{n}' = \bar{n}'_0$ we get:

$$\min_{\bar{n}'} \mathsf{S}_{lp} = \frac{\log(f_e/n_e)}{2 \tan\theta \cos^3\theta}. \tag{5.52}$$

5.1.4.1 Parabolic spacing functions

Choosing $\bar{n}' = \bar{n}'_0$ and a finite b, such that $b \neq \bar{n}'_0$ and $b > 0$ or $b < -1$, δ_{lp} becomes parabolic. With $b > \bar{n}'_0$, the resulting parabola is concave-up. With $b < \bar{n}'_0$ the parabola is concave-down. Figure 5.12 shows the base error distribution $\widetilde{\mathrm{M}}_{lpu,t}$ and $\widetilde{\mathrm{M}}_{lpd,t}$ corresponding

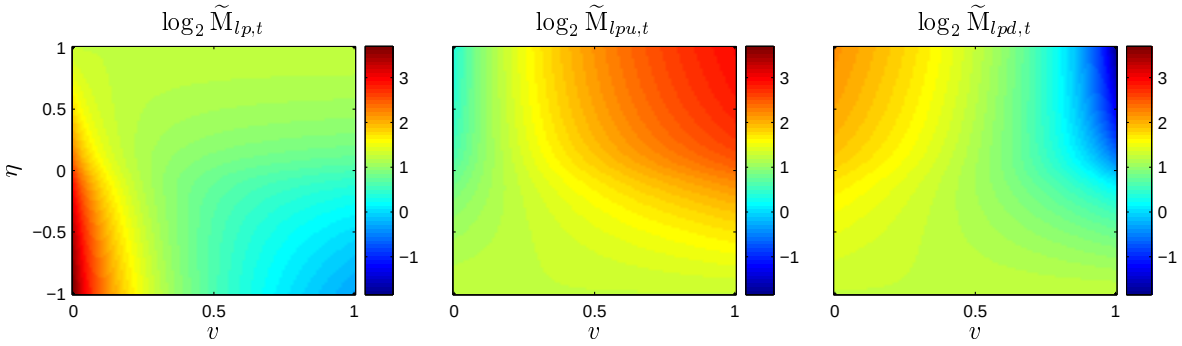

Figure 5.12: ***Error distribution over all warping parameter values for parabolic version of*** δ_{lp}***.*** *Base error distributions* $\widetilde{\mathrm{M}}_{lpu,t}$ *and* $\widetilde{\mathrm{M}}_{lpd,t}$ *corresponding the concave-up and concave-down versions of the parabolic* δ_{lp}*. The linear version* $\widetilde{\mathrm{M}}_{lp}$ *is included for comparison. The frustum parameters are* $n_e = 1$ *and* $f_e = 16$*.*

to the concave-up and concave-down cases, respectively. The warping parameter b for $\widetilde{\mathrm{M}}_{lpu,t}$ is computed as $b = \bar{n}'(\eta)$ using Equation 5.33. The warping parameter for $\widetilde{\mathrm{M}}_{lpd,t}$ is computed as $b = -(1 + \bar{n}'(\eta))$.

Two parabolic spacing distribution functions can be used to better fit the curved portions of $\delta^{\text{side}}_{b,t}$ (see Figure 5.11). Fitting two parabolas, however, is more expensive because it requires that the face be split into two partitions with each partition parameterized separately. The optimal split point y_s is difficult to compute, but $y_s = l_y$ produces good results that are close to optimal. Figure 5.11 also shows the minimal S_{lp} with $y_s = l_y$, $\bar{n}' = \bar{n}'_0$, and b free. The curves for $R_{lp,t}$ are very similar to those for $R^{\text{side}}_{b,t}$. One disadvantage of this parameterization is that a closed-form solution for the optimal parameter b_{opt} for each parabola does not exist. Therefore b_{opt} must be computed numerically, which is somewhat involved. We would like to implement a shadow map algorithm using this parameterization to explore its visual impact.

5.1.5 Handling shear

Due to either limitations on shadow map resolution or the effects of surface orientation, it is not always possible to completely remove aliasing artifacts. When aliasing is present, it is possible to make out the projection of individual shadow map texels onto the scene at shadow boundaries. The shape of the texel projections in the image depends on three factors: the shape of the cross-section of the light beams, the relative orientation of the surface onto

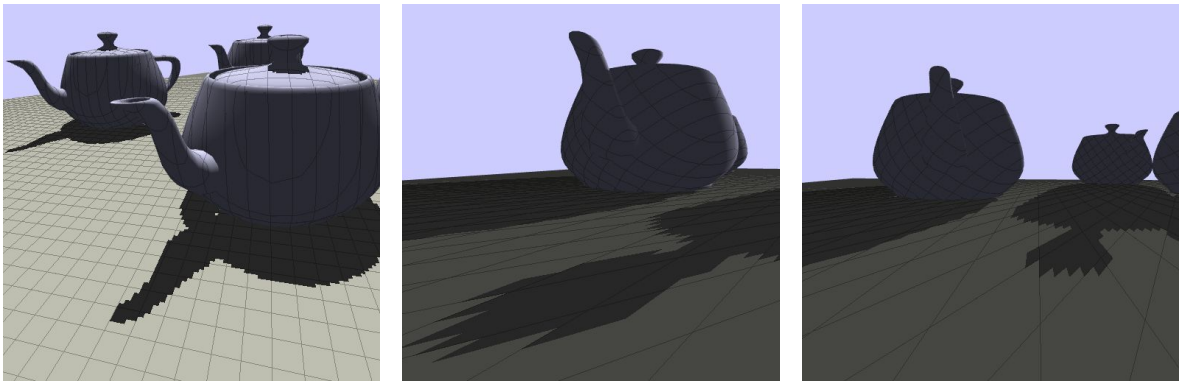

Figure 5.13: ***Shearing artifacts from a standard shadow map.*** (Left) *With the ground plane oriented nearly perpendicular to the light direction the texel projections visible at shadow edges are rectangular.* (Middle) *With the ground plane oriented nearly parallel to the light, the texel projections become extremely sheared. These tend to be more disturbing than rectangular texel projections.* (Right) *From a different point of view, however, the texel projections appear less sheared.*

which the beams project, and the angle from which the projection is viewed. Figure 5.13 shows a standard shadow map with low enough resolution that aliasing is visible. The cross-sections of the light beams are square. When the light is directly over the ground plane the texel projections are mostly rectangular, but when the light strikes the ground plane at a glancing angle, the texels projections become extremely sheared. They look like jagged teeth, an unnatural appearance that is less pleasing than a more rectangular shape. Diffuse shading helps somewhat because as the surface orientation approaches parallel to the light, the contrast between shadowed and unshadowed regions is reduced to zero. From a different viewing angle, the texel projections can appear less sheared, because as the camera moves, the texel projections remain fixed w.r.t the scene.

When we parameterize the faces of the view frustum and fix the texture resolution of the shadow map used for each face, the texel projections on the faces remain fixed regardless of the light or camera position or orientation. Thus for some light positions the light beam cross-sections (measured perpendicular to the beam direction) can become sheared. Warping itself can also cause the cross-sections to become somewhat sheared. The shearing with face partitioning can be more noticeable than that of a standard shadow map because on the same surface the shearing can vary dramatically from one face partition to the next (see Figure 5.14). Because our error metrics do not take this shearing into account, redistributing the

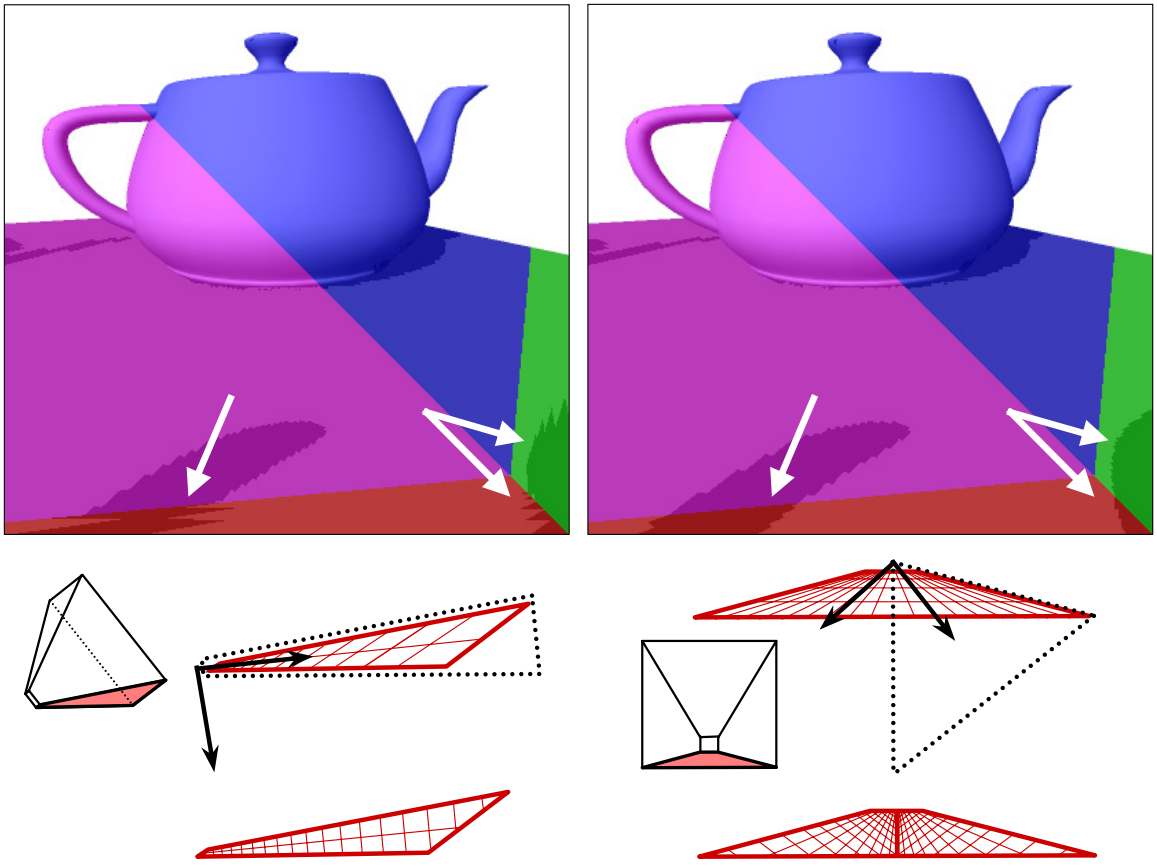

Figure 5.14: ***Handling shear artifacts.*** (Top-left) *Shear artifacts resulting from face partitioning. We use a low-resolution shadow map to make them easier to see. Regions are colored according to the face partition to which they belong.* (Top-right) *Shear artifacts removed by coordinate frame adjustment.* (Bottom) *View frustum as seen from the light in cases in which shearing occurs. To handle the first case, we adjust the coordinate frame by realigning the projection axis with the bisector of the side edges of the face make the other axis orthogonal in the light view. For the second case, we first split the face along the bisector into two sub-faces and apply the coordinate frame adjustment to each of the sub-faces. The corrected faces are shown below.*

resolution according to error may also make this shearing more noticeable.

5.1.5.1 Coordinate frame adjustment

One way to reduce shearing is to compute the parameterization relative to the light instead of the face. Consider the view of a side face from a directional light shown in Figure 5.15. The shearing of the light beam cross-sections can be minimized by fitting a symmetric warping frustum to the face in the light's view. We align the midline of the trapezoid with the bisector

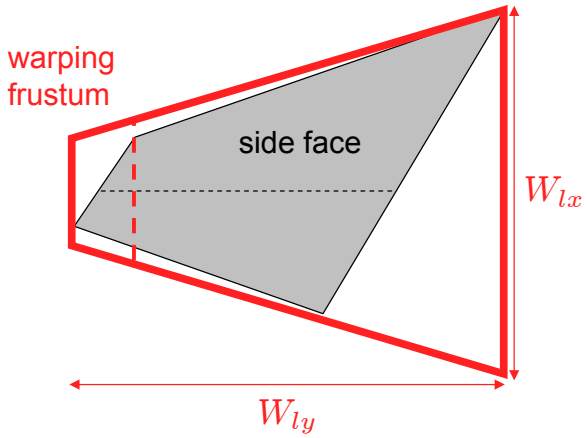

Figure 5.15: ***Coordinate frame adjustment.*** *In the view from a directional light, the symmetric warping frustum is aligned with the bisector of the side face edges in order to minimize shearing. Numerical problems with side faces nearly parallel to the light can be avoided by restricting the warping frustum to the extents of the side edge that touches its far plane. The excluded region (left of the dotted line) can be covered by expanding the partition corresponding to the near plane of the view frustum.*

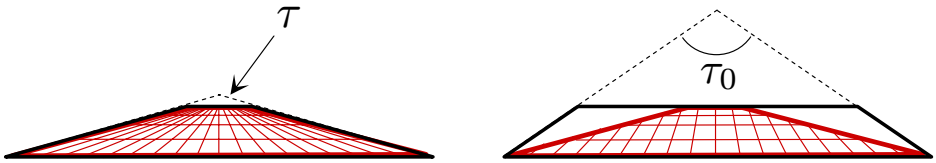

Figure 5.16: ***τ-Limiting.*** (Left) *a face partition with excessive shearing near the edges caused by a large angle τ between the edges of the trapezoidal region to which to parameterization is applied (black).* (Right) *Limiting τ to some constant τ_0 reduces the shearing at the expensive of higher error at the narrow end of the face partition.*

of the two side edges of the face. The resulting projection is no longer a tight fit, but the shearing is reduced.

As the light direction moves parallel to a side face, the area covered by the warping frustum goes to zero, as does the storage factor. However, for a small subset of light directions, the far to near plane distance ratio of the warping frustum can be become very large, leading to numerical robustness problems and high errors for side faces that occupy almost no area in the light view. One way to handle these is to only include part of a side face in the warping frustum as shown in Figure 5.15 and expand near plane face to cover the excluded portion. This only slightly increases the overall storage factor. We have used this approach only for the numerical results presented in this chapter.

5.1.5.2 τ-limiting

When the angle between the two edges of a side face is high, coordinate frame adjustment may provide little improvement (see the red face in Figure 5.17c). One possibility for handling this situation is to simply place a limit the angle τ between the edges of the warping frustum (see Figure 5.16). This approach can lead to large errors on the parts of the face close to the near plane. Naive resolution redistribution allocates most of the resolution to the problem face. The maximum error on the face is equalized with the other faces, but the error over the entire view frustum goes up (Figure 5.17e). We could also use the resolution that would have been allocated to the face had we not changed the parameterization. For surfaces near the viewer, the error may be extremely high at the narrow end of the face partition, but acceptable for surfaces farther away (Figure 5.17f). Depending on the application, this may not be a problem, especially since the high error situations occur for face partitions that cover a very small part of the view frustum. A compromise might be to use a resolution for the face that is some blend of the two extremes.

5.1.5.3 Face splitting

Another possibility of handling large τ is to first split the face along its bisector and then apply the coordinate frame adjustment to each half. We split a face when the angle between the side edges τ exceeds a specified threshold τ_0. To avoid a sudden pop when a face is split, we specify another threshold $\tau_1 > \tau_0$ and "ease in" to the new coordinate frame over the interval $[\tau_0, \tau_1]$. Choosing $\tau_0 > 90^\circ$ ensures that no face will be split more than once and that no more than two faces will be split at the same time. This is an awkward solution because it requires additional partitions to handle problem faces which occupy little area in the light's view. However, it does help in many cases.

Neither τ-limiting nor face splitting is completely satisfactory. We have achieved the best results, however, using the coordinate frame adjustment in combination with face splitting. As we shall see later in Section 5.3, this scheme has only a minor affect on the average number of shadow maps required over all light positions.

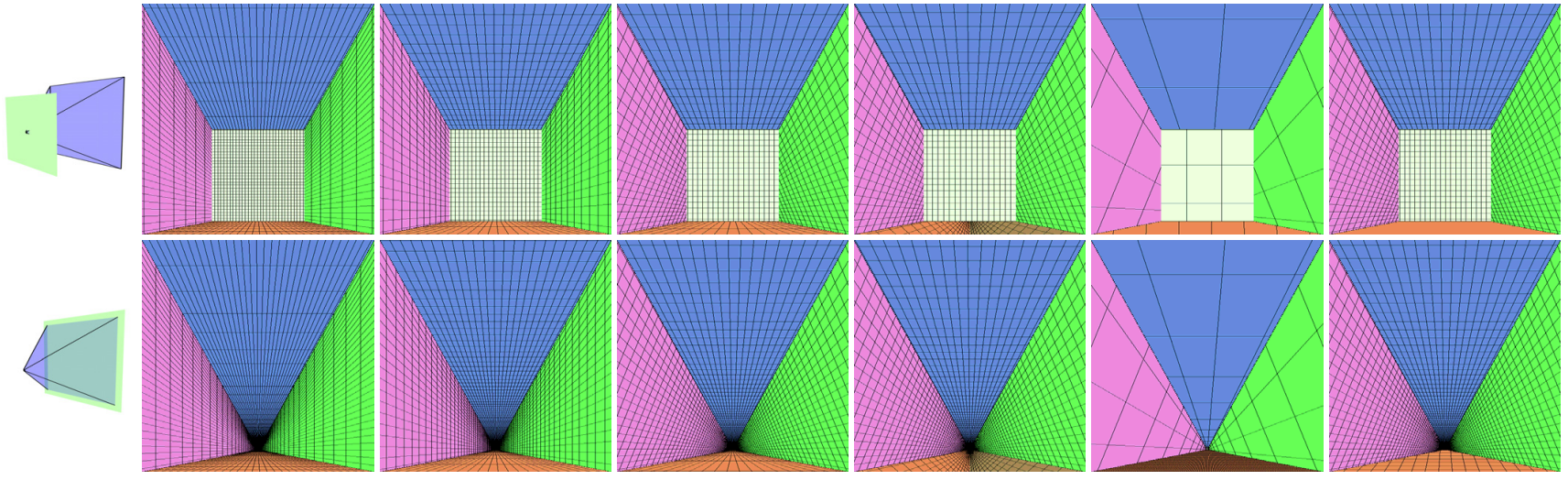

Figure 5.17: ***The effects of resolution redistribution and shear handling.*** *The top row shows the texel grid of the shadow map projected onto a plane near the viewer that is oriented perpendicular to the view direction. The bottom row uses a plane at the far end of the view frustum. (a) The same resolution is used in s and t for all partitions. The error is distributed unequally. (b) Resolution is distributed according to maximum error resulting in a more uniform parameterization. (c) Coordinate frame adjustment alleviates excessive shearing in the upper corners of the left and right face partitions, but does nothing for the bottom one. (d) Splitting the bottom face and adjusting the coordinate frame alleviates shearing. (e) Limiting the field of view of the parameterization applied to the bottom face results in high error. Resolution redistribution leaves little for the other partitions. (f) Using the original resolution allocation leads to high error close to viewer but acceptable results further away.*

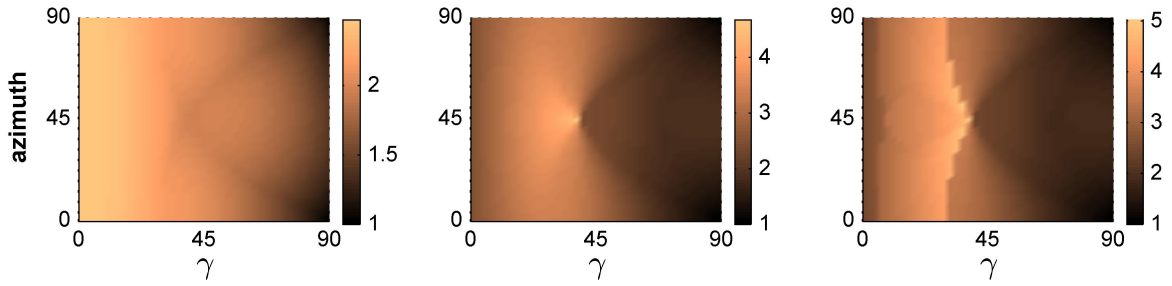

Figure 5.18: ***Total error distribution for face partitioning over varying light directions***. *These graphs show the storage factor produced when using* $\mathbf{F}_{lp}$ *on side faces with no shear handling (left), coordinate frame adjustment (middle), and coordinate frame adjustment with splitting (right).* γ *is the angle between the light direction and the view direction. The azimuthal angle is measured from the* x*-axis in eye space. The minimum error value is normalized to* 1 *to highlight the variation in error. The shape of the error distributions produced by using* $\mathbf{F}_p$ *on side faces are nearly identical to these shown here. The frustum parameters are* $n_e = 1$*,* $f_e = 1000$*, and* $\theta = 30°$*. The face splitting parameters are* $\tau_0 = 100°$ *and* $\tau_1 = 110°$*.*

5.1.6 Total error over all light directions

So far we have considered the error for only one face at a time. The total error of a face partitioning algorithm can be computed by simply summing the storage factors computed for each face.

Figure 5.18 shows the error distribution in terms of the storage factor over a quarter hemisphere (the error distribution over the whole hemisphere can be obtained by symmetry). F_{lp} gives the smoothest error distribution. However, the storage factor does not take shearing into account. Adding coordinate frame adjustment and face splitting to handle shearing introduces some sharper features in the error distribution. The change in storage factor using face splitting is continuous with change in light position, but changes very rapidly when a face changes from an entrance face to an exit face, causing a ridge in the error distribution. The magnitude of this ridge is small enough however that its effects are not noticeable as the light moves around. Section 5.3 contains some additional analysis of the error distributions over all light directions for face partitioning and other algorithms.

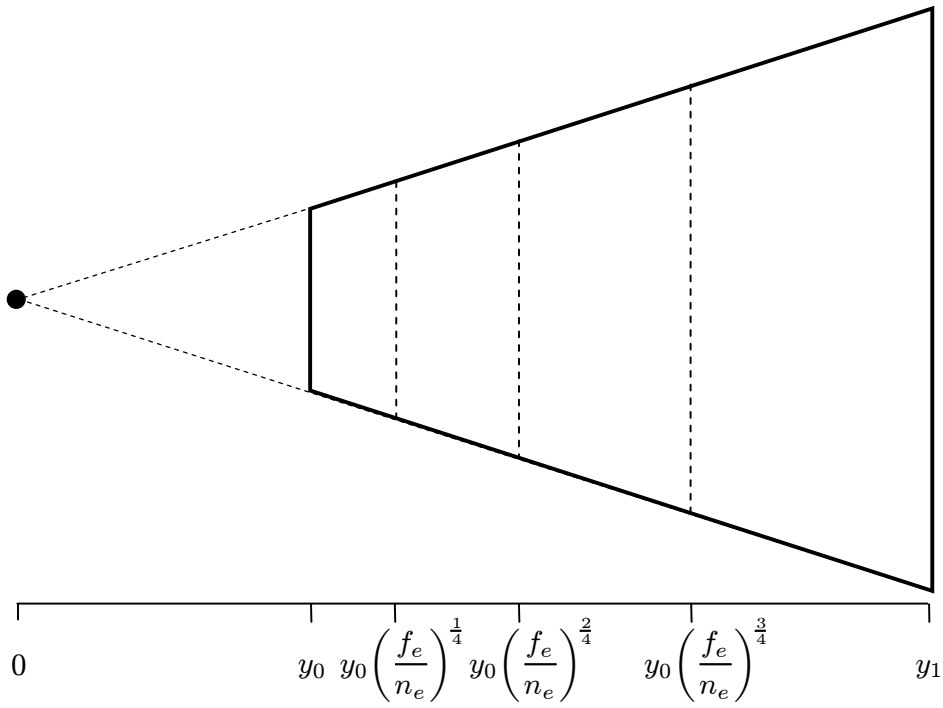

Figure 5.19: ***z-partitioning applied to a face.*** *Choosing self-similar partitions makes the error the same in each partition and minimal over all possible partitionings.*

5.1.7 Applying z-partitioning to faces

z-partitioning could be applied to side faces to reduce the error for uniform and perspective parameterizations. (A more apt name might be y-partitioning for the case of faces, but we will stick with our previous terminology since the y-axis on the faces corresponds roughly with the eye space z-axis). The main difference between various z-partitioning algorithms is where partitions are made. The choice of partition locations affects the errors in each partition. We can see from Table 5.3 that the storage factor (and thus the error) grows with f_e/n_e. To minimize the maximum error over all partitions, we should therefore minimize the far to near plane distance ratio for each partition and ensure that the maximum error of each partition is the same. This can be accomplished by choosing the partition points $y_{0,i}$ and $y_{1,i}$

for $i \in 1, 2, ..., k$ such that the partitions are self-similar (as shown in Figure 5.19):

$$\begin{aligned} y_{0,i} &= y_0 \left(\frac{f_e}{n_e}\right)^{(i-1)/k} \\ y_{1,i} &= y_{0,i+1} = y_0 \left(\frac{f_e}{n_e}\right)^{i/k} \\ \frac{y_{1,i}}{y_{0,i}} &= \left(\frac{f_e}{n_e}\right)^{1/k}. \end{aligned} \tag{5.53}$$

This z-partitioning scheme can be viewed as a discrete approximation of the logarithmic parameterization with $n' = n_e$. With this warping parameter, the inverse of $F^{\text{side}}_{\log}$, $G^{\text{side}}_{\log}$, is given by:

$$G^{\text{side}}_{\log}(t) = y_0 \left(\frac{f_e}{n_e}\right)^{t}. \tag{5.54}$$

Partitioning the t domain of the shadow map into k equal pieces creates the corresponding partitions on the face computed by Equation 5.53. We will denote this partitioning scheme $\mathsf{ZP}_{\mathsf{log}}$. The optimal maximum storage factors for the uniform and perspective parameterizations with these partition locations can be computed from Equations 5.13 and 5.38 using $y_{1,i}/y_{0,i}$ in place of f_e/n_e:

$$\begin{aligned} \mathsf{S}^{\text{side}}_{un,k} &= \sum_{i=1}^{k} (y_{1,i}/y_{0,i}) \frac{((y_{1,i}/y_{0,i}) - 1)}{2 \tan\theta \cos^3\theta} \\ &= k(f_e/n_e)^{1/k} \frac{\left((f_e/n_e)^{1/k} - 1\right)}{2 \tan\theta \cos^3\theta} \end{aligned} \tag{5.55}$$

$$\begin{aligned} \mathsf{S}^{\text{side}}_{p,k} &= \sum_{i=1}^{k} \frac{((y_{1,i}/y_{0,i}) - 1)}{2 \tan\theta \cos^3\theta} \\ &= k \frac{(f_e/n_e)^{1/k} - 1}{2 \tan\theta \cos^3\theta} \end{aligned} \tag{5.56}$$

The logarithmic perspective parameterization is actually insensitive to the partitioning scheme used, so long as the partitions are adjacent, i.e. $y_{0,i+1} = y_{1,i}$. From Equation 5.52 we get:

$$\begin{aligned}
\mathsf{S}^{\text{side}}_{lp,k} &= \frac{1}{2\tan\theta\cos^3\theta}\sum_{i=1}^{k}\log(y_{1,i}/y_{0,i}) \\
&= \frac{1}{2\tan\theta\cos^3\theta}\sum_{i=1}^{k}\log(y_{0,i+1}) - \log(y_{0,i}) \\
&= \frac{1}{2\tan\theta\cos^3\theta}\left(\log(y_{0,k+1}) - \log(y_{0,1})\right) \\
&= \frac{\log(f_e/n_e)}{2\tan\theta\cos^3\theta} = \mathsf{S}^{\text{side}}_{lp}. \qquad (5.57)
\end{aligned}$$

As we increase the number of partitions, both $\mathsf{S}^{\text{side}}_{un,k}$ and $\mathsf{S}^{\text{side}}_{p,k}$ converge to $\mathsf{S}^{\text{side}}_{lp}$. The limit $\lim_{k\to\infty}\mathsf{S}^{\text{side}}_{p,k}$ yields the indefinite form $\infty\cdot 0$. Rewriting $\mathsf{S}^{\text{side}}_{p,k}$ as a quotient and applying L'Hospital's rule we get:

$$\begin{aligned}
\lim_{k\to\infty} k\frac{(f_e/n_e)^{1/k}-1}{2\tan\theta\cos^3\theta} &= \lim_{k\to\infty}\frac{(f_e/n_e)^{1/k}-1}{2/k\tan\theta\cos^3\theta} \\
&= \lim_{k\to\infty}\frac{\frac{-(f_e/n_e)^{1/k}\log(f_e/n_e)}{k^2}}{-2/k^2\tan\theta\cos^3\theta} \\
&= \frac{\log(f_e/n_e)}{2\tan\theta\cos^3\theta}.
\end{aligned}$$

The limit $\lim_{k\to\infty}\mathsf{S}^{\text{side}}_{un,k}$ can be computed similarly.

Zhang et al. (2006) use a combination of $\mathsf{ZP}_{\mathsf{log}}$ with a linear partitioning scheme for their parallel split shadow maps (PSSMs). The linear partitioning scheme $\mathsf{ZP}_{\mathsf{lin}}$ creates uniform width partitions:

$$y^{\mathsf{lin}}_{0,i} = y_0 + (y_1 - y_0)\frac{i-1}{k} \qquad (5.58)$$

$$y^{\mathsf{lin}}_{1,i} = y_{0,i+1} = y_0 + (y_1 - y_0)\frac{i}{k}. \qquad (5.59)$$

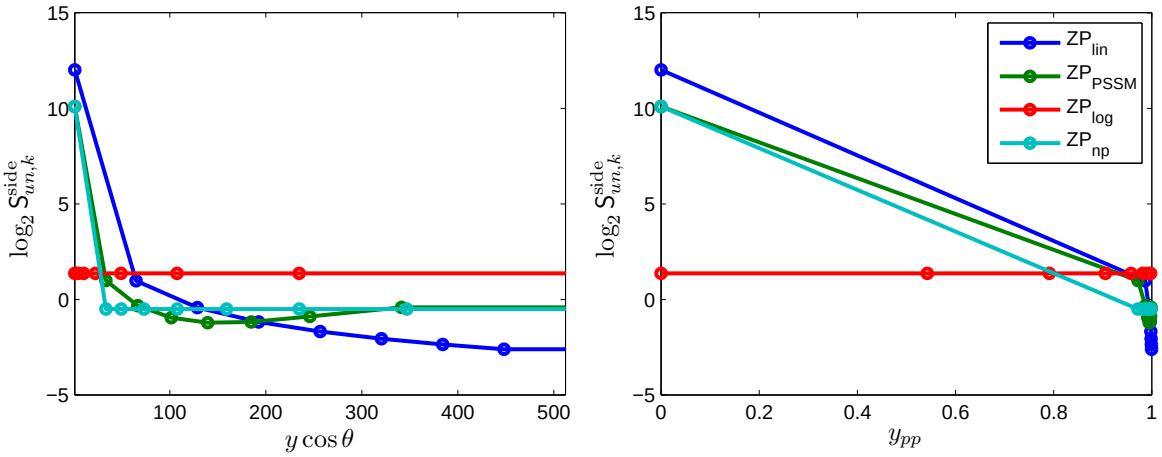

Figure 5.20: ***Maximum error for different z-partitioning schemes.*** *These graphs show the maximum error over each partition in terms of* $\mathsf{S}^{side}_{un,k}$ *for various z-partitioning schemes. 8 partitions were used for these graphs. The graph on the right shows the locations of the partitions in the post-perspective space of the camera. The* $1/(2\tan\theta\cos^3\theta)$ *term has been factored out.* $f_e/n_e = 512$.

The PSSM scheme, $\mathsf{ZP}_{\mathsf{PSSM}}$, takes the average of the partition locations for $\mathsf{ZP}_{\mathsf{log}}$ and $\mathsf{ZP}_{\mathsf{lin}}$:

$$y^{\mathsf{PSSM}}_{0,i} = \frac{y^{\mathsf{lin}}_{0,i} + y^{\mathsf{log}}_{0,i}}{2} \tag{5.60}$$

$$y^{\mathsf{PSSM}}_{1,i} = \frac{y^{\mathsf{lin}}_{1,i} + y^{\mathsf{log}}_{1,i}}{2}. \tag{5.61}$$

Figure 5.20 shows the error for these z-partitioning schemes. $\mathsf{ZP}_{\mathsf{PSSM}}$ trades off high error near the viewer in favor of areas farther away. Though the areas near the viewer occupy just a small part of the view frustum in world space, these areas occupy most of the view frustum in the post-perspective space of the camera, and therefore they occupy most of the image as well.

5.1.7.1 Continuity

As the light position changes, the error on the faces also changes. It may be tempting to apply z-partitioning adaptively, i.e. to use it only for faces with high error. However, if the resolution of the shadow map is not sufficiently high, changing the number of z-partitions from frame to frame can cause temporal artifacts because it causes an abrupt shift in the distribution of aliasing error. In general, it is best to use the same number of z-partitions

for all faces and light directions to avoid popping. It might be possible in some situations to add z-partitions without popping by introducing a zero-width partition and morphing the partition locations to their new locations over multiple frames. We have not tried this however.

5.1.8 A z-partitioning scheme using a pseudo-near plane

The rationale for $\mathsf{ZP}_{\mathsf{PSSM}}$ is that the regions near the viewer are often empty. If this is the case, the best thing to do is to increase the near plane distance of the view frustum. Even small increases in n_e can dramatically reduce the error. Computing a tight upper bound on n_e can be expensive, though, because it can involve clipping geometry against the view frustum. Therefore, it is common practice to set n_e to a conservatively low value to avoid near plane clipping of geometry. An approach like $\mathsf{ZP}_{\mathsf{PSSM}}$ allows small values of n_e while keeping the error low for the middle range of the view frustum. It still provides shadow map coverage for regions close to the viewer just in case some surface actually does happen to enter the region, but here the error might be high.

We propose an alternative approach that uses a *pseudo-near plane* to provide information about the part of the frustum that the scene is likely to occupy. This allows us to still use a conservative n_e while keeping the aliasing error low. The price to be paid is that between n_e and the pseudo-near plane distance n_p the error can be high. For a side face, the point corresponding to n_p is $y_{np} = n_p / \cos\theta$. We compute the partition locations as follows:

$$y_{0,i} = \begin{cases} y_0 & i = 1 \\ y_{np}(f_e/n_p)^{(i-1)/k} & i > 1 \end{cases} \tag{5.62}$$

$$y_{1,i} = y_{np}(f_e/n_p)^{(i-1)/k}. \tag{5.63}$$

We denote this scheme as $\mathsf{ZP}_{\mathsf{np}}$. If we set $n_p = y_{1,1}^{\mathsf{PSSM}} \cos\theta$ then we get the same maximum error in the first partition as $\mathsf{ZP}_{\mathsf{PSSM}}$, but the maximum error over for the subsequent partitions is lower than that of the corresponding partitions of $\mathsf{ZP}_{\mathsf{PSSM}}$ (see Figure 5.20).

Because no single partitioning scheme can meet the needs of all applications we would

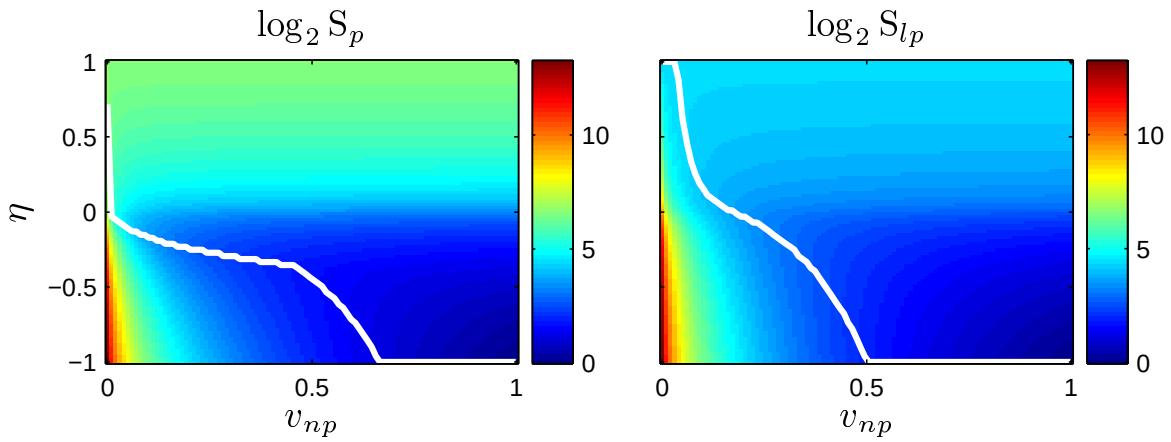

Figure 5.21: ***Warping parameters for varying pseudo-near plane positions.*** *These graphs show the storage factor* S *for a perspective parameterization (left) and a LogPSM parameterization (right) computed over the range* $[v_{np}, 1]$*, where* v_{np} *is the position of the pseudo-near plane. The warping parameter yielding the minimum* S *is shown by the white curve.*

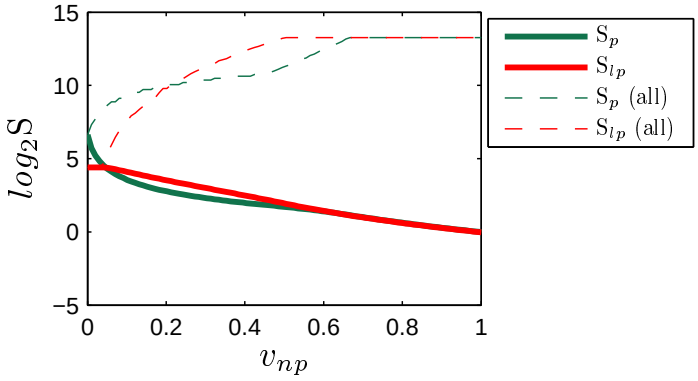

Figure 5.22: ***Comparison of maximum error for varying pseudo-near plane positions.*** *This graph shows the minimal storage factor* S *over both the range* $[v_{np}, 1]$ *(heavy lines) and the entire* $[0, 1]$ *range (dashed lines).*

like to develop a tool that would allow users to manipulate partition points interactively on graphs similar to those in Figure 5.20 to achieve the desired error distribution. We would also like to explore the possibility of creating partitions automatically to match a user specified error profile as closely as possible.

5.1.9 Warping algorithm using a pseudo-near plane

In the same spirit as $\mathsf{ZP_{np}}$, we can formulate a warping algorithm that uses a pseudo-near plane. For this algorithm we minimize the maximum error over the part of the face with

$v > v_{np}$, where $v_{np} \in [0, 1]$ is the location of the pseudo-near plane. Figure 5.21 shows S over just the range $[v_{np}, 1]$ for varying v_{np} as well as the curves for the warping parameter value that produces the minimal S for a given v_{np}. Initially, as v_{np} increases from 0, the optimal $\bar{n}'$ of a perspective parameterization results in the same error at the pseudo-near and far planes. $\bar{n}'$ can thus be found in the same way as $\bar{n}'_{\text{LiSPSM}}$, by solving for the $\bar{n}'$ that makes the values of $\widetilde{\text{M}}_{p,t}$ equal to each other at $v = v_{np}$ and $v = 1$:

$$\bar{n}'_{\text{opt}} = \bar{n}'_0 + \sqrt{(\bar{n}'_0 + v_{np})(\bar{n}'_0 + 1)}. \tag{5.64}$$

As v_{np} continues to increase, so does $\bar{n}'$. At a certain point the optimal maximum error is found only at v_{np} and not at the far plane. At this point Equation 5.64 produces a value that is too small. The optimal $\bar{n}'$ can then be found by solving $\text{dS}_p(v_{np})/\text{d}\bar{n}' = 0$ for $\bar{n}'$. We find three solutions, a double root at $-v_{np}$, which is invalid because of the constraint that $\bar{n}' > \bar{n}'_0$, and a third root which provides the correct solution:

$$\bar{n}'_{\text{opt}} = \frac{v_{np}}{2 - 3v_{np}}. \tag{5.65}$$

As v_{np} continues to increase to 2/3, $\bar{n}'_{\text{opt}}$ approaches infinity. Beyond $v_{np} = 2/3$, $\bar{n}'_{\text{opt}}$ remains at infinity. For $v_{np} < 2/3$ we choose $\bar{n}'_{\text{opt}}$ simply by taking the maximum of Equations 5.64 and 5.65.

A similar algorithm can be formulated using the LogPSM parameterization by finding the $\bar{n}'_{\text{opt}}$ that minimizes S_{lp} computed over the range $[v_{np}, 1]$. However $\text{dS}_{lp}(v_{np})/\text{d}\bar{n}' = 0$ has no analytic solution and must be solved numerically. One possible way to do this is to parameterize S_{lp} with η and use a bracketed minimization method (e.g. golden section search (Press et al., 1992)) to find the optimal η. Figure 5.22 compares the maximum error when using both the perspective and LogPSM parameterizations. Overall, the graphs are quite similar. Figure 5.23 shows a comparison between the LiSPSM algorithm and a perspective warping algorithm that uses the pseudo-near plane to compute the warping parameter. Because the scene occupies only a portion of the view frustum, lower error can be obtained by setting the pseudo-near plane farther away from the viewer.

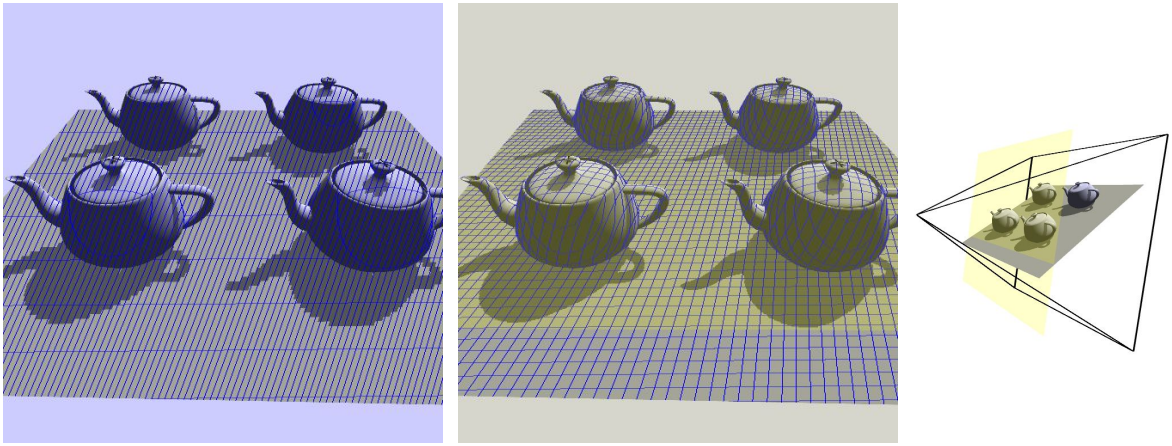

Figure 5.23: ***Comparison of LiSPSM and pseudo-near plane algorithms.*** *(Left) LiSPSM algorithm. (Middle) Algorithm that uses a pseudo-near plane to compute the optimal perspective warping parameter. The pseudo-near plane is shown in transparent yellow. (Right) A third person view showing the position of the view frustum and pseudo-near plane.*

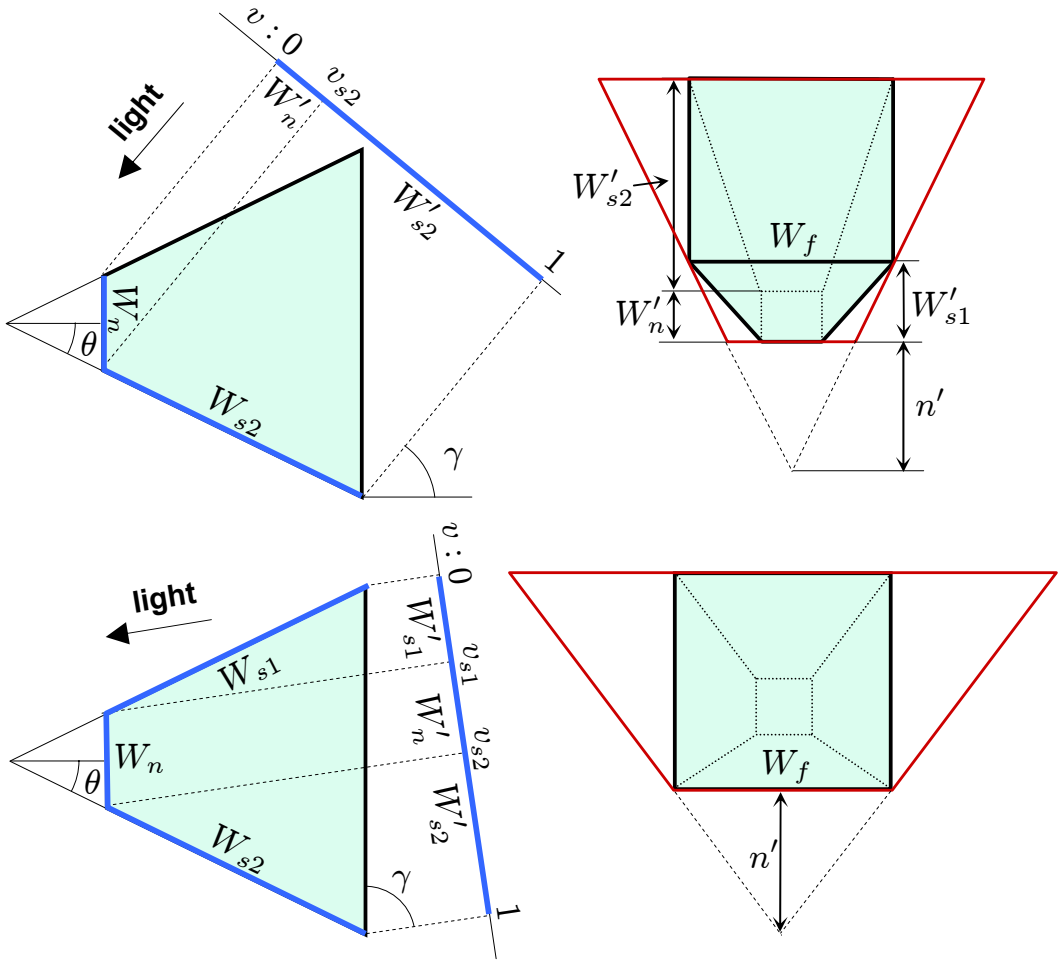

Figure 5.24: ***Parameterizing the frustum.*** *(Left) A side view of the view frustum. The back faces of the view frustum are projected onto the light image plane, which is oriented perpendicular to the light direction. (Right) The light's view of the view frustum. The warping frustum surrounding the view frustum is shown in red. On the top row $\gamma > \theta$ and on the bottom row $\gamma < \theta$.*

5.2 Frustum parameterization

In this section we analyze the error for parameterizations covering the entire view frustum. From an overhead directional light, the view frustum looks just like a side face and the analysis is very similar. This is the configuration that has been analyzed in previous work (Stamminger & Drettakis, 2002; Wimmer et al., 2004). For lights in general position, especially point lights, the analysis becomes more tedious. The aliasing error equations have several forms depending on the position of the light. The main reason for this is that the subset of the view frustum vertices that constrain a tight-fitting warping frustum changes as the light position

changes. We make several simplifications to keep the equations as simple as possible. We restrict our attention to light positions in the plane formed by the view frustum y and z axes. Furthermore, we consider only directional lights here. The analysis can be extended to point lights, but the equations become more involved. Frustum parameterization methods are typically used for directional lights anyway, or for spot lights with a fairly narrow field of view which behave much like directional lights.

Frustum parameterizations typically have the lowest error for a light direction perpendicular to the view direction. However, as the light direction approaches parallel or anti-parallel with the view direction, the parameterization must revert to uniform to avoid high error. One of the main purposes of the analysis in this section is to determine how to smoothly transition from optimal warping to no warping while still keeping the error low.

Figure 5.24 shows the light image plane and warping frustum surrounding the view frustum. We parameterize the light direction by the angle γ between the light and view directions. We will consider only $\gamma \in [0°, 90°]$. For this range, the maximum perspective error occurs on the back faces of the view frustum. The near face is back-facing for all $\gamma < 90°$. Both side faces are back-facing when $\gamma < \theta$, while only one of them is for $\gamma > \theta$. We orient the light image plane perpendicular to the light direction. To compute bounds on perspective aliasing error we need to be able to compute δ_b (Equation 4.16) for points along the appropriate faces. The various terms of δ_b can be computed from Figure 5.24 using simple geometry:

$$W_{ly} = W'_n + W'_{s2} + \begin{cases} W'_{s1} & \gamma \in [0, \theta] \\ 0 & \gamma \in [\theta, 90°] \end{cases} \tag{5.66}$$

$$W_{lx} = (n' + W_{ly}) \begin{cases} \frac{W_f}{n'} & \gamma \in [0, \theta] \\ \frac{W_f}{n' + W'_{s1}} & \gamma \in [\theta, 90°] \end{cases} \tag{5.67}$$

$$d_e(v) = n_e + (f_e - n_e) \max\left(\frac{W_{ly}(v_{s1} - v)}{W'_{s1}}, 0, \frac{W_{ly}(1 - v)}{W'_{s2}} \right) \tag{5.68}$$

$$v_{s1} = \begin{cases} \frac{W'_{s1}}{W_{ly}} & \gamma \in [0, \theta] \\ 0 & \gamma \in [\theta, 90°] \end{cases} \qquad v_{s2} = v_{s1} + \frac{W'_n}{W_{ly}} \tag{5.69}$$

$$W'_n = W_n \cos\gamma, \tag{5.70}$$

$$W'_{s1} = W_s \left(1 - \cos(\theta - \gamma)\right) \tag{5.71}$$

$$W'_{s2} = W_s \sin(\theta - \gamma) \tag{5.72}$$

$$W_n = W_e, \qquad W_f = W_n \frac{f_e}{n_e}, \qquad W_s = \frac{f_e - n_e}{\cos\theta}. \tag{5.73}$$

Because we are using a directional light source and the light image plane is perpendicular to the light direction, both the n_l/d_l and $\cos\phi_l$ terms of δ_b are 1. With these values, the equations for our bounds on perspective aliasing error can be written as:

$$\widetilde{M}_s(u,v) = \frac{r_i}{r_s}\frac{\delta_l(u,v)}{\delta_{b,s}(u,v)} = \widetilde{\mathrm{M}}_s(u,v)\frac{r_i}{r_s}\frac{1}{\cos\theta} \qquad \widetilde{\mathrm{M}}_s(u,v) = \delta_l(u,v)\frac{W_{lx}}{W_e}\frac{n_e}{d_e(v)} \tag{5.74}$$

$$\widetilde{M}_t(v) = \frac{r_j}{r_t}\frac{\delta_l(v)}{\delta_{b,t}(v)} = \widetilde{\mathrm{M}}_t(v)\frac{r_j}{r_t}\frac{1}{\cos\theta} \qquad \widetilde{\mathrm{M}}_t(v) = \delta_l(v)\frac{W_{ly}}{W_e}\frac{n_e}{d_e(v)}. \tag{5.75}$$

The equations for $\widetilde{\mathrm{M}}_s$ and $\widetilde{\mathrm{M}}_t$ are slightly different than those that we defined at the beginning of Section 5.1. Because it is not possible to define a $\widetilde{\mathrm{M}}$ that is completely independent of θ for frustum parameterizations, we have decide to include the $W_e/n_e = 2\tan\theta$ term in $\widetilde{\mathrm{M}}_s$ and $\widetilde{\mathrm{M}}_t$. The spacing distribution functions for the various parameterizations considered in the last section can be used directly with these modified base error distribution equations. We note that $\widetilde{\mathrm{M}}_{p,s}$ and $\widetilde{\mathrm{M}}_{p,t}$ are similar to the equations used in the analysis of perspective warping by Zhang et al. (2006). The main difference is that they analyze error along the view direction while we evaluate the error along the faces.

Figure 5.25 shows $\widetilde{\mathrm{M}}$ of perspective and logarithmic parameterizations for different γ. For all $\gamma \in [0°, 90°]$, the maximum error occurs on the edge of the near plane and the side face furthest from the $v = 0$ at $v = v_{s2}$. For $\gamma \in [\theta, 90°]$, $\widetilde{\mathrm{M}}$ does not change very much, especially if the f_e/n_e ratio is high. When $\gamma = 0°$, the maximum error is minimized for all parameterizations at $\eta = -1$, which is why the single shadow map warping algorithms all degenerate to a uniform parameterization for this light direction.

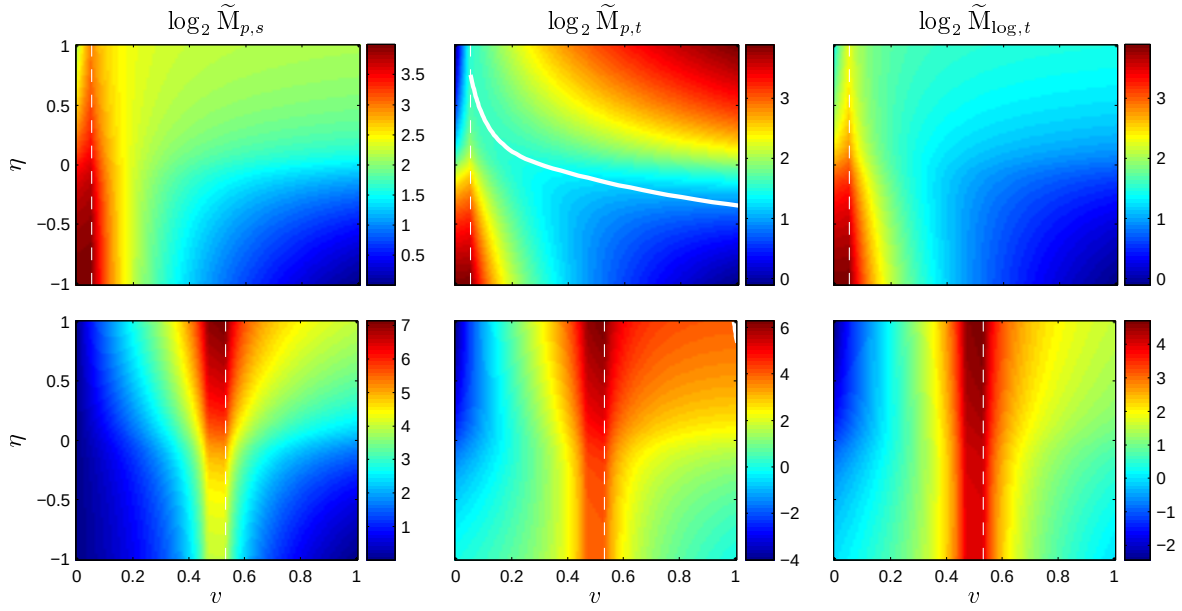

Figure 5.25: ***Error distribution for varying warping parameter values.*** *(Top)* $\gamma = 40°$*. These graphs are very similar to those shown in Figure 5.5 for a side face, except for the portion between* $v = 0$ *and* $v = v_{s2}$ *(indicated by the dashed white line) which corresponds to the near face of the view frustum. The white line on the graph for* $\widetilde{\mathrm{M}}_{p,t}$ *is the location of the minimum error over* v *as a function of* η*. (Bottom)* $\gamma = 0°$*. The frustum parameters are* $n_e = 1$*,* $f_e = 16$*, and* $\theta = 30°$*.*

5.2.1 Warping parameter shaping functions

In order to compute a function that gives us a smooth degeneration to a uniform parameterization as $\gamma \to 0°$ while keeping the error low, we examine the behavior of $\widetilde{\mathrm{M}}$ over varying γ in Figure 5.26. Based on the analysis of Figure 5.25, we plot the error at $v = v_{s2}$, $v = 1$, and for $\widetilde{\mathrm{M}}_{p,t}$, at the location of the minimum $v = v_{\min}$. For an overhead light, both R and the variation in $\widetilde{\mathrm{M}}$ for $F_{p,s}$ and $F_{lp,t}$ are minimal at $\eta = 1$. As γ decreases, the variation in error increases slightly at $\eta = 1$, but $\mathrm{R}_{p,s}$ rises significantly, while $\mathrm{R}_{lp,t}$ remains practically the same. At $\gamma = \theta$, there is a rapid shift in R for $\eta \in [0, 1]$. $\widetilde{\mathrm{M}}_{p,s}$ also sees a rapid change at $v = v_{s2}$ for η in this range, but the shift in $\mathrm{R}_{p,t}$ is more gradual because the maximum error is already high. As γ continues to decrease, the maximum error for all the functions grows steadily until it is higher than that of a uniform parameterization for all $\eta \neq 0$. From this we can see that as γ decreases, it is important that η be close to 0 by the time $\gamma = \theta$. As γ continues to decrease, η should eventually decrease to -1.

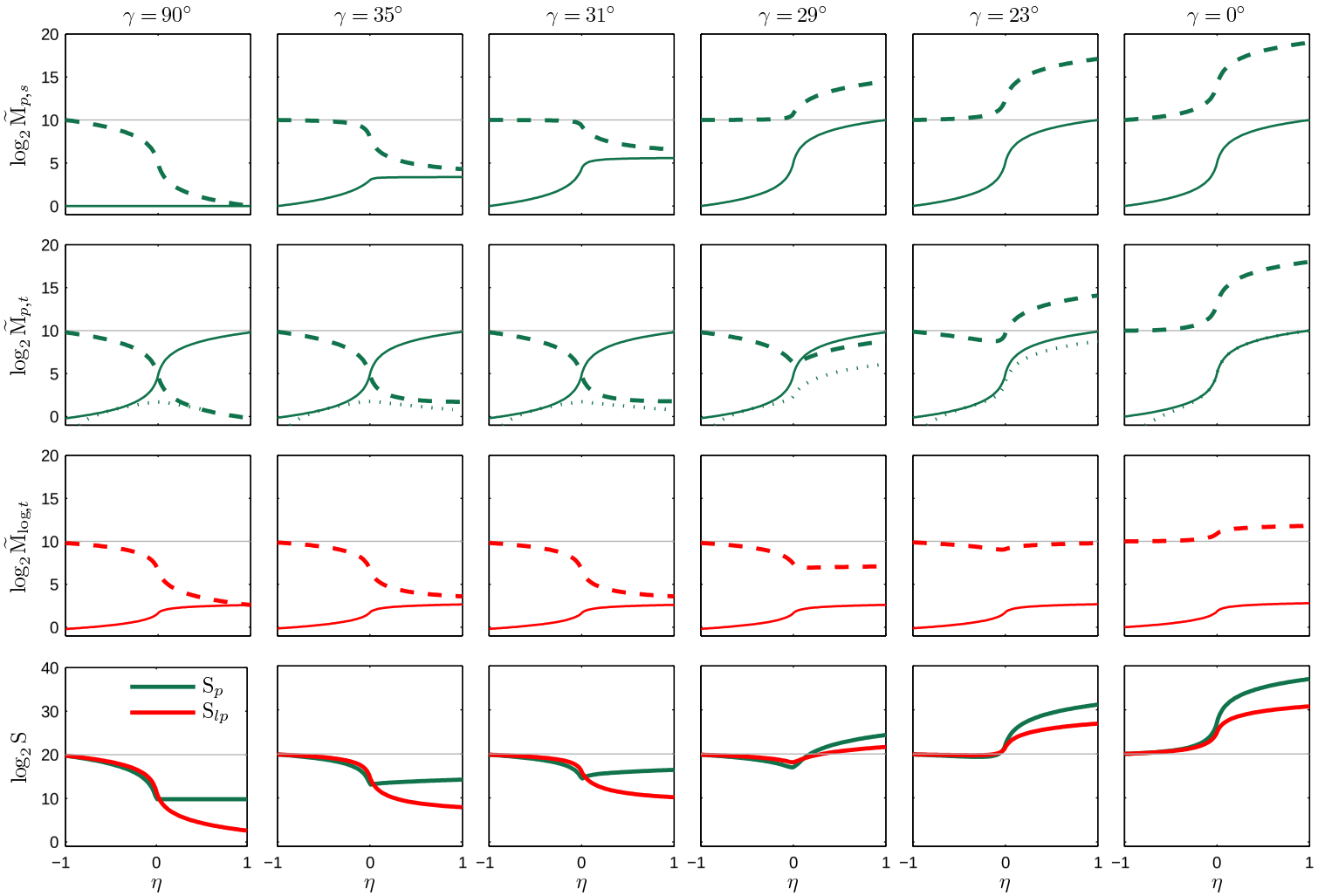

Figure 5.26: ***Perspective error over all warping parameter values for several light directions.*** *These graphs plot the error at $v = v_{s2}$ (heavy dotted line), $v = 1$ (thin solid line), and for $\widetilde{\mathrm{M}}_{p,t}$ $v = v_{\min}$ (thin dotted line), where $v_{\min}$ is the location of the minimum value. The frustum parameters are $n_e = 1$, $f_e = 1024$, and $\theta = 30°$.*

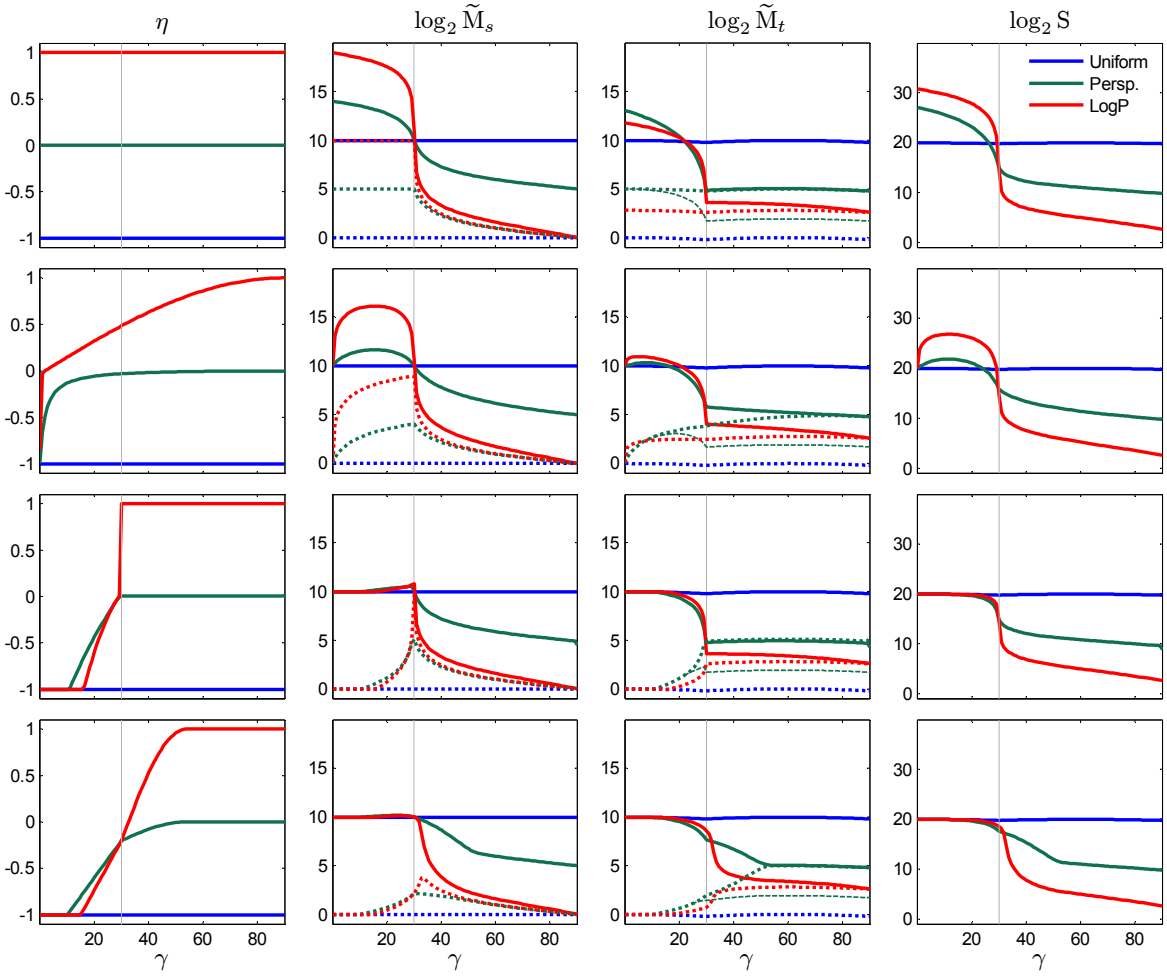

Figure 5.27: ***Various shaping functions for the warping parameter.*** $\widetilde{\mathrm{M}}$ *plots are for* $v = v_{s2}$ *(solid lines),* $v = 1$ *(dotted lines), and with* $\widetilde{\mathrm{M}}_{p,t}$, $v = v_{\min}$ *(dashed line).* (First row) *The optimal parameter value for* $\gamma = 90°$ *is used over all light directions (no shaping function). The error is extremely high for* $\gamma < \theta$. (Second row) *The LiSPSM's* $1/\sin\gamma$ *factor is used to ramp off* n'. *(*n' *is converted to* η *in this graph).* (Third row) *The warping parameter value is chosen to minimize* S, *but the transition from low to high error at* $\gamma = \theta$ *is too rapid.* (Fourth row) *Our shaping function keeps the error low for* $\gamma > \theta$ *while avoiding excessive error for* $\gamma < \theta$. *In addition, it provides a smoother transition. Note that the uniform parameterization actually does not have a warping parameter. The other two parameterizations converge to uniform as* $\eta \to -1$. *The frustum parameters are* $n_e = 1$, $f_e = 1024$, *and* $\theta = 30°$.

Figure 5.27 shows the error over $\gamma \in [0°, 90°]$ using various shaping functions for the warping parameter. When each parameterization uses its optimal parameter without any shaping, the error for $\gamma < \theta$ is high. The LiSPSM algorithm uses a $1/\sin\gamma$ factor to modulate the optimal parameter. The figure shows that the effect of this shaping function is relatively minor for $\gamma \in [\theta, 90°]$, which is desirable for keeping the error low. But for $\gamma \in [\theta, 90°]$, the falloff is not fast enough to escape the increase in error in $\widetilde{\mathrm{M}}_{p,s}$ for $\eta \neq -1$. $\widetilde{M}_{\log,s}$ has even higher error because $\eta_{\log}$ is further from -1 than η_p and thus takes even longer to transition. If we compute the optimal parameter that minimizes S, we see that for $\gamma \in [\theta, 90°]$, η is the same as with no shaping at all. At $\gamma = \theta$, η suddenly jumps to 0 and then decreases almost linearly to -1 as γ approaches a point somewhere between 0 and θ. The optimal S_p and S_{lp} never rise above S_{un}. One problem with the S-optimal warping parameter is that S can change very rapidly near $\gamma = \theta$.

Computing the equations for the parameter than minimizes S_p and S_{lp} is somewhat involved. Instead we propose the following simple shaping function:

$$\eta = \begin{cases} 0 & \gamma < \gamma_a \\ -1 + (\eta_b + 1)\left(1 + \cos\left(90\frac{\gamma - \gamma_a}{\gamma_b - \gamma_a}\right)\right) & \gamma_a < \gamma < \gamma_b \\ \eta_b + (\eta_c - \eta_b)\sin\left(90\frac{\gamma - \gamma_b}{\gamma_c - \gamma_b}\right) & \gamma_b < \gamma < \gamma_c \\ \eta_c & \gamma < \gamma_c \end{cases} \tag{5.76}$$

Equation 5.76 can nearly replicate the curve for the S-optimal warping parameter, but also gives the user better control over the shape of the error. This function provides a linear transition from -1 to η_b on the interval $\gamma \in [\gamma_a, \gamma_b]$. From there it provides a smooth transition to η_c over $\gamma \in [\gamma_b, \gamma_c]$. Based on observations of the optimal S curves over the typical range of θ and f_e/n_e and some experimentation, we choose the following values for perspective and logarithmic perspective parameterizations:

	γ_a	γ_b	γ_c	η_b	η_c
$\mathbf{F}_p$	$\frac{\theta}{3}$	θ	$\theta + 0.3(90 - \theta)$	-0.2	0
$\mathbf{F}_{lp}$	$\frac{\theta}{2}$	θ	$\theta + 0.3(90 - \theta)$	-0.2	1

We choose $\gamma_b = \theta$ because this gives us precise control over η at the point where the error functions begin to change rapidly. We choose a value for γ_c that ensures that the error is kept low for $\gamma > \theta$ while extending the transition to the high error near $\gamma = \theta$ so that it is not too abrupt. η_b should be 0 to replicate the behavior of the S-optimal curve. However, this causes the value of $\widetilde{\mathrm{M}}_{p,s}$ at $v = 1$ to rise to the same value as at $v = v_{s2}$. By decreasing η_b a small amount, it is possible to mitigate this effect without affecting S too much. The behavior of S using this warping parameter shaping function is fairly consistent over varying θ and f_e/n_e.

Our shaping function is based on the optimal warping parameter value using S as the error metric. S is only one possible error metric and may not be suited for all applications. Our shaping function may not be able to provide a good fit for other metrics. For this reason we would like to create a more flexible system that would allow a user to build a shaping function interactively using something like splines. By presenting users with graphs similar to those used in this chapter, they could interactively generate a shaping function that meets their particular needs.

5.2.2 Error over all light directions

Figure 5.28 shows the error distribution over varying light directions for frustum parameterizations. A standard shadow map using a uniform parameterization shows little variation. The variation using $\mathbf{F}_p$ is less than that of $\mathbf{F}_{lp}$. The graph of the error for $\mathbf{F}_p$ relative to $\mathbf{F}_{lp}$ shows that while the error is roughly the same for $\gamma < 30°$, the error of $\mathbf{F}_p$ is much higher for $\gamma > 30°$. The variation in error is greatly reduced for low f_e/n_e ratios.

5.2.2.1 Using pseudo-γ for $\mathbf{F}_{lp}$

The error distribution for $\mathbf{F}_{lp}$ shows small islands of higher error near $\gamma = 30°$. The error is mostly concentrated in R_s. When the light direction crosses through these regions, the abrupt change in shadow quality can be quite noticeable (see Figure 5.29). Figure 5.30 shows how the problem occurs. Because of the way the view frustum is rotated for this light direction, vertices from the far face have dipped below the eye location. This situation is similar to the $\gamma < \theta$ case illustrated in Figure 5.24. According to our analysis, the warping should be

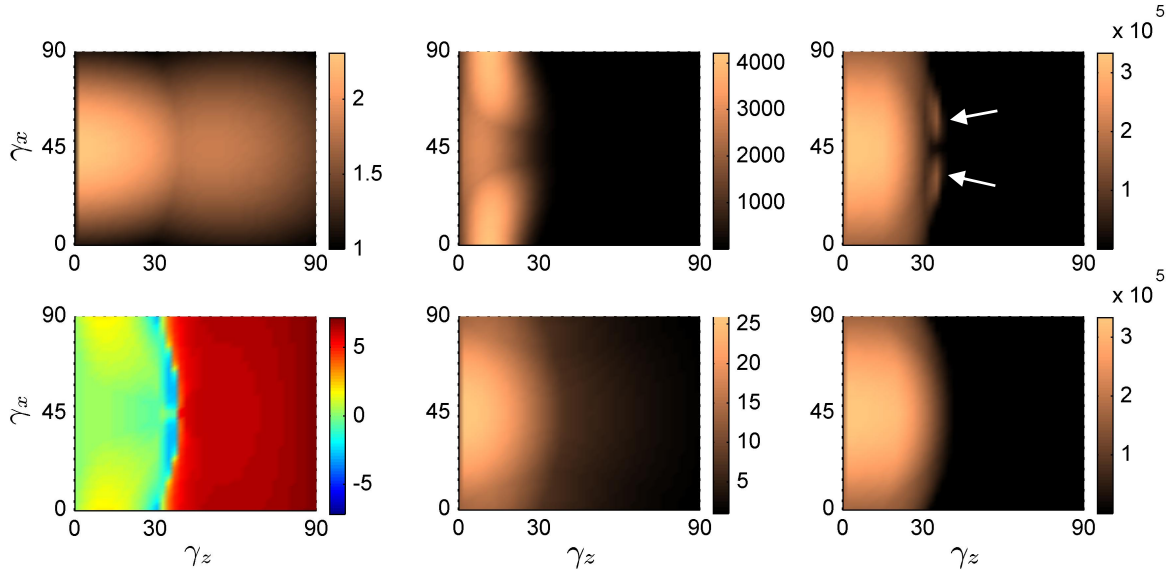

Figure 5.28: ***Error distribution for frustum parameterizations over varying light directions.*** *These graphs show the error in terms of the storage factor for a frustum parameterized with* $\mathbf{F}_{un}$ (top-left), $\mathbf{F}_p$ (top-middle and bottom-middle) *and* $\mathbf{F}_{lp}$ (top-right and bottom-right). γ_z *is the angle between the light direction and the view direction and* γ_x *is the azimuthal angle measured from the* x*-axis in eye space. The minimum error value is normalized to* 1 *to highlight the variation in error. We have used our shaping function for the warping parameter for* $\mathbf{F}_p$ *and* $\mathbf{F}_{lp}$*. The frustum parameters are* $n_e = 1$, $f_e = 1000$, *and* $\theta = 30°$, *except for* (bottom-middle) *where* $f_e = 10$. (Bottom-left) *shows the error of* $\mathbf{F}_p$ *relative to* $\mathbf{F}_{lp}$ *using a* $\log_2$ *scale.* (Top-right) *Arrows highlight small islands of relatively higher error.* (Bottom-right) *Using the pseudo-*γ, γ_p, *in place of* γ_z *removes these islands but causes higher error around* $\gamma_z = 30°$.

relatively weak for this case, resulting in a nearly uniform parameterization. However, γ is actually greater than θ, so the warping is still fairly strong, leading to high error. One way to avoid this problem is to define a pseudo-γ, γ_p, that behaves like γ for light directions in the yz plane, but is defined in terms of the position of the eye relative to the extents of the far face. From Figure 5.30 we can see that $d_0 = f_e \tan\gamma$ and $d_1 = f_e \tan\theta$. By the properties of similar triangles, $d_0'/d_1' = d_0/d_1$. From this we can conclude that:

$$\gamma = \tan^{-1}\left(\frac{d_0'}{d_1'}\tan\theta\right). \tag{5.77}$$

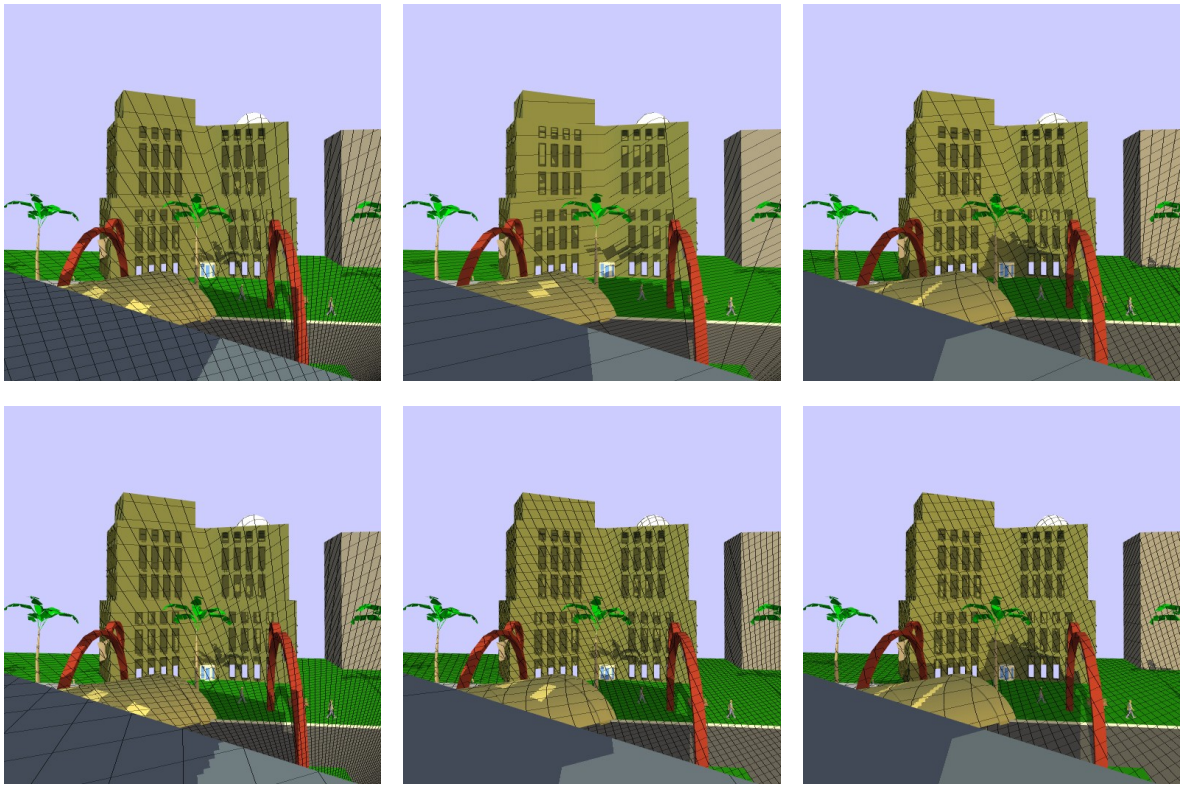

Figure 5.29: *Problem case for* $\mathbf{F}_{lp}$. *(Top row) A small change in γ causes a large change in error. Grid lines for every 10 shadow map texels are shown. On the left, the error is fairly evenly distributed across the view frustum, as can be seen on the ledge in the foreground and the building in the background. In the middle, the error becomes very large in the s direction for $\gamma \approx \theta$. This is because the warping is too strong. Resolution redistribution has been disabled in these images to highlight this effect. On the right, the warping quickly falls off with a few more degrees of rotation, restoring the high quality in the background and causing higher error in the foreground. (Bottom row) Using pseudo-γ instead of γ with our warping parameter shaping function removes this problem at the expense of higher error in the foreground for $\gamma \approx \theta$ caused by a faster falloff in the warping.*

If we define $d'_0 = v_e - v_{f0}$ and $d'_1 = (v_{f1} - v_{f0})/2$, where v_{f0} and v_{f1} define the extents of the far face projected onto the light image plane, we can compute our pseudo-γ as:

$$\gamma_p = \tan^{-1}\left(\frac{|v_e - v_{f0}|}{v_{f1} - v_{f0}} \tan\theta\right). \tag{5.78}$$

The absolute value is needed to make γ_p valid over all light directions. If the light direction lies in the yz plane, then $\gamma_p = \gamma$. Otherwise $\gamma_p > \gamma$ and using γ_p in place of γ in the warping parameter shaping function causes the transition to the uniform parameterization to occur slightly sooner, thus eliminating the islands of higher error (see Figure 5.28 and Figure 5.29).

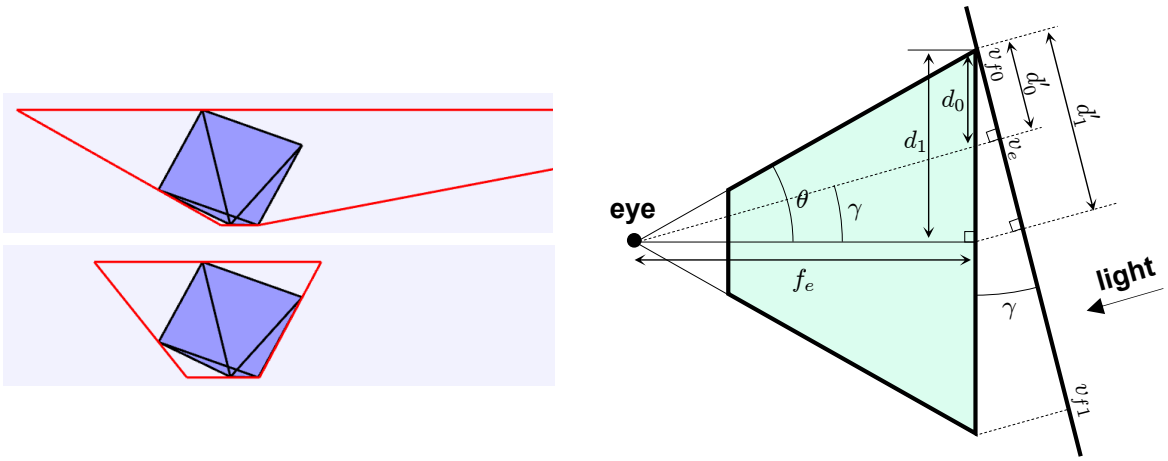

Figure 5.30: ***Frustum in problem case for*** $\mathbf{F}_{lp}$**.** *(Left) The view frustum as seen from a directional light. The warping is too strong in the top image leading to high error. The bottom image uses a pseudo-*γ *to compute the warping parameter value. (Right) The view frustum for a directional light in the yz plane. These quantities are used to compute the pseudo-*γ *in terms of the position on the light image plane of the eye point* v_e *relative to the extents of the far face* v_{f0} *and* v_{f1} *rather than the actual angle between the light and view directions* γ*.*

5.2.3 z-partitioning

All global warping methods have high error for $\gamma \in [0, \theta]$. z-partitioning can be used to dramatically reduce this error. z-partitioning also provides substantial error reductions for $\gamma \in [\theta, 90°]$. In the next section we present a more detailed analysis of the effects of z-partitioning when we compare it to face partitioning.

5.3 Comparisons

In this section, we present empirical results for LogPSMs obtained using a simulator for logarithmic rasterization (see Chapter 6 for details). We also perform several comparisons between several different shadow map algorithms. We use the following abbreviations to classify the different algorithms:

- P: Perspective warping. Unless otherwise indicated, we use the LiSPSM parameter and our new warping parameter shaping function.
- Po: Perspective warping with the original $1/\sin\gamma$ falloff used by LiSPSMs.
- LogP: Logarithmic + perspective warping.
- ZPk: z-partitioning using k partitions. We use only the $\mathsf{ZP}_{\mathsf{log}}$ partitioning scheme in this section.
- FP: Face partitioning.
- FPc: Face partitioning with coordinate frame adjustment to handle shearing.
- FPcs: Face partitioning with coordinate frame adjustment and face splitting.

Partitioning schemes can be combined with warping, e.g. ZP5 + P stands for z-partitioning with 5 partitions and perspective warping, and FPc + LogP is face partitioning with coordinate frame adjustment and the logarithmic+perspective warping. FPc + ZP2 + P means face partition using perspective warping with z-partitioning applied to the side faces. When ZPk appears alone without a warping algorithm, a uniform parameterization is used.

5.3.1 Experimental results

Showing the quality of one shadow map algorithm relative to another from images alone is often difficult. The regions of maximum error can differ between algorithms. To see the maximum error, surfaces containing shadow edges must lie in these regions. In order to more easily visualize the aliasing error, we project texel grid lines from the shadow map onto the scene. In addition, we generate color coded images of aliasing error. We use the area of

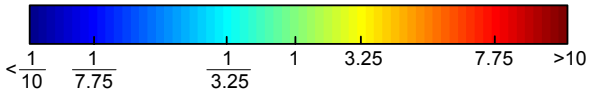

Figure 5.31: ***Color mapping for error comparison images.***

the projected texels in the image (measured in pixels) as the aliasing metric. The area is a measure that combines the error in both shadow map directions, while the projected texel grid helps convey the shape of the projections. The color mapping used for the color coded images is shown in Figure 5.31 with red representing high error, green representing no error, and blue representing over sampling.

Figure 5.32 shows a comparison between various shadow map algorithms. The aliasing is extremely high near the viewer for the standard shadow map, but much less in the distance. The $\mathsf{FPc} + \mathsf{P}$ algorithm is comparable to the perspective-warped cubemap algorithm of Kozlov (2004), except that the LiSPSM parameter is used for warping instead of the PSM parameter. The LiSPSM parameter gives a more uniform distribution of error between the s and t shadow map directions. $\mathsf{FPcs} + \mathsf{LogP}$ has lower error than $\mathsf{FPcs} + \mathsf{P}$ due to the better parameterization. $\mathsf{ZP5} + \mathsf{P}$ is similar to cascaded shadow maps (Engel, 2007), but adds warping for further error reductions. $\mathsf{ZP5} + \mathsf{P}$ always renders 5 shadow maps while $\mathsf{FPcs} + \mathsf{LogP}$ can render anywhere from 1 to 7. For this view, $\mathsf{FPcs} + \mathsf{LogP}$ renders only 3. $\mathsf{FPcs} + \mathsf{LogP}$ has the most even distribution of error.

Figure 5.33 shows a dueling frustum situation which is especially difficult for single shadow map algorithms to handle. Here $\mathsf{FPc} + \mathsf{LogP}$ produces less error than $\mathsf{ZP5} + \mathsf{P}$. The portion of the image around the light direction is over sampled for surfaces far from the viewer.

Figure 5.34 is a comparison using a single shadow map. The light is nearly in the optimal position for both P and LogP. When the light is behind or in front of the viewer, both of these algorithms degenerate to a standard shadow map.

Figure 5.35 shows $\mathsf{FP} + \mathsf{LogP}$ used with point lights. We compare the algorithm against Kozlov's perspective-warped cube map (Kozlov, 2004). The LogP parameterization also provides lower error for point lights.

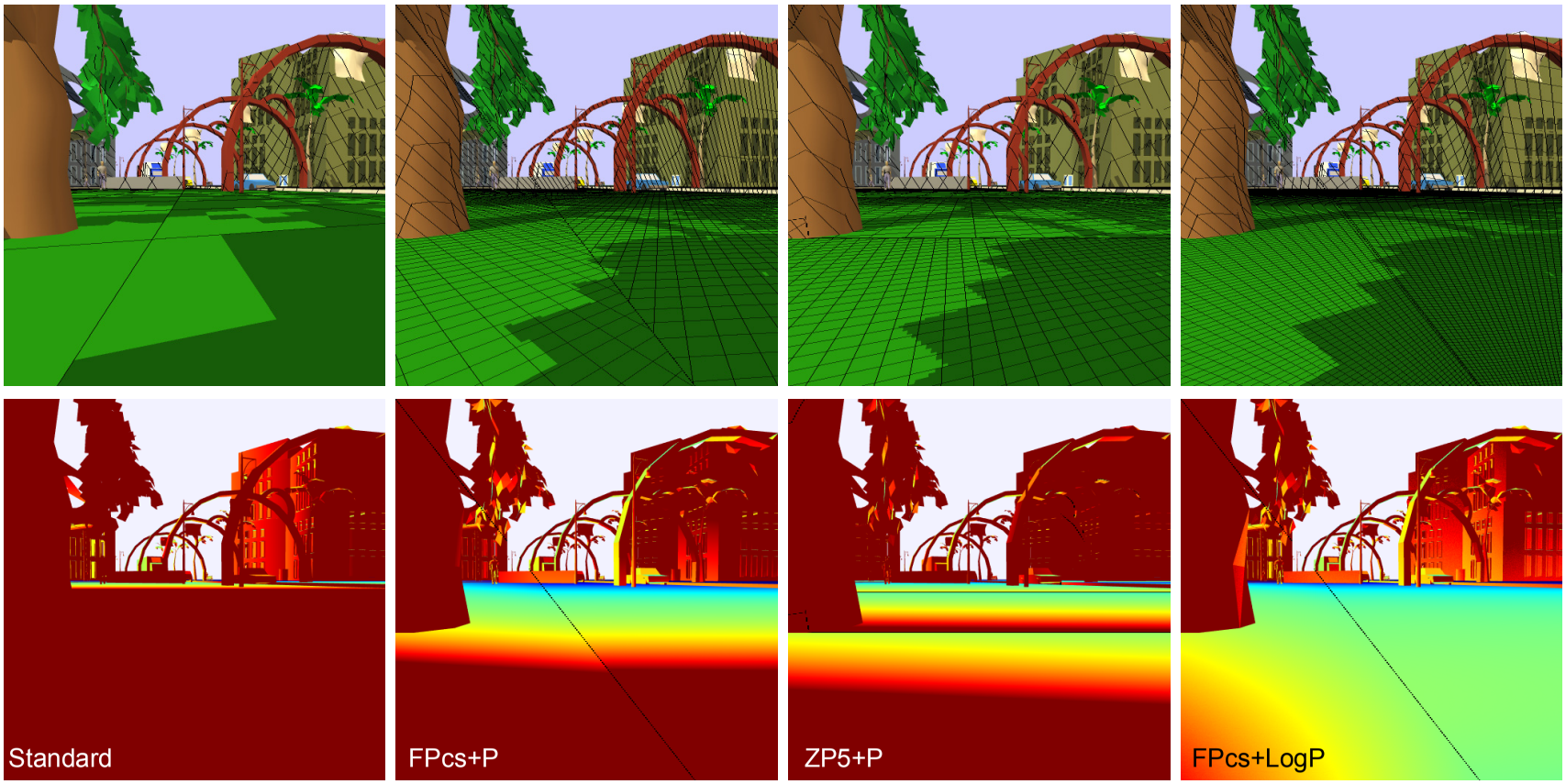

Figure 5.32: ***Comparison of various algorithms.*** *The viewer is positioned below a tree in a town scene.* (Top row) *Grid lines for every 5 texels projected on the scene.* (Bottom row) *Color coding of aliasing error in terms of projected shadow map texel area in pixels.* FPcs + P *is a combination of face partitioning and perspective warping.* ZP5 + P *uses 5 z-partitions combined with perspective warping.* FPcs + LogP *is a combination of face partitioning with a logarithmic perspective warping. The face partitioning algorithms use only* 3 *shadow maps for this view. Pixels are black at partition boundaries where derivatives are not well-defined. Both the image and* total *shadow map resolution are* 512×512*. The LogPSM produces lower, more evenly distributed error.* ($f/n = 1000$, $\theta = 30°$)

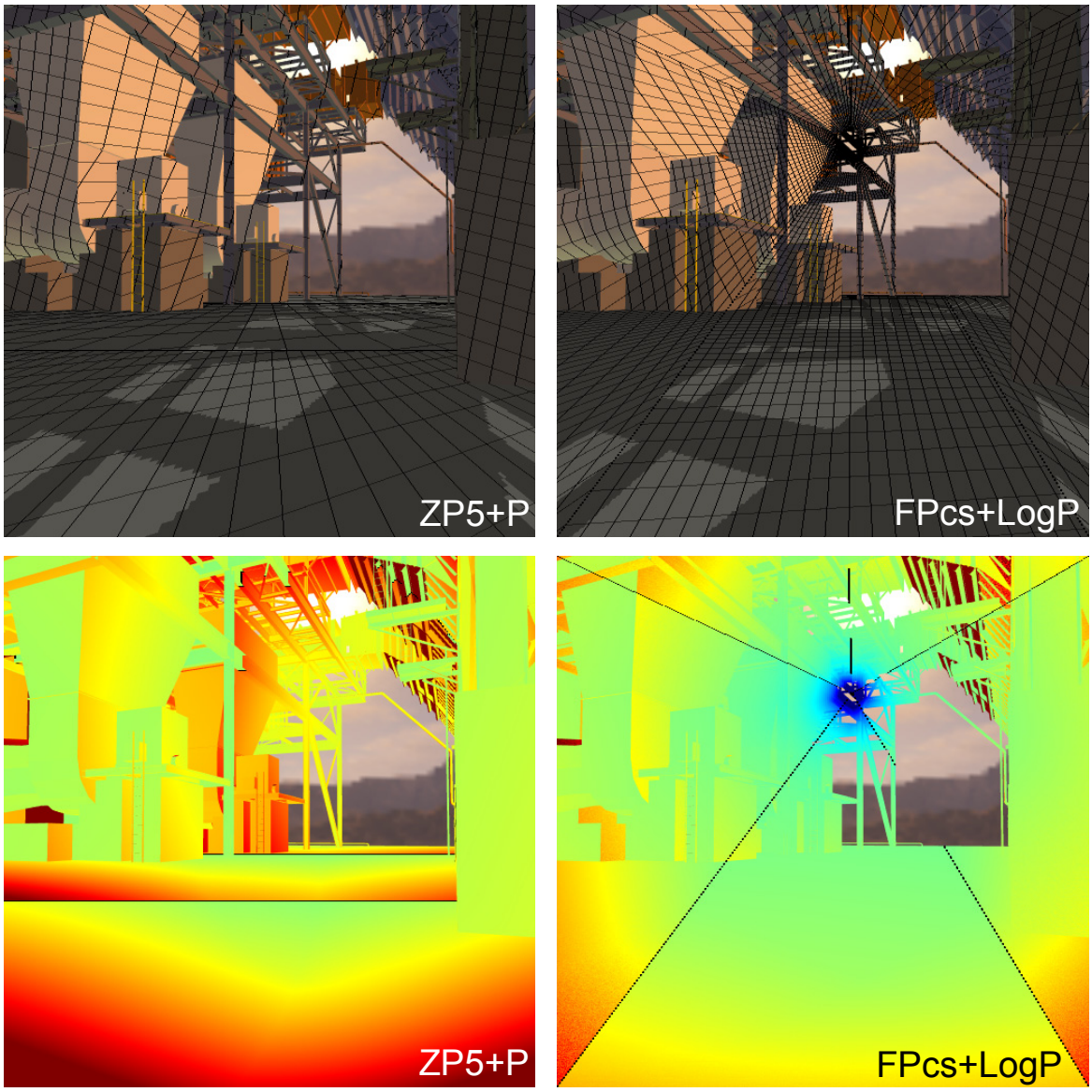

Figure 5.33: ***Comparison in a power plant model.*** *The light is placed almost directly in front of the viewer. This is the dueling frustum case that is difficult for single shadow map algorithms to handle. Both* FP *and* ZP *algorithms can handle this situation well, though* FPcs + LogP *produces less error than* ZP5 + P*. Both the image and total shadow map resolution are* 512×512*.* $(f/n = 1000,\ \theta = 30°)$

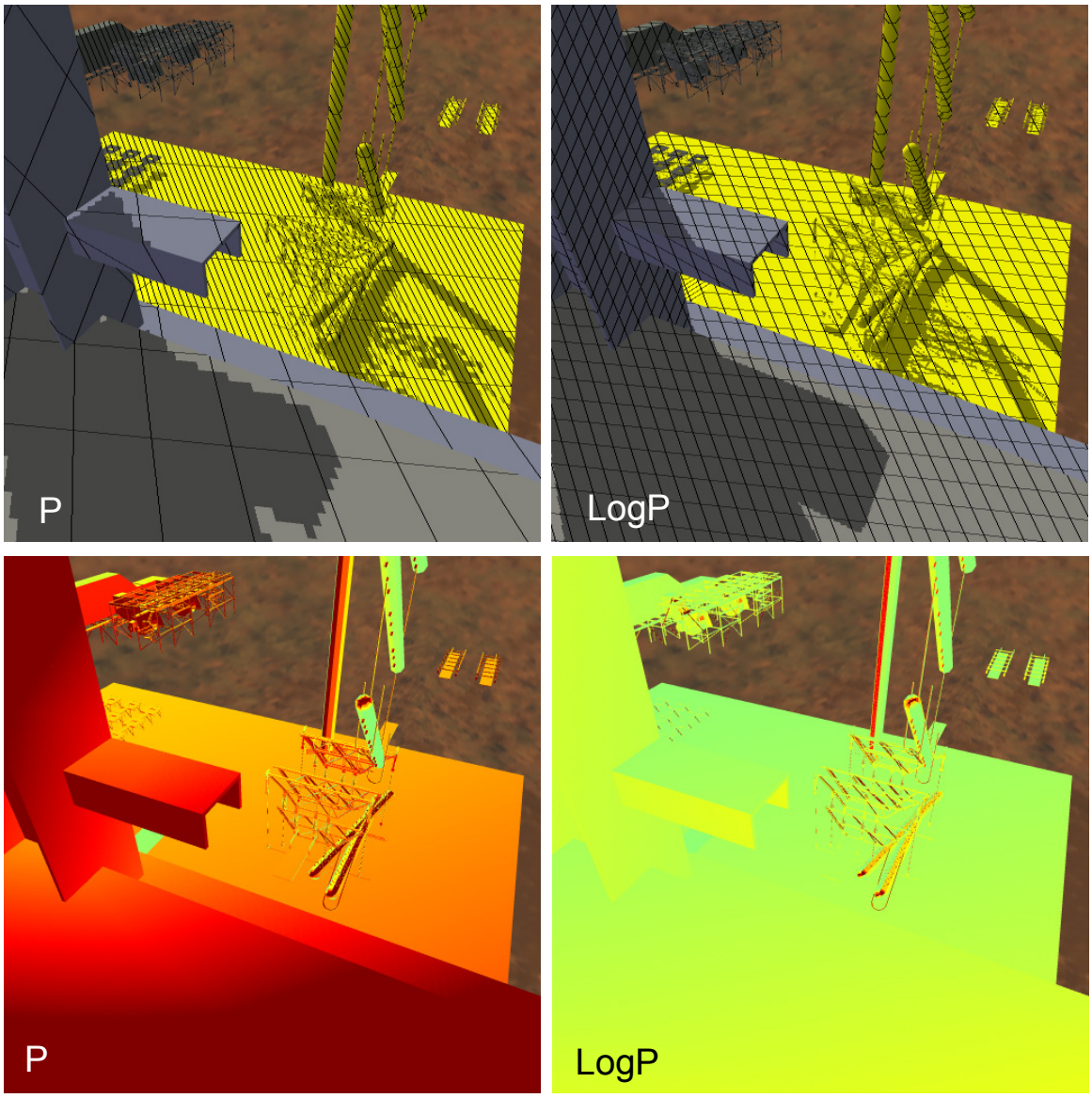

Figure 5.34: ***Comparison with single shadow map in a power plant model.*** *Here we compare the* P *and* LogP *parameterizations using a single shadow map. The image resolution is* 512×512 *and the shadow map resolution is* 1024×1024*. Grid lines are shown for every* 10 *texels. (*$f/n = 500$*,* $\theta = 30°$*)*

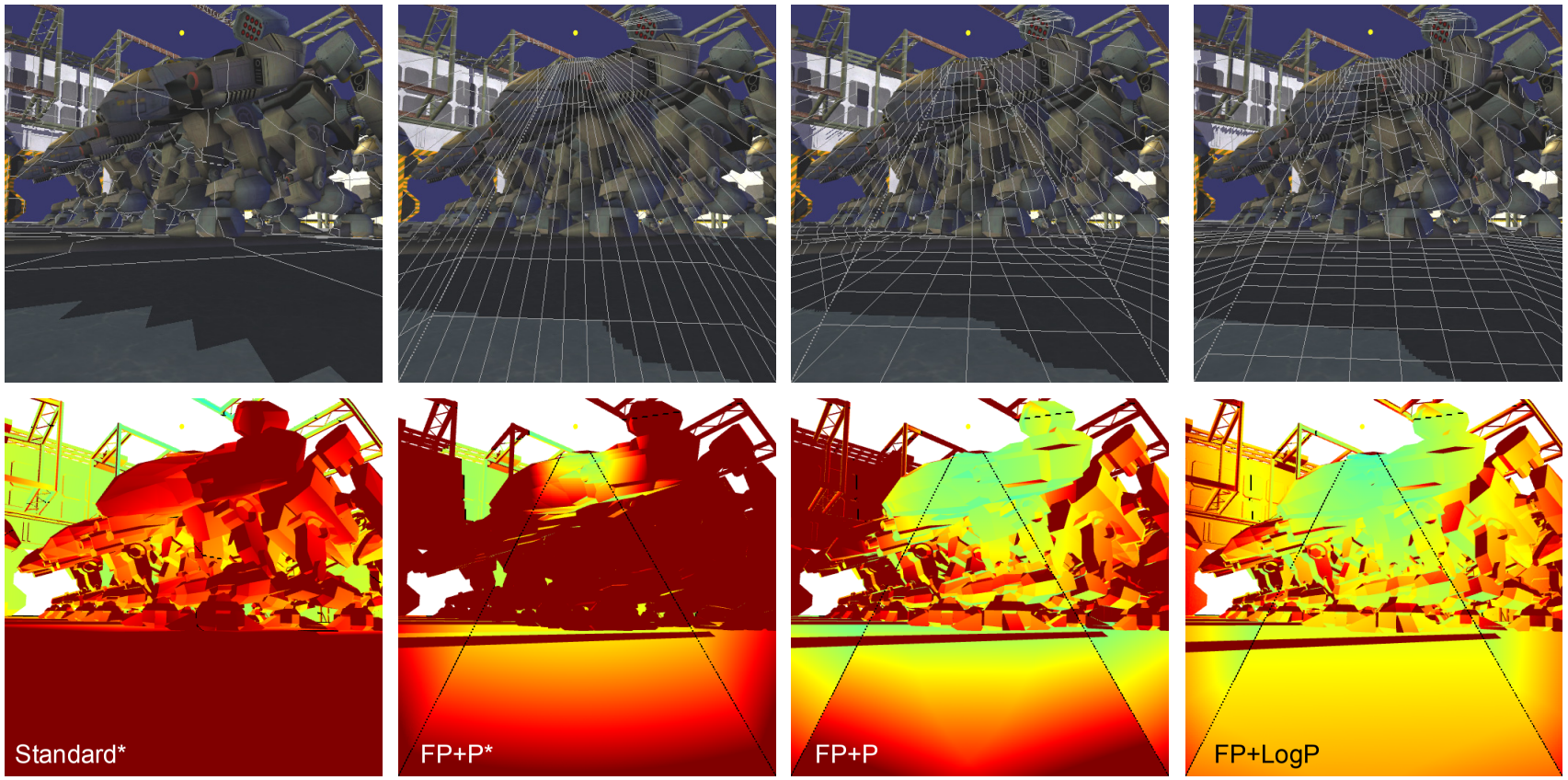

Figure 5.35: ***Comparison with point lights.*** *This is a night-time scene of robots in a hangar with a point light. All algorithms use a cube map with a* total *resolution of* 1024×1024*. The images have a resolution of* 512×512*. The algorithms marked with* $*$ *do not use resolution redistribution. A standard cube map for this view has high error near the viewer. Perspective warping (*$\mathsf{FP} + \mathsf{P}^*$*) (Kozlov, 2004) reduces error near the viewer, but increases error on the back wall. Here the shadows are extremely stretched, except for one portion corresponding to an end face partition which has low error. Adding resolution redistribution (*$\mathsf{FP} + \mathsf{P}$*) moves some resolution from the end faces to the side faces.* $\mathsf{FP} + \mathsf{LogP}$ *provides higher quality both near the viewer and in the distance. Grid lines are shown for every* 20 *texels. LogPSMs provide significantly lower maximum error and the error is more evenly distributed. (*$f/n = 300$*,* $\theta = 30°$*)*

Algorithm	Min S	Max S	Max S/Min S	Mean	Rel. mean	Mean # SMs
Standard	8.65×10^5	2.00×10^6	2.31	1.32×10^6	1.21×10^5	1
P	865	3.65×10^6	4.22×10^3	2.53×10^5	2.32×10^4	1
Po	865	2.00×10^6	2.30×10^3	1.59×10^5	1.45×10^4	1
Log	5.98	2.00×10^6	3.33×10^5	1.85×10^5	1.69×10^4	1
FP+P	865	2000	2.31	1490	136	3.6
FPc+P	865	2980	3.45	1840	168	3.6
FPcs+P	865	3170	3.66	1880	171	3.8
ZP5	51.4	158	3.08	94.1	8.60	5
ZP5+P	12.9	159	12.3	53.7	4.91	5
ZP5+Log	5.98	158	26.5	46.3	4.23	5
ZP7	27.4	101	3.68	56.6	5.17	7
ZP7+P	10.2	101	9.87	42.2	3.85	7
ZP7+Log	5.98	101	16.8	38.4	3.51	7
FP+Log	5.98	14.8	2.48	10.9	1	3.6
FPc+Log	5.98	25.4	4.24	15.4	1.40	3.6
FPcs+Log	5.98	27.7	4.62	15.6	1.43	3.8

Table 5.4: ***Storage factor over all light directions.*** *Since the storage factor is greatest as the light moves toward infinity, we used a directional light in order to obtain an upper bound on the storage factor. This table summarizes statistics for various combinations of perspective warping (*P*), logarithmic+perspective warping (*LogP*), face partitioning (*FP*), and z-partitioning (*ZP*). The second to last column shows the mean storage factor relative to* FP + LogP*. The last column shows the mean number of shadow maps used. Over all light directions LogPSMs have the lowest minimum storage factor.* FP + Log *and its variations also have the lowest maximum, and mean storage factor. ($f/n = 1000, \theta = 30°$). The $1/\cos^3\theta = 1.5$ term has been factored out of the values reported here.*

5.3.2 Analysis

Table 5.4 shows the variation in perspective aliasing error measured by the storage factor over all light directions for various algorithms. Standard shadow maps have the highest error, but over all light directions the variation in the error is fairly small. The single shadow map warping algorithms P, Po, and LogP provide lower error for overhead views, but must degenerate to standard shadow maps when the light moves behind or in front of the viewer. This leads to a huge variation in error that makes these algorithms more difficult to use. The table shows that in contrast to Po, our improved shaping function for the warping parameter keeps the maximum error of P below that of a standard shadow map. Even though LogP has a much lower minimum error than P, it ramps off to a uniform parameterization slightly faster than P and the extremely high error of the uniform parameterization dominates the mean. However, it can be seen from Figure 5.36 that LogP provides significant improvement

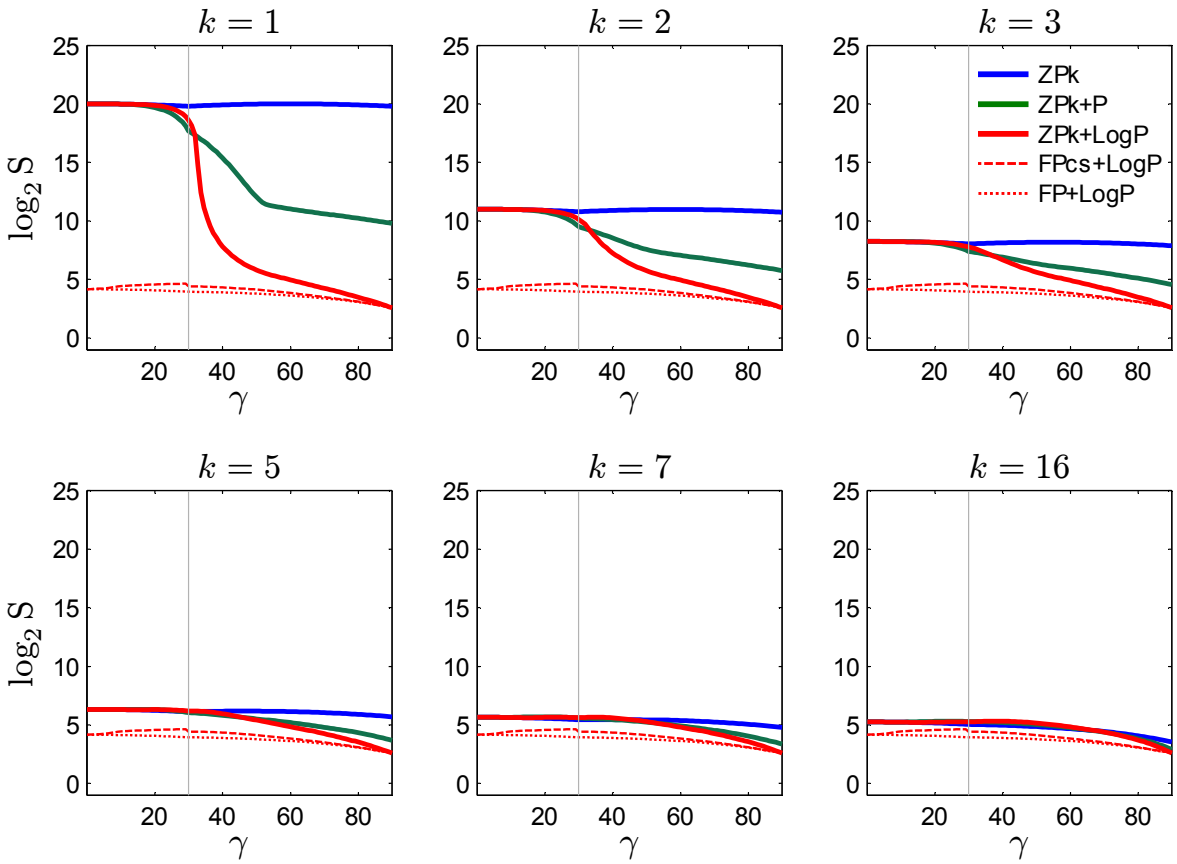

Figure 5.36: ***z-partitioning using various parameterizations.*** *z-partitioning leads to significant error reductions but requires many partitions to converge to the same error as a face partitioning scheme that uses a logarithmic+perspective parameterization. ($f_e/n_e = 1024$, $\theta = 30°$)*

over P for almost the entire range of $\gamma \in [\theta, 90°]$.

Face partitioning leads to much lower variation in error over all light directions. Coordinate frame adjustment and face splitting reduces shearing error not accounted for by the storage factor, which causes a slight increase in the storage factor but an overall decrease in actual error. The FP $*$ +LogP algorithms have much lower error than the FP $*$ +P algorithms due to the better parameterization.

As with a single shadow map, z-partitioning with a uniform parameterization has the least variation in error over all light directions. Adding warping reduces the error for $\gamma \in [\theta, 90°]$. The minimum error for ZPk + LogP, which occurs for an overhead directional light, is the same for any k. With this light position, increasing the number z-partitions has no effect on the parameterization. This is not the case for uniform and perspective parameterizations.

Figure 5.36 shows that for other light positions, increasing k produces drastic reductions in error that then trail off. The benefit of warping is also reduced. For comparison, the error for $\mathsf{FP} + \mathsf{Log}$ and $\mathsf{FPcs} + \mathsf{LogP}$ are also shown in Figure 5.36. A small rise in $\mathsf{FPcs} + \mathsf{LogP}$ can be seen at $\gamma = \theta$ where a new face partition appears and is split. We have shown the error for the $\mathsf{ZP}k$ algorithms with $k = 5$ and $k = 7$ because these are the maximum number of shadow maps required for the FP and FPc algorithms and the FPcs algorithm, respectively. As the number of z-partitions increases, the error begins to approach that of $\mathsf{FP} + \mathsf{LogP}$. On average, however, the number of shadow maps used by FP, and FPc is only 3.6, and for FPcs the average number is only 3.8.

The statistics reported in Table 5.4 show scene-independent maximum perspective aliasing error over all light directions. To get an idea of how well this correlates with the actual aliasing error in the image for a real scene we uniformly sampled the hemisphere of light directions above the scene and created a histogram of the error in the image. Figure 5.37 shows the cumulative distribution of the error. The value at a point x on the curve is the fraction of pixels with error less than x. The distribution of error depends on the positions of the surfaces within the view frustum. We see that the single shadow map algorithms tend to have higher error spread out over a wider range. When the surfaces are far from the eye, as in the first view shown in Figure 5.37, warping and partitioning of any kind may produce little benefit over standard shadow maps. In fact, the warping algorithm P produces error that is significantly worse than a standard shadow map for this view. When the surfaces are closer to the eye, as in the second view, $\mathsf{FPc} + \mathsf{LogP}$ can produce significantly less error than other parameterizations.

5.3.3 Benefits of LogPSMs

In this analysis we have examined three types of LogPSMs: LogP, $\mathsf{ZP} + \mathsf{LogP}$, and $\mathsf{FP} + \mathsf{LogP}$. We will now discuss the advantages of each algorithm relative to existing algorithms in terms of error reduction and the number of shadow maps. Ideally, we want an algorithm that gives the most error reduction with the fewest number of shadow maps.

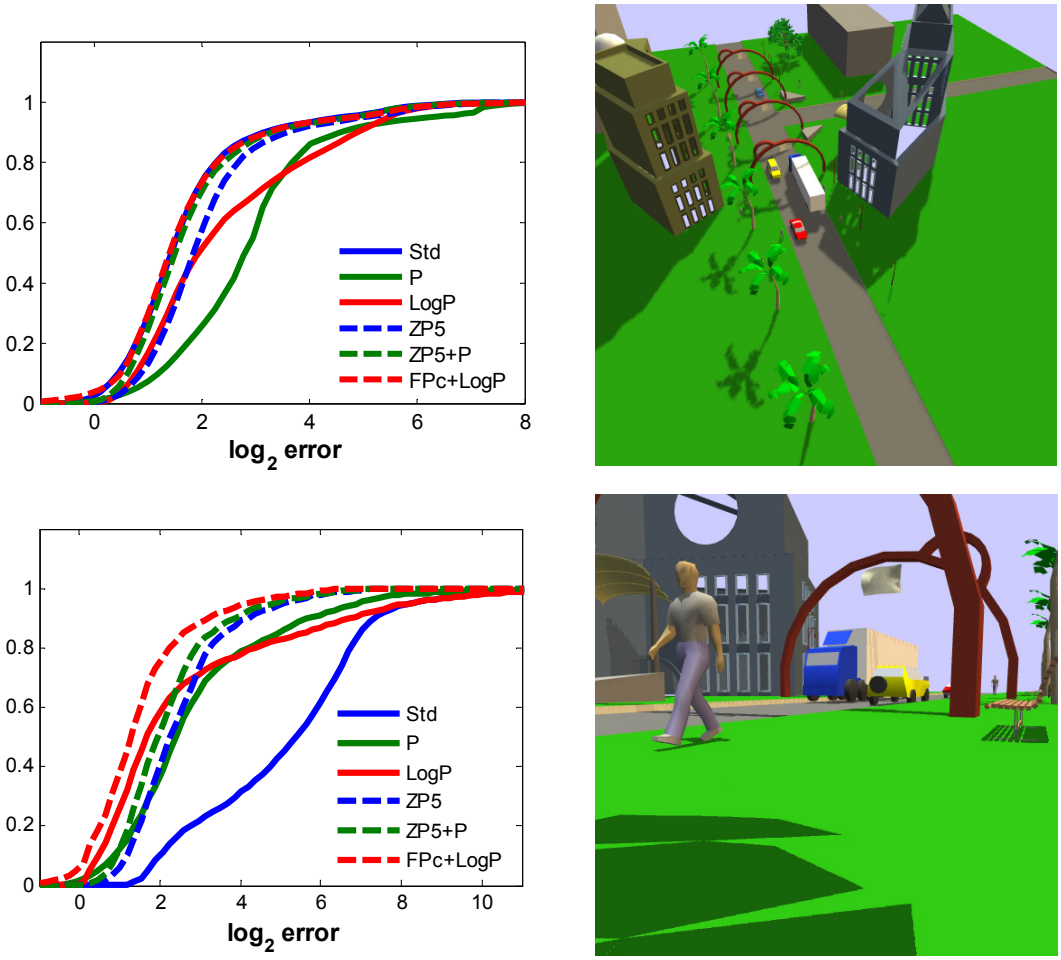

Figure 5.37: ***Cumulative aliasing error distribution for several algorithms over randomly sampled light directions.*** *The graphs on the left were generated for the views on the right. The area of the projected texel was used for the error metric. The actual benefit of warping and partitioning methods depends on the location and orientation of surfaces within the view frustum.*

LogP. The LogP algorithm provides the greatest error reductions of any technique with the fewest shadow maps (only 1), but only for a limited range of light positions. For high f_e/n_e ratios, the maximum error of LogP is significantly lower than any other single shadow map warping scheme. For light directions nearly perpendicular to the view direction, a ZP scheme requires a large number partitions to approach the same levels of error. The LogP algorithm is most useful applications where the light direction does not approach the view direction (e.g. shadows from the sun around mid-day in a driving simulator). But over all light positions, LogP shares the same high maximum error as other single shadow map algorithms.

ZP + LogP. The high error of LogP for $\gamma < \theta$ can be reduced dramatically with a few z-partitions. As the number of z-partitions is increased, however, the relative benefit of ZP + LogP versus ZP or ZP + P decreases. Thus ZP + LogP is best for applications that can only afford to render 2–4 shadow maps. Though FP + LogP can deliver lower error using an average number of shadow maps in the same range, ZP + LogP is much simpler to implement.

FP + LogP. Compared to other algorithms using the same number of shadow maps, FP + LogP produces the least error over all light directions. FP + LogP has lower error than a ZP scheme using the same maximum number of shadow of maps (5 or 7 depending on whether face splitting is used), but the error is not dramatically lower. The advantage of LogP is somewhat more significant if we compare to a ZP scheme with the same average number of shadow maps (3.6–3.8). The real advantage of FP + LogP over the ZP algorithms is with omnidirectional point lights. Omnidirectional lights require multiple shadow maps to cover all light directions, e.g. a cube map. Adding z-partitioning to each cube map face leads to an explosion in the number of shadow maps. However, FP + LogP requires no more shadow maps for omnidirectional point lights than it does for directional lights.

5.3.4 Limitations of LogPSMs

LogPSMs inherit some of the limitations of other sample redistribution techniques. Warping creates a non-uniform distribution of depth values, which can make it more difficult to select a suitable constant bias. This is less of an issue for slope scaled bias. Warping also tends to increase the instability of texture coordinate derivative calculations on surfaces nearly parallel to the light (some noise can be seen in Figure 5.32). Filtering can also become more complicated due to warping because the filter kernel must take into account the distortion introduced by warping. Handling the shearing artifacts created by face partitioning requires up to 2 extra shadow maps and added complexity. Face partitioning also uses a variable number of shadow maps, which can require more implementation overhead. Most importantly, logarithmic rasterization is not available on current GPUs. It can be simulated with a fragment program, but it is considerably slower than linear rasterization because the required depth writes disable important GPU optimizations.

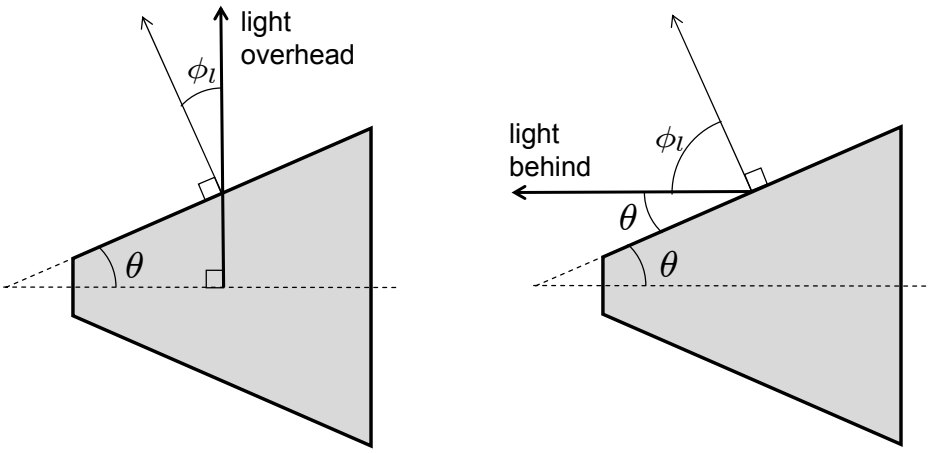

Figure 5.38: ***Light angle relative to side face.*** *For the light overhead $\phi_l = \theta$. For the light behind $\phi_l = 90° - \theta$.*

5.3.5 Lowest error for perspective warping: z-partitioning for faces or frustum?

We now consider how to obtain the lowest error on current hardware without logarithmic rasterization. We consider combinations of perspective warping, face partitioning, and z-partitioning. Face partitioning extends the benefits of warping to all light directions. To get further error reductions, z-partitioning must be applied to the faces. z-partitioning can also be applied directly to the view frustum without using face partitioning. Which approach gives the greatest error reduction for the least number of shadow maps? To analyze the effects of each type of partitioning on aliasing error we consider two light directions relative to the viewer: light overhead ($\gamma = 90°$), and light behind (has the same error as $\gamma = 0°$).

Because z-partitioning splits the view frustum into subfrusta, we will use the same equations for base error distribution that we used in Section 5.2 where we used a light image plane oriented perpendicular to the light direction. We also we moved the $\cos\phi_l$ and $W_e = 2\tan\theta$ terms into $\widetilde{\mathrm{M}}_t^{\mathrm{side}}$. To convert from error quantities in t on side faces to the error into terms of the t direction for frusta we multiply by $\cos\phi_l/(2\tan\theta)$. The base storage factor S can be converted by multiplying by the same factor. After multiplying S by $\cos\phi_l/(2\tan\theta)$, the result differs from S on faces by a factor of $\cos\phi_l\cos^2\theta$. Because we have explicitly computed S for all the parameterizations, we will perform the conversion using this factor. From Figure

5.38 we can see that the $\cos\phi_l$ term relative to the face becomes $\cos\theta$ and $\cos(90° - \theta) = \sin(\theta)$ for the light overhead and behind, respectively.

Light overhead. With the light overhead, only one face is visible to the light. Converting from face error values we get:

$$\mathrm{S}^{\mathrm{over}} = \mathsf{S}^{\mathrm{side}} \cos^3\theta. \tag{5.79}$$

Applying this equation to $\mathsf{S}^{\mathrm{side}}_{un,k}$ (Equation 5.55) we obtain the base storage factor for ZP with k partitions and no warping:

$$\mathrm{S}^{\mathrm{over}}_{\mathsf{ZP}} = k(f_e/n_e)^{1/k} \frac{\left((f_e/n_e)^{1/k} - 1\right)}{2\tan\theta}. \tag{5.80}$$

Likewise, from $\mathsf{S}^{\mathrm{side}}_{p,k}$ (Equation 5.56) we can compute the base storage factor for z-partitioning with perspective warping:

$$\mathrm{S}^{\mathrm{over}}_{\mathsf{ZP+P}} = k \frac{\left((f_e/n_e)^{1/k} - 1\right)}{2\tan\theta}. \tag{5.81}$$

For this light position, $\mathsf{FP} + \mathsf{ZP} + \mathsf{P}$ gives the exact same results as $\mathsf{ZP} + \mathsf{P}$:

$$\mathrm{S}^{\mathrm{over}}_{\mathsf{FP+ZP+P}} = \mathrm{S}^{\mathrm{over}}_{\mathsf{ZP+P}}. \tag{5.82}$$

For a logarithmic perspective parameterization, the storage factor can be computed from $\mathsf{S}^{\mathrm{side}}_{lp}$ (Equation 5.57):

$$\mathrm{S}^{\mathrm{over}}_{\mathsf{LogP}} = \mathrm{S}^{\mathrm{over}}_{\mathsf{FP+LogP}} = \frac{\log(f_e/n_e)}{2\tan\theta}.$$

This provides a baseline for the comparisons.

Light behind. Figure 5.39 shows the view frustum as seen from a light behind the viewer. A ZP scheme cannot use warping because the view frustum is square. A critical resolution shadow map will have the same texel spacing as the image. Therefore the storage factor is

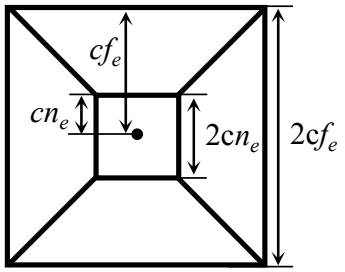

Figure 5.39: ***View frustum as seen by the light behind the viewer.*** *The width of the near and far planes are proportional to n_e and f_e, respectively. ($c = \tan\theta$).*

simply the ratio of the area covered by the shadow map to the area covered by the image:

$$\begin{aligned} \mathrm{S}^{\text{behind}}_{\mathsf{ZP+P}} = \mathrm{S}^{\text{behind}}_{\mathsf{ZP}} &= \frac{(2\tan\theta f_e)^2}{(2\tan\theta n_e)^2} = (f_e/n_e)^2 && \text{with } k = 1 \\ &= k(f_e/n_e)^{2/k} && \text{with } k \geq 1. \end{aligned} \tag{5.83}$$

If we add face partitioning, we can use warping. We use a shadow map for each of the four side faces and the near face. The base storage factor for end faces is 1. The conversion from face error values for the light behind is given by:

$$\mathrm{S}^{\,\text{behind}} = 4\mathsf{S}^{\,\text{side}}\sin\theta\cos^2\theta + 1. \tag{5.84}$$

Applying this equation to $\mathsf{S}^{\text{side}}_{p,k}$ we get:

$$\mathrm{S}^{\text{behind}}_{\mathsf{FP+ZP+P}} = 2k((f_e/n_e)^{1/k} - 1) + 1. \tag{5.85}$$

The storage factor for $\mathsf{FP} + \mathsf{LogP}$ is similar:

$$\mathrm{S}^{\text{behind}}_{\mathsf{FP+LogP}} = 2\log(f_e/n_e) + 1. \tag{5.86}$$

Figure 5.40 shows ZP, $\mathsf{ZP} + \mathsf{P}$, and $\mathsf{FP} + \mathsf{ZP} + \mathsf{P}$ for a varying number of shadow maps. Even ZP without warping does better than $\mathsf{FP} + \mathsf{ZP} + \mathsf{P}$ in the overhead case. Since we must use a fixed number of z-partitions over all light directions in order to avoid popping, the $\mathsf{FP} + \mathsf{ZP} + \mathsf{P}$ scheme gets only one z-partitioning for every four shadow maps. For large

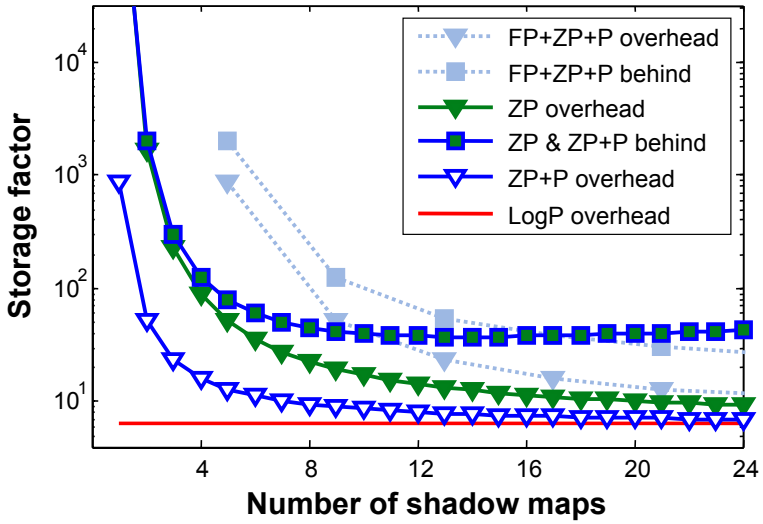

Figure 5.40: ***Storage factor for varying number of shadow maps.*** *The storage factor is shown for the light overhead and behind the viewer for various combinations of z-partitioning (*ZP*), face partitioning (*FP*), and perspective warping (*P*). The storage factor for a single shadow map with a logarithmic perspective parameterization* LogP *is shown as a baseline. (View frustum parameters:* $f_e/n_e = 1000$ *and* $\theta = 30°$*)*

values of (f_e/n_e) we have:

$$\mathrm{S}_{\mathsf{ZP}}^{\mathrm{over}} \sim k(f_e/n_e)^{2/k} \tag{5.87}$$

$$\mathrm{S}_{\mathsf{FP+ZP+P}}^{\mathrm{over}} \sim k(f_e/n_e)^{4/k}, \tag{5.88}$$

where k is the number of shadow maps. The storage factor decreases more rapidly for ZP than for FP + ZP + P as the number of shadow maps increases.

With the light behind, the error for ZP schemes decreases rapidly and then begins to grow slowly. This growth is caused by the significant amount of overlap of the shadow maps that occurs with this light direction. In fact, as $k \to \infty$, $\mathrm{S}_{\mathsf{ZP}}^{\mathrm{behind}}$ diverges to ∞. The FP + ZP + P scheme has very little overlap and as the number of shadow maps increases, it eventually has lower error than the ZP schemes and converges to the same error as $\mathrm{S}_{\mathsf{FP+LogP}}^{\mathrm{behind}}$. Figure 5.41 shows where the cross over occurs between the two schemes.

Based on our analysis, z-partitioning with warping (ZP + P) is the best scheme to use for rendering shadows with a low number of shadow maps in scenes with a high depth range. Most of the benefit comes from the z-partitioning. Over all light directions, the maximum error is

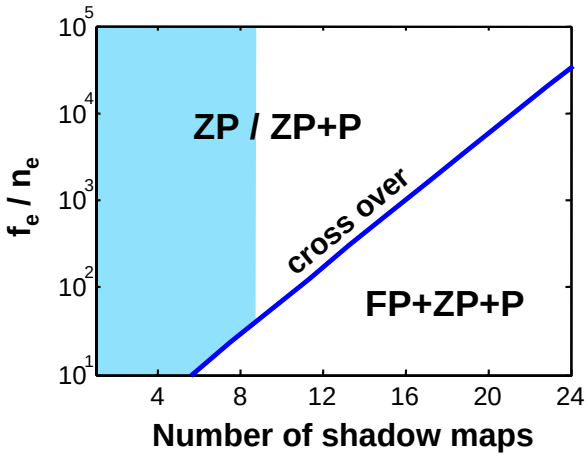

Figure 5.41: ***Lowest error for light behind.*** *This graph shows the parameter values for which the z-partitioning (*ZP*,* ZP + P*) schemes and face partitioning (*FP + ZP + P*) yield the lowest error. z-partitioning is the best scheme for high depth range with few shadow maps (blue region).*

not affected much by the warping because it cannot be used in the high error configurations where the light direction is aligned with the view vector. However, warping does reduce the average maximum error. The effect of warping is diminished with an increased number of partitions because the depth ratio of each partition decreases.

We have analyzed only two light directions. Closed form expressions for the error in the general case are difficult to formulate because of the complex operation of fitting a warping frustum with varying parameters to an arbitrarily oriented view frustum. Figure 5.42 shows the maximum storage factor over all light directions computed by a dense sampling of light directions on the hemisphere. For all combinations of warping, partitioning, and number of partitions, we see that the worst case S is within a factor of 2–3 times of that which we computed analytically for the light behind case.

5.3.6 Approximating logarithmic warping with z-partitioning

The analysis in this chapter can be used to determine how many partitions are needed in order to reduce the maximum error of z-partitioning relative to a LogPSM to within a specified amount. Suppose we want the error of ZPk to be within a factor of ϵ of the error of FP + LogP. The ratio of the errors will change with varying light position. To be conservative we will compare the maximum errors over all light directions from both algorithms . Closed forms

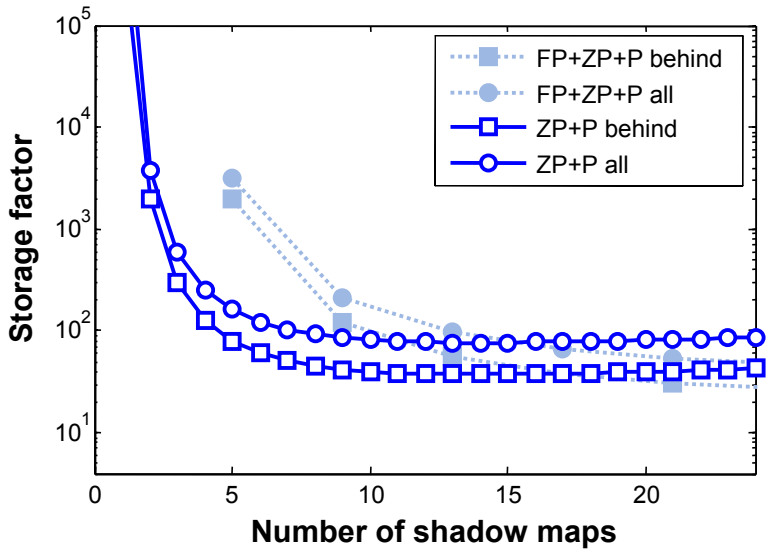

Figure 5.42: ***Storage factor over all light directions.*** *The maximum storage factor over all light directions is compared to storage factor for the light behind the viewer. (View frustum parameters:* $f_e/n_e = 1000$ *and* $\theta = 30°$*)*

for the maximum error are tedious to derive. However, we can use the equations we derived earlier for a light behind the view frustum because the maximum error over all light directions is a constant multiple of the error for the light behind (numerical experiments show that over the range of $f_e/n_e \in [10^1, 10^5]$ the difference between the error over all directions and error for the light behind is a factor of 2 for ZPk, 1 for FP + LogP, 1.5 for FPc + LogP, and 1.7 for FPcs + LogP). Combining Equations 5.83 and 5.86 we get:

$$\epsilon = \frac{\mathrm{S}^{\text{behind}}_{\mathsf{ZP}}}{\mathrm{S}^{\text{behind}}_{\mathsf{FP+LogP}}} = \frac{k(f_e/n_e)^{2/k}}{2\log(f_e/n_e)+1}. \tag{5.89}$$

$\mathrm{S}^{\text{behind}}_{\mathsf{FPc+LogP}}$ and $\mathrm{S}^{\text{behind}}_{\mathsf{FPcs+LogP}}$ are equivalent to $\mathrm{S}^{\text{behind}}_{\mathsf{FP+LogP}}$.

Figure 5.43 shows the relative error ϵ for several values of f_e/n_e and k. As we saw before, the error curves do not converge, but rather they reach a local minimum and then grow slowly. Figure 5.44 shows the minimum k and ϵ for the typical range of f_e/n_e values. We note that over this range with a light behind, z-partitioning alone gets no closer than about a factor of 2 from the error of a LogPSM. To determine the number of partitions needed to obtain an ϵ greater than this minimum value we solve Equation 5.89 for k and obtain the following

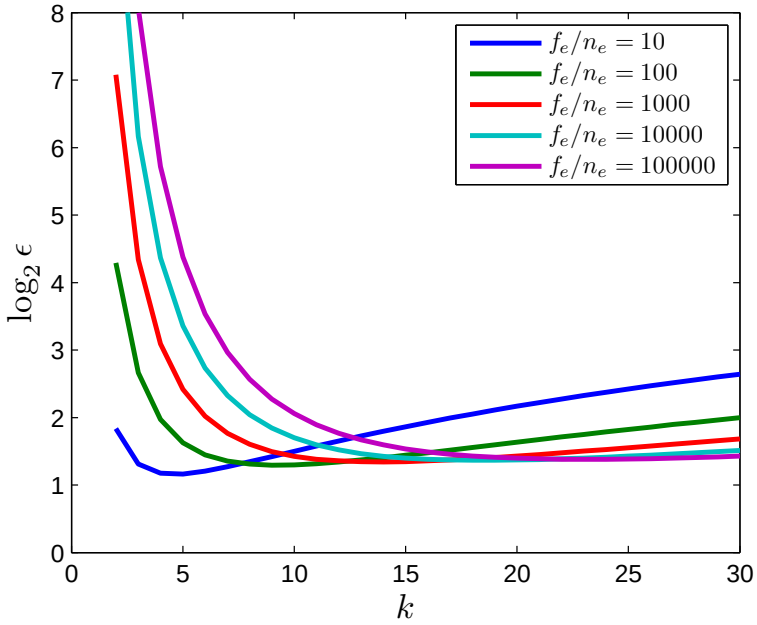

Figure 5.43: ***Error of z-partitioning relative to LogPSM.*** *The relative error ϵ initially decreases with the number of z-partitions k and then slowly rises. The relative error is higher for high f_e/n_e ratios.*

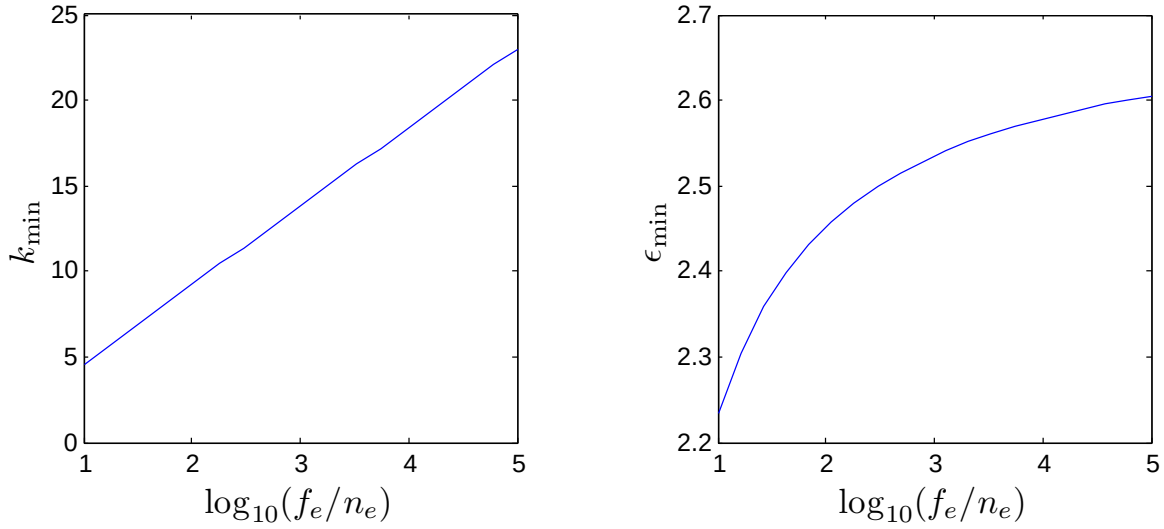

Figure 5.44: ***Minimum relative error of z-partitioning relative to a LogPSM.*** (Left) *The number of partitions $k_{\min}$ at which the minimum relative error $\epsilon_{\min}$ occurs and* (right) *the value of $\epsilon_{\min}$ for varying f_e/n_e ratios.*

analytic solution:

$$k = -\frac{2\log(f_e/n_e)}{W_{-1}\left(\frac{2\log(f_e/n_e)}{\epsilon+2\log(f_e/n_e)}\right)} \tag{5.90}$$

where W is the *Lambert-W* function (also called *product log*) (Corless et al., 1996). W_{-1} is the branch of W that produces real values that are less than -1. The W_{-1} function is not typically included in standard libraries. It can be computed using a series expansion and/or an iterative algorithm. However, a routine to accurately compute W_{-1} is not necessary. Because we are only interested in integer values of k and the possible range of k values for typical f_e/n_e ratios is not especially large, it is feasible to just iterate over successive values of k, stopping when ϵ falls below a specified threshold or starts to increase.

5.4 Summary

In this chapter we have analyzed various combinations of warping and partitioning. We have shown that LogPSMs provide error bounds over all light directions which are the same a near optimal parameterization. We have proposed several ways to improve shadow maps, including the use of a pseudo-near plane, an improved warping parameter shaping function, and a pseudo-γ parameter for single shadow map LogPSMs. We have performed extensive comparisons between the various algorithms to highlight the benefits of LogPSMs and have provided guidelines for obtaining low error shadow maps when a logarithmic perspective parameterization is not available. In the next chapter, we will look at several issues and tradeoffs that arise when implementing these kinds of shadow map algorithms.

CHAPTER 6

Implementation notes

This chapter discusses a number of issues related to the implementation of shadow maps on graphics hardware. We start with an overview of shadow mapping using projective texture mapping. We briefly discuss various alternatives for implementing shadow map algorithms that use multiple partitions. We describe issues related to fitting a light frustum to a partition, dynamic texture allocation for the shadow maps, accelerating shadow map rendering with partition culling, and selecting the appropriate shadow map to query while rendering the image. We also describe the fragment program that we used to simulate logarithmic rasterization. The experimental results and images in this thesis were obtained with this simulator. We also provide code for the grid shader used throughout this thesis, since we have found this to be a useful debugging/visualization tool.

6.1 Shadow mapping using projective texture mapping

Shadow mapping was first implemented using projective texture mapping on graphics hardware by Segal et al. (1992). The main idea is to use a projective matrix to compute texture coordinates from object vertices, which are then used to perform the shadow map query. The texture coordinates can be computed in a vertex program by multiplying the world space vertex coordinate by the matrix $\mathbf{M}_l$, which is the concatenation of the light's view matrix $\mathbf{V}_l$ and projection matrix $\mathbf{P}_l$ used to render the shadow map. The homogeneous 3D texture coordinates computed at the vertices are linearly interpolated over the primitives. The texture coordinates are then used to query the shadow map. The texture look-up into the shadow map typically performs the division by the homogeneous coordinate to convert to Euclidean 3D coordinates automatically. Some GPUs also have a texturing mode that automatically

performs the comparison between the resulting texture z coordinate and the depth stored in the shadow map, returning either 0 or 1 to indicate the result of the comparison. GPUs with hardware PCF filtering can return a result in the range $[0, 1]$. The result of the shadow map lookup can then be used to modulate the lighting on the surface.

6.1.1 Modifications for LogPSMs

LogPSMs are built on top of projective texture mapping. The perspective part of the logarithmic perspective parameterization is handled as just described. The logarithmic part requires two modifications. First, the shadow map is rendered using logarithmic rasterization. Later in this chapter we discuss how we simulate logarithmic rasterization with a fragment program. The next chapter describes how to extend existing hardware to handle logarithmic rasterization. Second, when rendering the image we perform the homogeneous divide on the texture coordinates manually and then apply the logarithmic transformation in Equation 5.43 to the y coordinate to get the final texture coordinates for the shadow map query.

6.2 Handling multiple shadow maps

Partitioning techniques result in multiple shadow maps that cover the scene. There exist many possibilities for rendering an image using multiple shadow maps. The image can be rendered with multiple passes using one shadow map at a time, with a single pass using all shadow maps at once, or with a combination of multiple passes each using multiple shadow maps. Each method has its own advantages and disadvantages. The best approach depends on the application and the graphics hardware used.

Multipass rendering. The most straightforward multipass approach is to accumulate the shadows in the final image by alternating between rendering a shadow map and rendering to the image from the eye's view using the shadow map. The first image pass can be rendered as normal. Subsequent passes, however, must be careful to modify only the regions covered by the current shadow map. Possibilities for restricting rendering to these regions include user clip planes, the stencil buffer, or the fragment kill instruction. Rendering the scene

multiple times can incur significant overhead, especially for complex scenes where the vertex transformation load is high. This cost can be reduced by culling objects that do not lie in the partition of the current shadow map as discussed later in Section 6.5, but this comes at the cost of increased implementation complexity.

Another possibility is to used deferred shadowing. In this approach, the first pass is generated as before, except that the eye-space depth of each pixel is stored in a texture. Subsequent passes render a single screen-sized quad instead of the scene. The eye-space depth texture is used to compute the fragment location in world space and perform the shadow map query. The main disadvantage of this approach is that texture coordinates and other interpolated quantities, as well as screen space derivatives (useful for filtering), have to either be computed manually, which can be significantly more expensive, or precomputed and stored in additional textures, which requires additional memory and bandwidth. Thus deferred shadowing trades vertex transformation load for higher storage, bandwidth, and fragment shader costs. For more details on deferred shadowing see (Giegl & Wimmer, 2007a; Giegl & Wimmer, 2007b).

Single pass rendering. Single pass rendering has several advantages over multipass rendering. It avoids the redundant computation of multipass rendering, and it also has access to hardware interpolators and derivatives that are not available to deferred shadowing. Single pass rendering also has several disadvantages. First, all the shadow maps have to be stored in memory. Shadow maps tend to be large and can quickly exhaust memory resources. Second, the fragment shader incurs extra cost in determining which shadow map to query. The texture coordinates for each shadow map can be interpolated in hardware, but the number of available interpolators is generally quite limited. When there are not enough interpolators, the texture coordinates must be computed in the fragment program, which also incurs greater cost.

Our current implementation of LogPSMs uses a single pass approach. The basic algorithm proceeds as follows:

1. Compute partitions. The partitions may consist of the entire view frustum (for a single

shadow map algorithm), z-partitioned subfrusta, or face partitions.

2. Compute the parameterizations for the shadow maps corresponding to each partition. This step involves fitting a light frustum to the partition and computing its corresponding matrix $\mathbf{M}_l$.

3. Allocate texture resolution for each shadow map based on the maximum error of each partition.

4. Render the shadow maps.

5. Render the image.

Multiple passes with multiple shadow maps. Rendering multiple passes with several shadow maps at a time can help alleviate the problem of too few interpolators without incurring as much of the overhead of multipass rendering. However, this approach also incurs the implementation complexities of both single pass and multipass approaches.

6.3 Fitting a light frustum to a partition.

The light frustum is determined by the light matrix $\mathbf{M}_l$. To maximize the use of the shadow map's spatial and depth resolution, the light frustum should be fit as tightly as possible to its corresponding partition. The shadow map need only include the visible portions of the scene. Therefore we compute a bounding polyhedron for each shadow map representing the intersection of its partition with the view frustum. For a single shadow map, the polyhedron is the view frustum itself. For z-partitioning schemes the polyhedron is a subfrustum of the view frustum. For face partitioning the polyhedron can be formed by clipping the view frustum polyhedron with planes passing through the light and each edge of the face. Tighter polyhedra may be obtained by intersecting with the scene bounds and performing further analysis to determine which parts of the scene are visible (Brabec et al., 2002b).

The shadow map only needs to store the depth of occluders within the view frustum. For occluders outside the view frustum, any depth value will work so long as it is less than the depths of the occluded surfaces. Therefore the far and near planes of the light frustum

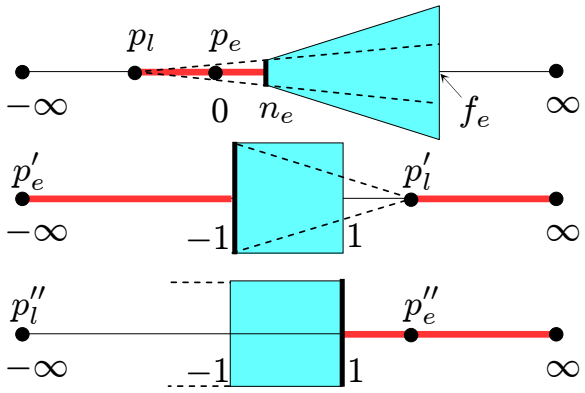

Figure 6.1: ***Post-perspective parameterization of frustum faces.*** (Top) *With the light p_l behind the eye p_e we parameterize the front face. The red points along the line contain potential occluders and must be included in the shadow map.* (Middle) *In post-perspective space the frustum becomes a unit cube, the eye goes to $-\infty$ and the light wraps around ∞. The front face is now a back face.* (Bottom) *After applying the light's perspective transform, p''_l is at $-\infty$ and the projection becomes orthographic. The inverted depth ordering can be corrected by leaving the light at $-\infty$ and scaling by -1 along the view direction.*

can be fit tightly to the polyhedron bounding the visible geometry. Occlusion information for occluders lying between the light and the near plane of the light frustum can be captured using depth clamping. If depth clamping is not available, the near plane of the light frustum must be set to include all potential occluders. When the light itself lies inside the view frustum, surfaces between the light and the near plane of the light frustum will not be shadowed. Therefore, the near plane should be chosen close enough to minimize the unshadowed region, but far enough away to preserve depth resolution. Unclamped floating point depth buffers, such as those supported by the recent *NV_depth_buffer_float* extension, can eliminate the unshadowed region, but the near plane must still be chosen carefully in order obtain sufficient depth resolution.

6.3.1 Fitting face partitions in post-perspective space

For face partitioning we can use a perspective warped cube map algorithm (Kozlov, 2004) to handle the uniform parameterization on end faces and the perspective portion of a logarithmic parameterization on side faces. The main idea is to parameterize the faces of the view frustum in the post-perspective space of the camera. First, the light $\mathbf{l}$ is transformed to $\mathbf{l}'$ by the camera matrix $\mathbf{M}_c$. Under this transformation, the view frustum is mapped to the unit cube. Second,

a projection matrix for the light $\mathbf{P}_l$ is fit to each back face of the unit cube with the light frustum's near plane oriented parallel to the face. $\mathbf{P}_l$ is a simple orthographic projection if $\mathbf{l}'$ is a directional light (homogeneous coordinate l'_w is 0) or an off-centered perspective projection if $\mathbf{l}'$ is a point light. If l'_w is negative, the depth order relative to the light becomes inverted. To restore the proper depth order, the row of $\mathbf{P}_l$ corresponding to the z-axis should be scaled by -1 (see Figure 6.1). The inversion occurs whenever the face selection criterion in Equation 4.15 selects the front faces, i.e. $\hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e}) > 0$. Thus parameterizing back faces in post-perspective space always selects the appropriate faces. The partition polyhedra can be computed by taking the convex hull of the back faces of the unit cube and the light and intersecting them with the unit cube. The final matrix used to render each shadow map includes the camera matrix, $\mathbf{M}_l = \mathbf{P}_l\mathbf{M}_c$.

When using face splitting or coordinate frame adjustment to handle shearing, we have found it easier to work in light space rather than the post-perspective space of the camera.

6.4 Dynamic allocation of texture resolution.

The available texture resolution can be distributed so as to balance the maximum error between partitions and shadow map directions as described in Section 4.4.1. To do so we need we need to compute R_s and R_t (Equation 4.5). For a directional light, the only term in δ_b that varies is d_e, because $n_l/d_l \to 1$ and $\cos\phi_l$ is constant. Because δ_l is nondecreasing for uniform, perspective, and logarithmic perspective parameterizations and d_e is monotonic, the error distribution is monotonic over each face of the view frustum. This means that we can compute R_s and R_t by evaluating δ_l/δ_b only at the vertices of the view frustum faces and taking the maximum.

For a point light, computing the maximum of δ_l/δ_b over a face can be more involved. Evaluating the error at the point on each face that is closest to the light is an inexpensive approximation that is still fairly accurate. As a point light approaches a face, the resolution allocated to the face should drop to zero. However, the uniform, perspective, and logarithmic perspective parameterizations are not able to concentrate resolution locally near the light. Thus high error near the light leads to high resolution allocated across the whole face, which

takes resolution from the shadow maps for other partitions. The parabolic spacing functions discussed in Section 5.1.4.1 should be able to handle this situation better. When the point light stays within the view frustum, more pleasing results are obtained by leaving out the $\cos\phi_l$ term in the error calculations.

Current GPUs drivers are not necessarily optimized to handle the scenario of a texture changing resolution with each frame. For the experimental results presented in this paper we layout the shadow maps in a single fixed-sized texture atlas large enough to accommodate the changing shadow map dimensions.

It can sometimes be useful to favor the allocation of texture resolution in one direction more than another. This can be done by modifying Equation 4.41:

$$r_s = \sqrt{\frac{(1-\alpha)}{\alpha}\frac{R_s}{R_t}\sigma T}, \qquad r_t = \sqrt{\frac{\alpha}{(1-\alpha)}\frac{R_t}{R_s}\sigma T}, \tag{6.1}$$

where $\alpha \in [0,1]$ controls how much the resolution allocation is biased towards the t direction. A value of 0.5 balances equally between both directions.

6.5 Partition culling

Partitioning techniques can require multiple render passes, both for the shadow maps and the image, depending on the implementation. Each render pass typically involves only a single partition. In order to maximize the rendering performance for each pass, objects should only be rendered if they actually lie within the active partition. We call this *partition culling*. Partition culling can be accomplished by testing an object against the planes that contain the light and the partition silhouette edges using straightforward extensions of the existing view frustum culling techniques typically employed in graphics applications. As with view frustum culling, large objects can be decomposed into small sub-objects, and the entire scene can be arranged into a bounding volume hierarchy. With sufficiently fine culling granularity, the number of objects that overlap multiple partitions will be relatively low compared to the total number of objects, and most objects will only be rendered once. Thus with partition culling, the extra vertex transformation load can be significantly reduced. For simple scenes, partition

culling is of only small benefit and may not be worth the overhead, but for geometrically complex scenes this can be an important optimization.

Testing an object against each partition individually can involve redundant computations because the partition planes are often shared between adjacent partitions. Therefore, it can be more efficient to perform partition culling for all partitions at once. This can be done, for example, by arranging the partition planes into a balanced BSP tree.

GPUs that support DirectX 10 features can render to multiple render targets in a single pass. The main advantage of this is that vertex data need be transferred to the GPU only once. However, if the vertex data already resides on the GPU, this does little to reduce the overhead of multipass rendering because the vertex transformation load remains the same. So partition culling can still be an important optimization here as well.

6.6 Shadow map selection during image rendering

When rendering with multiple shadow maps in a single pass, we must determine which shadow map to query for each fragment and compute an appropriate set of texture coordinates. We will first discuss generally the issues surrounding shadow map and texture coordinate selection and then look at approaches for specific partitioning schemes.

On current GPUs the shadow maps are typically accessed in a fragment program using a texture lookup function that takes as arguments a texture sampler and a set of texture coordinates. The texture sampler is usually associated with an opaque pseudo-register which can be neither read nor written. The texture coordinate can be stored in a general register. Ideally, the texture coordinates for each shadow map should be interpolated. The number of interpolators, however, is usually quite limited. If there are not enough interpolators, texture coordinates have to be computed in the fragment program. One way to do this is by multiplying the world space coordinate of the fragment by the appropriate transformation matrix.

The most general straightforward way to store the data for multiple shadow maps would be in an array accessed by partition index. The programming model for fragment programs, however, has limitations on dynamic indexing of arrays. Samplers and interpolators cannot

be indexed dynamically. Older GPUs do not support any dynamic array indexing at all. Alternatives to dynamic array indexing include masked sums, conditionals, and dependent texture fetches. The masked sum approach uses a weighted sum of all the input data, where the weights are 0 for all partitions except for the desired partition, which is 1. The weights can be stored as a mask in the multiple channels of a vector data type. Suppose that we use hardware interpolation for the texture coordinates (denoted `TEX0-3`) of a shadow map with 4 partitions. We can select the proper set of coordinates using a 4 channel mask as follows:

```
float4 partitionMask = computePartitionMask();
float4 texCoord = partitionMask.x * TEX0 +
                  partitionMask.y * TEX1 +
                  partitionMask.z * TEX2 +
                  partitionMask.w * TEX3;
```

Masks for more than 4 partitions can be split across multiple variables, i.e. `mask0_3`, `mask4_7`, etc. The masking approach usually cannot be used for selecting a texture sampler because mathematical operations are not allowed on these objects. Instead, we can use conditionals:

```
sampler2D sampler;
if( partitionIndex < 2 )
{
  if( partitionIndex < 1 )
    sampler = partitionSamplers[0];
  else
    sampler = partitionSamplers[1];
}
else
{
  if( partitionIndex < 3 )
    sampler = partitionSamplers[2];
  else
    sampler = partitionSamplers[3];
```

```
}
```

The conditionals are arranged in a tree so that with dynamic branching only $\log(k)$ tests have to be performed, where k is the number of partitions. For a small k, masked sums may actually be faster because there is typically a performance penalty associated with a dynamic branch. On older GPUs conditionals are implemented with predicated instructions. Both sides of the conditional are executed so all k tests are performed. True dynamic indexing on older GPUs can be implemented with a texture fetch. Partition data can be placed in a texture and accessed as an array using the partition index as a texture coordinate. This approach has several limitations as well. Again, because of semantic restrictions, textures can not be used to store texture samplers. Textures can be used for matrices for computing texture coordinates, but the multiple texture fetches that can be required to retrieve all the entries have an associated performance penalty.

6.6.1 z-partitioning

The partition index for z-partitioning can be computed efficiently with a comparison and a dot product (Engel, 2007):

```
float4 zGreaterThanN = float4(n1_4 < eyePos.z);
float  partitionIndex = dot(zGreaterThanN, 1.0f);
```

The constant `n1_4` contains the near plane distances of the first four z-partitions (n_1, n_2, n_3, n_4). This method is easily extended to handle more partitions. A partition mask can be computed similarly:

```
float4 partitionMask = float4((n1_4 < eyePos.z) & (eyePos.z <= f1_4));
```

where `f1_4` contains the far plane distances of the first four z-partitions.

6.6.2 Face partitioning

For face partitioning we compute a partition index or mask using a cube map with a single texel in each face. In the texel of each face we store the index or mask of the partition corresponding to that face. We perform the cube map lookup in the camera's post-perspective

space using the method described by Kozlov (2004). We do not store the shadow maps directly in a cube map because cube maps do not currently support non-square textures of differing resolution or the necessary logarithmic transformation. We also store a flag in each face of the cube map indicating whether the face corresponds to a side face or an end face. This flag is needed in order to determine whether or not to apply the logarithmic transformation.

If side face splitting is used to handle shearing, we store the partition indices for both halves of the face in the cube map. The second index remains unused for unsplit faces. To choose the proper index we test the world space position of the fragment against the splitting plane for the face. We adopt the convention that the first index corresponds to the negative side of the plane. For faces without a split we use the plane equation $(0, 0, 0, -1)$ which is guaranteed to always give a negative result. To save computation in the fragment program we compute the distance of vertex positions from the split planes in a vertex program and interpolate. A code fragment using this technique is shown in Figure 6.2.

6.7 Logarithmic rasterization.

The LogPSM parameterization requires logarithmic rasterization. Logarithmic rasterization causes planar primitives to become curved. The logarithmic transformation could be computed on vertices in a vertex program, but because current GPUs only support projective transformations, the primitives would still be decomposed into planar triangles when rendered. When the f_e/n_e ratio is large, a very fine tessellation of the scene would be required in order to avoid error with this approach. We currently simulate logarithmic rasterization by performing brute-force rasterization in a fragment program. We first render the scene using $\mathbf{M}_l$ with the viewport set to the range $[0, 0] \times [1, 1]$ and readback clipped triangles using the OpenGL feedback mechanism. This gives us the triangles transformed by $\mathbf{F}_p(u, v)$. We compute the edge equations and depth interpolation equation for each triangle. We then transform the vertices of each triangle by the logarithmic part of $\mathbf{F}_{lp}$ and render a bounding quad. We use the fragment program in Figure 6.3 to invert the logarithmic portion of $F_{lp,t}$ on each fragment position. We then compare this position against the linear triangle edge equations and discard fragments that fall outside the triangle. The equation for the inverse

```
float computeFacePartition(
   uniform float4 lPosPP,   // location of light (post-perspective)
   uniform float3 invD0,    // 1/(d0,d1,d2)   dn is distance from lPos to
   uniform float3 invD1,    // 1/(d3,d4,d5)     cube face n
   varying float4 posPP,    // location of fragment (post-perspective)
   varying float4 splitDist,// distance to split planes for side faces
   samplerCUBE faceTex;

 {
   // Compute texture coordinate for cube map lookup.
   // From "PSM Care and Feeding" [Kozlov04]
   float3 cubeTexCoord;
   float3 v = posPP.xyz / posPP.w;
   float3 p = lPosPP;
   float3 dir = v - p;
   float3 maxTmp = max(dir * invD0, dir * invD1);
   float a = max( max(maxTmp.x, maxTmp.y), maxTmp.z);
   cubeTexCoord = p + dir/a;

   // compute partition from data stored in cube map face
   float4 faceData = texCUBE( faceTex, cubeTexCoord );
   float4 mask0_3 = unpack_4ubyte(faceData.r);
   float4 faceIndex = faceData.g;
   return faceIndex.g
        + dot( float4(splitDist > 0), mask0_3 ) * 4;
 }
```

Figure 6.2: ***CG code used for face partitioning.*** *The side faces are assigned indices 0–3. If a face n is split, its other half is assigned index $n + 4$. The end faces are assigned indices 6 and 7. The cube texture stores the index of each face and a mask corresponding to the face. All channels of the mask are 0 for end faces. Note that light positions at infinity in post-perspective space must be approximated as a distant point light for Kozlov's cube map texture lookup code (Kozlov, 2004) to work properly.*

of the logarithmic portion $F_{lp,t}(v)$ is given by:

$$
\begin{aligned}
v &= d_0 e^{d_1 t} + d_2 \\
d_0 &= \frac{1}{c_1(y_1 - y_0)} \\
d_1 &= \frac{1}{c_0} \\
d_2 &= -\frac{c_1 y_0 + c_2}{c_1(y_1 - y_0)}.
\end{aligned}
\tag{6.2}
$$

```
void logRasterize(
  varying float3 stzPos,   // 2D shadow map position (s,t) + depth (z)
  varying float3 edgeEq0,
  varying float3 edgeEq1,
  varying float3 edgeEq2,
  varying float3 depthEq,
  uniform float  d[3],
  out float depth : DEPTH )
{
  // invert transform
  float3 p = stzPos;
  p.y = d[0]*exp( d[1]*p.y )+d[2];

  // test against edges
  if( dot( p, edgeEq0 ) < 0  ||
      dot( p, edgeEq1 ) < 0  ||
      dot( p, edgeEq2 ) < 0    )
    discard;

  // compute depth
  depth = dot(p, depthEq);
}
```

Figure 6.3: ***CG code used to simulate logarithmic rasterization in a fragment program.***

With our unoptimized simulator we observe frame rates of 2–3 frames per second on a PC with a GeForce 6800GT graphics card and a 2.8 GHz processor. Most of that time is spent in computing the bounding quads. GPUs that support geometry shaders can probably be used to create the bounding quads for the warped triangles much more efficiently, but we have not yet implemented this. Even with a geometry shader, however, logarithmic rasterization would likely be considerably slower than linear shadow map rasterization. GPUs have a number of optimizations which are disabled when a fragment program outputs depth information, such as z-culling and higher rasterization rates for depth-only rendering.

In the following chapter we show how logarithmic rasterization can be implemented on graphics hardware in order to render LogPSMs at rates comparable to other algorithms based on linear rasterization.

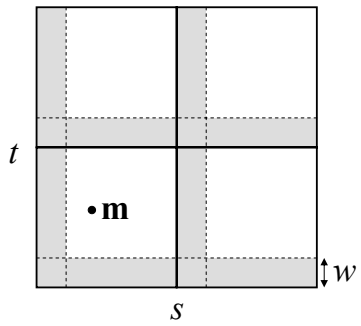

Figure 6.4: ***Procedural grid shader.*** *The heavy lines represent texel boundaries in texture space. A point m lies on a grid line if the fractional components of its s and t coordinates are less than the grid line width w.*

6.8 Procedural grid shader

Throughout this thesis we have used a grid to better visualize the projection of the shadow map texels onto the scene. We compute this grid procedurally in a fragment shader. The shader works by determining if a given point lies on a grid line in texture space. This is done by comparing the fractional components of the texture coordinate to the width of the grid line, as shown in Figure 6.4. When the texture is projected onto the scene, however, the width of the grid lines can become stretched or compressed. Therefore we scale the grid line width by the length in screen space of a unit step in s or t (calculated using derivatives). Code for the shader is provided in Figure 6.5.

```
float3 gridShader (
  varying float3 baseColor,
  varying float2 texCoord,
  uniform float  gridLineWidth,
  uniform float3 gridColor)
{
  float2 scaledTC = texCoord/spacing;
  float2 dS(ddx(scaledTC.s),ddy(scaledTC.s));
  float2 dT(ddx(scaledTC.t),ddy(scaledTC.t));
  float2 m = frac( scaledTC );
  if( m.s < gridLineWidth*length(dS) ||
      m.t < gridLineWidth*length(dT) )
  {
    return gridColor;
  }
  else
    return baseColor;
}
```

Figure 6.5: *CG code for procedural grid shader.*

CHAPTER 7

Logarithmic rasterization hardware

The importance of shadows has led vendors to introduce hardware features and optimizations targeted specifically at reducing the cost of shadow rendering (NVIDIA Corp., 2004). In this chapter we present additional hardware enhancements to support high-quality rendering of shadow maps. Shadow maps capitalize on the enormous fill rates of current GPUs. For example, the GeForce 8800 generates 192 samples per clock for z-only rendering (NVIDIA Corp., 2006). At the GPU's clock rate of 575 MHz, this amounts to an astounding fill rate of 110 billion pixels per second. These fill rates are obtained by using rasterization in conjunction with depth buffer compression. Rasterization makes heavy use of parallelism and exploits a high degree of coherence both in computation and memory access. Compression provides more efficient use of available memory bandwidth. Current GPUs also have high-speed memory interfaces. Nevertheless, memory bandwidth can still be a bottleneck for high resolution shadow maps. High resolution shadow maps also have high storage costs and increase contention for limited GPU memory.

LogPSMs can help alleviate these problems. For the same error as competing shadow map algorithms, LogPSMs require less resolution. Unfortunately, the logarithmic rasterization required by LogPSMs is not supported on current GPUs. In this chapter, we discuss how to implement logarithmic rasterization through incremental modifications to graphics hardware. The primary advantage of our proposed enhancements is that they capitalize on existing hardware designs. Our low-cost enhancements enable significant improvement over existing shadow map algorithms without compromising previous capabilities. Logarithmic rasterization requires a modest amount of additional computational power and can produce significant bandwidth and storage savings. Therefore these enhancements align well with

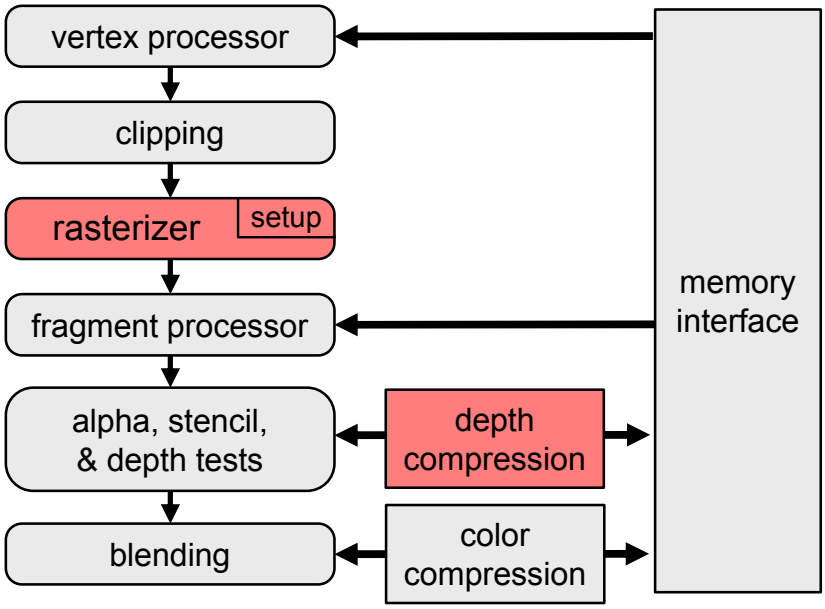

Figure 7.1: ***Graphics pipeline.*** *We propose modifications to the rasterizer and depth compression units. Note that both compression units perform decompression as well.*

current hardware trends of decreasing cost for on-chip computation and a relatively high cost for off-chip bandwidth.

We adopt a somewhat simpler notation in this chapter. We will use $(x, y) \in [0, 1]^2$ to denote coordinates after applying the perspective transformation. y' denotes the y coordinate after applying the logarithmic transformation F:

$$y' = F(y) = c_0 \log(c_1 y + 1) \tag{7.1}$$
$$c_0 = \frac{-1}{\log(f/n)}, \quad c_1 = \frac{1 - (f/n)}{(f/n)},$$

where n and f are the near and far plane distances used for the perspective transformation.

The perspective part of the LogPSM parameterization can be handled using the standard graphics pipeline. The logarithmic part, however, requires logarithmic rasterization, which causes planar primitives to become curved. To rasterize these curved primitives at the same rate as planar primitives we modify just two fixed-function hardware components – the rasterizer and the depth compression. After briefly reviewing rasterization based on edge equations, we describe our modifications.

7.1 Rasterizing with edge equations

The rasterizer performs two main functions. First, it determines which pixels are covered by a primitive. Second, it linearly interpolates attributes from the vertices to the covered pixels. Coverage determination on modern, high-performance graphics hardware is typically performed using implicit edge equations of the general form:

$$E(x, y) = Ax + By + C. \tag{7.2}$$

The sign of E at a point (x, y) indicates on which side of the edge the point lies. Points inside a convex primitive lie on the positive side of all its edges. One of the main advantages of rasterizing with edge equations is that the evaluation of E over many pixels is easy to parallelize. Current GPUs evaluate edge equations for a tile of samples (e.g. a 4×4 block) in parallel. To make the most efficient use of hardware, GPUs perform a hierarchical traversal of the image. A coarse stage identifies tiles that are at least partially covered by a primitive. The tiles are traversed in a manner that is favorable for paged memories and maximizes cache coherence (Pineda, 1988; McCormack & McNamara, 2000; McCool et al., 2001). A fine stage then computes coverage for individual samples within a tile. Attributes for the pixels, such as color, depth, and texture coordinates, can be linearly interpolated from the vertices using an equation of the same form as Equation 7.2. Olano and Greer (1997) show how to compute the coefficients for these equations. The coefficients are computed in the setup phase of the rasterizer.

The edge equations are typically computed with fixed point precision. The conversion of vertex positions from floating point to fixed point can be thought of as "snapping" the vertices to discrete coordinates on a uniform grid. With discrete coordinates the edge equations can be evaluated at grid locations *exactly* by using a sufficient number of bits to avoid truncation. This provides a water-tight rasterization without the double-hit or missing pixels that can result from numerical robustness problems.

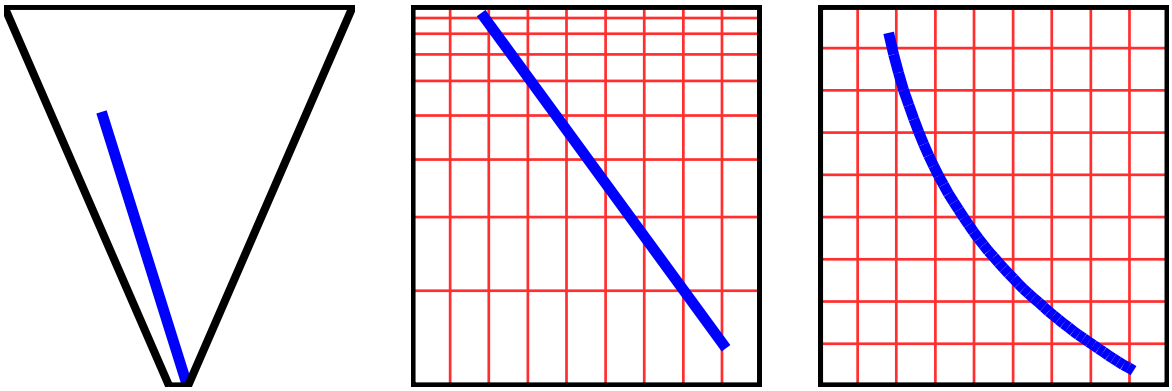

Figure 7.2: ***Transformation pipeline for LogPSMs.*** (Left) *A world space view of a line (blue) and a side face of the view frustum (black).* (Middle) *After the perspective transformation the side face has been become a square.* (Right) *The logarithmic transformation causes the line to become curved. The uniform sample grid (red) in the log warped space corresponds to a nonuniform rectilinear grid in the linear post-perspective space.*

7.2 Logarithmic rasterization

Logarithmic rasterization can be thought of as rasterizing curved primitives on a uniform grid, or as rasterizing linear primitives on a grid that is nonuniform in only one dimension (see Figure 7.2). We adopt the latter view to facilitate the explanation of how linear rasterization can be extended to support logarithmic rasterization. Our approach is to snap the nonuniform grid samples to positions on an underlying uniform grid that is fine enough to distinctly represent the samples. The edge equations can then be evaluated exactly using fixed-point arithmetic to provide water-tight rasterization.

We will now describe our modified edge equations, how to compute coverage with them, the amount of precision required to evaluate them, and a generalized polygon offset for handling self-shadowing artifacts.

7.2.1 Modified edge and interpolation equations

The edge equations for logarithmic rasterization can be computed by transforming y' in the warped space back to y in the linear space using the inverse of Equation 7.1 and composing

the result with Equation 7.2:

$$E'(x, y') = E(x, G(y')) = Ax + BG(y') + C \tag{7.3}$$

$$G(y') = F^{-1}(y') = \frac{e^{(y'/c_0)} - 1}{c_1} = \frac{(f/n)(1 - (f/n)^{-y'})}{(f/n) - 1}. \tag{7.4}$$

The interpolation equations are handled similarly. Note that the coefficients of these equations are based on the vertex positions in linear space resulting from the perspective part of the LogPSM parameterization. Rasterization algorithms that initialize tile traversal using vertex positions can apply Equation 7.1 to the y coordinate of the vertices during triangle setup to get a starting point.

Several algorithms used during rasterization rely on the fact that quantities computed at tile corners provide conservative bounds for those quantities over the whole tile. For example, the values of the edge equations at the tile corners are used to steer tile traversal. Conservative depth bounds obtained from tile corners are used for culling optimizations such as the hierarchical z-buffer or z-min and z-max culling. The warped edge and interpolation equations remain monotonic so values at tile corners still provide conservative bounds. This means that algorithms that rely on this property can be used for logarithmic rasterization without modification.

7.2.2 Coverage determination

Coverage for a tile of samples could be computed brute-force by an array of edge equation evaluators for each sample in the tile. We conserve die area, however, by exploiting the linearity of E' in the x dimension to compute the edge equations incrementally:

$$E'(x_0 + \Delta x, y') = E'(x_0, y') + A\Delta x. \tag{7.5}$$

We first perform a full evaluation of the edge equations in parallel for the samples in the first column of the tile. We then compute the values for the remaining columns in parallel by simply adding a constant to the first column. These calculations can be pipelined, so that a sustained rate of one 4×4 tile of samples can be computed per clock. A few cycles

of downstream buffering allow the coverage bits for each tile to be aggregated together and presented broadside to downstream units. The extra latency incurred by this approach is small compared to the depth of current GPU pipelines. Because the logarithmic rasterization mode is expected to be used only for shadow maps, issues regarding sample placement for multisample antialiasing are not a concern.

Several other possibilities exist for evaluating the edge equations. One possibility is to also compute the $G(y')$ term incrementally along the first column. After a full evaluation of $G(y')$ for the first row, $y' = y'_0$, the values on subsequent rows k could be computed in parallel with a multiply-add operation:

$$G(y'_0 + k\Delta y') = G(y'_0)P_k + Q_k \tag{7.6}$$

$$P_k = (f/n)^{-k\Delta y'}, \quad Q_k = (f/n)\frac{(1-(f/n)^{-k\Delta y'})}{(f/n)-1}. \tag{7.7}$$

The constants P_k and Q_k depend only on the parameterization and can be used for all primitives in the shadow map. Computing $G(y')$ for the first column incrementally is somewhat more involved because it requires an extra step and creates extra latency. Another possibility is to precompute the values of $G(y')$ at each scanline and store them in a table. The table could possibly be constructed using floating point hardware already available for vertex/fragment processors. The evaluation of the edge equations for the first column in a tile would then involve only a table look-up and a dot product. Because tiles are traversed in a predefined order, table entries used by a tile could be prefetched. The table would have to be quite large for a high resolution buffer. The size could be reduced somewhat by precomputing $G(y')$ only for the first row of a tile and computing the other rows incrementally.

7.2.3 Precision requirements

To evaluate the edge equations exactly we require fixed-point $G(y')$ values. We must use at least enough fixed-point precision that any two subsequent values of $G(y')$ and $G(y' + \Delta y')$ can be represented distinctly. In other words, the least significant bit should represent a quantity no larger than $\Delta G_{\min} = \min_{y'} |G(y') - G(y' + \Delta y')|$. Given that the range of G is

$[0, 1]$, the minimum number of bits required is:

$$b_{\min} = \lceil \log_2(1/\Delta G_{\min}) \rceil. \tag{7.8}$$

Because $\Delta G_{\min}$ depends on the far to near plane depth ratio (f/n), using a fixed number of bits will place constraints on the range of depth ratios that can be rendered accurately. Suppose we desire to rasterize a $2^{b_x} \times 2^{b_y}$ buffer. The distance between uniform samples in $y' \in [0, 1]$ is $\Delta y' = 1/2^{b_y} = 2^{-b_y}$. From Figure 7.2b we can see that the resulting nonuniform spacing in $y = G(y')$ is minimal at $y' = 1$. Thus $\Delta G_{\min} = G(1) - G(1 - 2^{-b_y})$. Plugging this value into Equation 7.8 yields an expression that can be bounded fairly tightly for $(f/n) \in [1, 10^5]$ as:

$$b_{\min} < b_y + 0.8 \log_2(f/n). \tag{7.9}$$

The vertices used to setup the edge equations are limited to the 24 bit precision of a 32-bit floating point mantissa. Therefore we choose to represent fixed-point y coordinates and $G(y')$ values with 24 bits. Plugging $b_{\min} = 24$ in Equation 7.9 we get an expression for the largest (f/n) that can be handled accurately:

$$(f/n)_{\max} = 2^{\frac{(24-b_y)}{0.8}}. \tag{7.10}$$

For a $4K \times 4K$ buffer $b_y = 12$ and $(f/n)_{\max} \approx 3.3 \times 10^4$. This depth ratio is quite large. For view frusta with even higher depth ratios, z-partitioning can be used to reduce the depth ratios of the individual pieces. Unlike existing shadow map methods, this z-partitioning is not for controlling aliasing error, but to deal with precision limitations. Fortunately, few splits if any are required. For example, a single LogPSM can accurately render shadows on receivers from 1m to $33,000$m away. A single split squares this range to nearly $1,000,000$m.

Standard rasterization typically renders to a sub-pixel grid for increased accuracy. Adding sub-pixel resolution in y effectively increases b_y, which would impose further limitations on $(f/n)_{\max}$. However, this may not be necessary because our approach already provides sub-

pixel resolution in y. ΔG increases linearly from $\Delta G_{\min}$ at $y = 1$ to $(f/n)\Delta G_{\min}$ at $y = 0$. When the depth ratio is equal to $(f/n)_{\max}$, the regions near the viewer enjoy an ample amount of sub-pixel resolution. While less sub-pixel resolution is available for the distant regions, this may not be a problem because the distant regions tend to be less important. For low depth ratios, sub-pixel resolution is available for both the near and distant regions.

7.2.4 Attribute interpolation

Attribute interpolation uses equations of the same form as E'. The interpolation equations can be evaluated in the same way as the edge equations. The only difference is that they need not be evaluated exactly, so floating point or lower precision fixed-point representations may be used. We modify the depth interpolation slightly to handle polygon offset as described in the following section.

7.2.5 Generalized polygon offset

Polygon offset is often used to generate a depth bias used to prevent false self-shadowing. The standard offset is $s \cdot m_z + u$, where s is a scale factor, m_z is the depth slope of the polygon, and u is a constant. The OpenGL Specification (Segal & Akeley, 2006) defines an approximation of the depth slope m_z as:

$$m_z = \max\left(\left|\frac{\partial z}{\partial x}\right|, \left|\frac{\partial z}{\partial y}\right|\right). \tag{7.11}$$

The polygon offset for linear rasterization can be computed in setup and folded into the depth interpolation equation. For logarithmic rasterization we compute the value of the warped space $|\partial z \partial y'|$ in linear space using the chain rule:

$$\left|\frac{\partial z}{\partial y'}\right| = \left|\frac{\partial z}{\partial y}\right| \left|\frac{\partial y}{\partial y'}\right| = \left|\frac{\partial z}{\partial y}\right| \left|\frac{c_1 y + 1}{c_0 c_1}\right|. \tag{7.12}$$

The $|(c_1 y + 1)/(c_0 c_1)|$ term decreases with y. Unlike linear rasterization where m_z is either $|\partial z/\partial x|$ or $|\partial z/\partial y|$ over the entire polygon, m_z with logarithmic rasterization may switch from $|\partial z/\partial y'|$ to $|\partial z/\partial x|$. One way to handle this is to use two depth interpolation equations, each combined with the polygon offset for $|\partial z/\partial x|$ and $|\partial z/\partial y'|$, respectively. The max operator

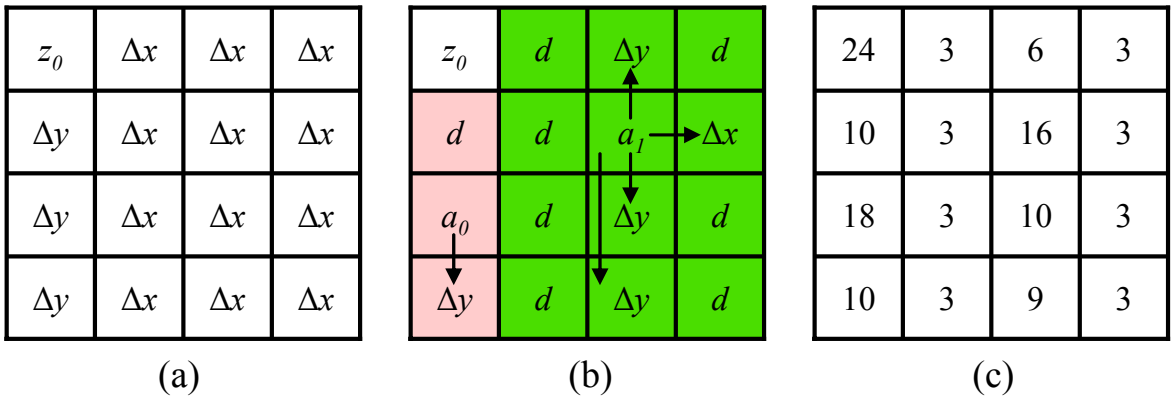

Figure 7.3: ***Depth buffer compression algorithm for logarithmic rasterization.*** *(a) First-order differentials are computed in y and x by subtracting the depth value of the neighbor immediately above or to the left, respectively. (b) This is followed by a variation of anchor encoding. For the pink region, the value at the anchor position a_0 and its Δy offset form a line used as a predictor with d as a correction term. In the green region a_1, Δx, and Δy values form a predicting plane. A different Δy is used for encoding each row. (c) Bit allocation for each quantity.*

could then be performed per pixel. Another possibility is to split the polygon along the scan line where the switch occurs. Each polygon half is then rasterized with the appropriate depth interpolation equation. The simplest approach, however, is to compute m_{z0}, the value of m_z at the lower y-bound of the polygon y_0, and m_{z1}, the value of m_z at the upper bound y_1, and use a single equation $m_z(y)$ that varies linearly between them:

$$m_z(y) = \frac{m_{z0}(y1 - y) + m_{z1}(y - y0)}{(y_1 - y_0)} \tag{7.13}$$

The offset computed using this equation can be folded into a single depth interpolation equation. When no switch in the maximum depth slope direction occurs on the polygon, Equation 7.13 gives the correct result. When there is a switch, it provides a conservative approximation. The maximum relative difference in the approximation is largest when $y_1 - y_0 = 1$ with a value of about 1, but for shorter extents the difference is far less (see Appendix B for an analysis of the error of this approximation). Note that for paths through our test scenes a switch actually occurs for less than 1% of the polygons. For those polygons with a switch, the mean of the maximum relative difference between our approximation the actual value is less than .001.

7.3 Depth buffer compression

Depth buffer compression is an important optimization for reducing the memory bandwidth requirements of rasterization. We refer the reader to Hasselgren and Möller (2006) for an excellent survey and detailed explanations of existing depth compression schemes. Many of the current algorithms exploit the planarity of a tile's depth values to achieve fast, lossless compression. Because logarithmic rasterization causes planar primitives to become curved, many of the existing algorithms fail to achieve good compression ratios.

We compose two existing first-order schemes to produce a second order compression scheme that is better able to capture the curvature in the y direction. The algorithm is summarized in Figure 7.3. First, we store the base value in the upper-left corner of the tile at full 24-bit precision. We compute first order differentials for the remaining values (DeRoo et al., 2002). For the green 4×3 block on the right, the differentials should be fairly small since the edge equations are linear in x. We then use a variation of an anchor encoding scheme (Van Dyke & Margeson, 2005). Offsets are computed with respect to anchor values. Together the offsets and anchor values form a line or plane that is used to predict values for the rest of the locations. Correction terms for the predicted values are also computed. If the values of the anchors, offsets, and correction terms all fit within their allotted bit budget, the tile can be compressed. Otherwise, it is stored uncompressed. Decompression simply reverses the process. Our scheme uses a 128-bit allocation to achieve lossless compression with a $3:1$ compression ratio.

7.3.1 Results

We evaluated our depth compression scheme in a way similar to that of Hasselgren and Akenine-Möller (2006). We captured data for camera paths through three models of varying complexity using 4 different resolutions. We also used some of the same compression algorithms (Van Dyke & Margeson, 2005; DeRoo et al., 2002; Ornstein et al., 2005; Hasselgren & Akenine-Möller, 2006). We could not test the depth offset algorithm because this requires the z-min and z-max values, which are not supplied by our simulator. For each frame we split the completely rendered depth buffer into 4×4 tiles (excluding tiles untouched during scene

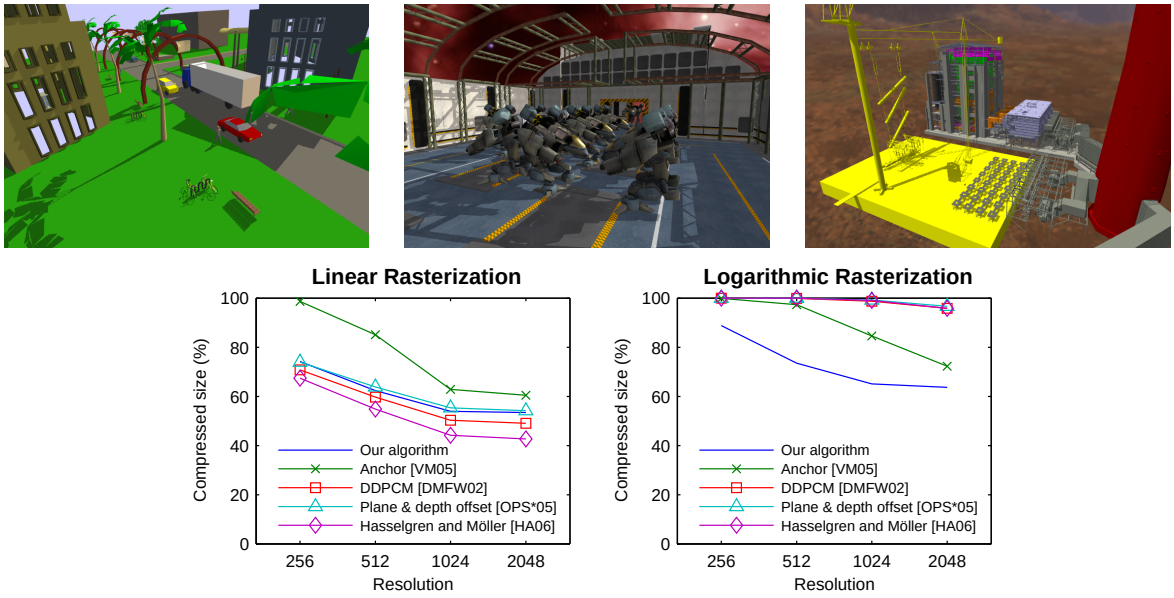

Figure 7.4: ***Depth compression results.*** *(Top row) Benchmark scenes: Town (58K triangles), Robots (95K triangles), and Power Plant (1.7M triangles) The full Power Plant has 13M triangles, but we left out internal pipes. (Bottom row) The two graphs show the average depth compression for several algorithms for shadow maps rendered with both linear and logarithmic rasterization. Our depth compression algorithm provides reasonable compression for both.*

rasterization) and ran them through the compression algorithms. A more detailed simulation would capture the bandwidth of partially completed tiles. The results shown in Figure 7.4 are intended to indicate the relative performance of the algorithms. A visual comparison of our algorithm with a linear depth compression scheme is shown in Figure 7.5.

On depth buffers rendered with linear rasterization, our compression scheme is on par with some of the other algorithms. For logarithmic rasterization the other algorithms fare poorly. Our compression scheme is still able to achieve fairly good compression ratios, especially at higher resolutions, although the compression ratios are somewhat lower than those for linear rasterization.

7.4 Feasibility

The following list is a summary of the modifications required to support logarithmic rasterization:

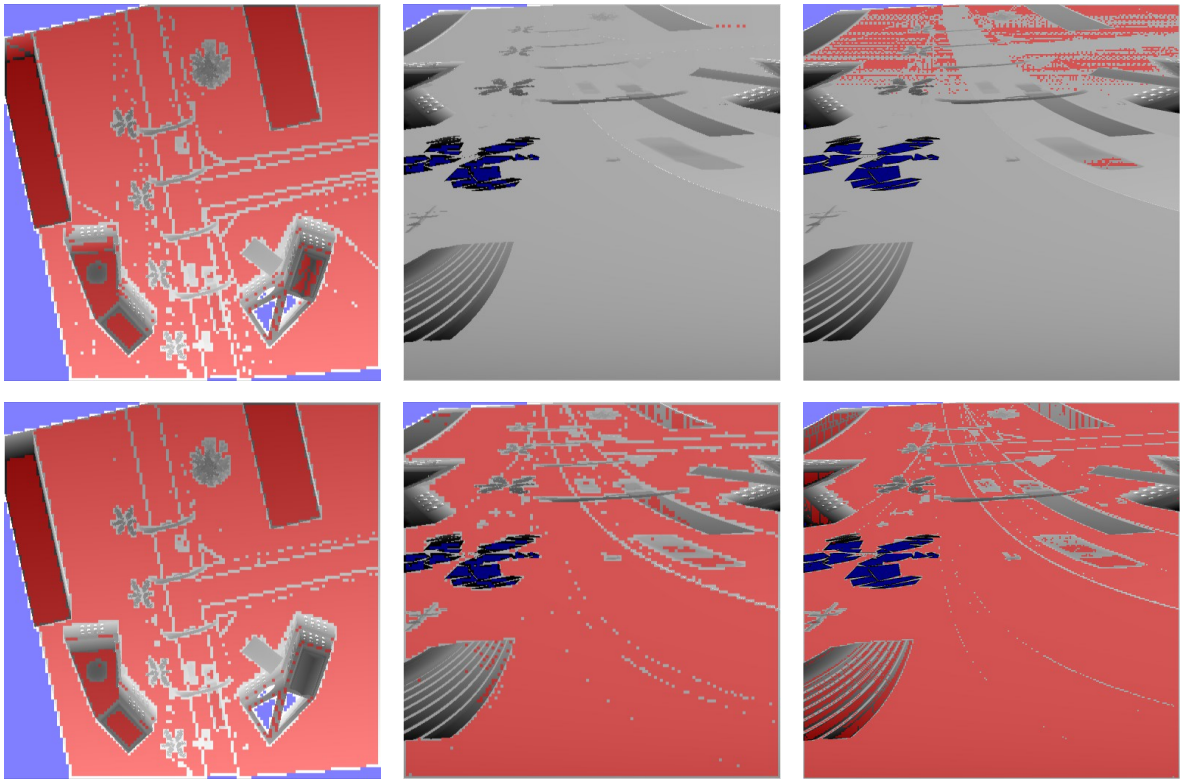

Figure 7.5: ***Depth compression visualization.*** *The depth buffers on the top row are compressed using the linear depth compression algorithm of Hasselgren and Akenine-Möller (2006). Our algorithm was used on the bottom row. The images are tinted red for compressed tiles and blue for tiles where the sample depths are either all* 1 *because they were untouched during rasterization or all* 0 *due to depth clamping.* (Left) *Both algorithms perform well on a standard shadow map using linear rasterization.* (Middle) *The linear depth compression fails completely for a logarithmic warped shadow map at lower resolutions (*512×512*) due to the curvature of the primitives induced by the warping.* (Right) *At higher resolutions (*1024×1024*), the linear depth compression starts to work on tiles far from the viewer where there is less curvature.*

- An increase in bit width in the rasterizer to support the 24-bit fixed-point y coordinates. Details about the bit widths in current GPUs are not generally available. A GPU supporting $4K \times 4K$ buffers with 4 bits of subpixel precision would have to support at least 16 bit fixed-point coordinates.

- Evaluators to compute the exponentials in the edge and interpolation equations.

- Application of the logarithmic parameterization to y coordinates in setup.

- Computation of generalized polygon offset in setup.

- A new depth compression unit.

The rest of the graphics pipeline remains the same.

We believe that these enhancements are feasible because they are incremental and leverage existing hardware designs. The modifications are isolated to only two fixed-function elements of the GPU pipeline. The changes to the rasterizer are incremental. The new depth compression unit using our algorithm would have complexity comparable to other algorithms. If desired, the depth compression unit could sit alongside the existing one used for linear rasterization. Because our algorithm also provides comparable compression ratios for linear rasterization, our depth compression unit could feasibly replace the existing unit, thereby enabling a single unit to handle both linear and logarithmic rasterization.

Our enhancements require only a modest increase in on-chip calculations but deliver significant reductions in bandwidth and storage requirements. Using increased computation to save bandwidth aligns well with current hardware trends because for the same cost, computational power continues to increase rapidly while bandwidth lags considerably behind. Therefore, these enhancements provide a good balance between bandwidth reduction and implementation cost.

CHAPTER 8

Conclusion

The goal of this thesis has been to generate high-quality, hard-edged shadows for real-time applications. Shadow maps are one of the most attractive algorithms for achieving this goal because of their flexibility and good performance characteristics, but they suffer from aliasing artifacts. Our focus has been on reducing aliasing error by using a better sampling distribution to render the shadow map. Ideally, we would like to sample exactly where the shadow map queries are generated. But this is difficult to do with real-time performance because the irregular data structures that such an approach requires are difficult to implement efficiently on current hardware. Instead we have proposed a new warping technique, logarithmic perspective shadow maps (LogPSMs), which is more suitable for current hardware designs. The logarithmic rasterization required to support this technique can be implemented through relatively minor enhancements to existing hardware, and can achieve performance comparable to the real-time shadow map algorithms used today, but with less error.

Like existing warping methods, LogPSMs use a perspective projection to tightly fit the extents of a shadow map to the view frustum or its side faces. This causes the spacing between shadow map samples to increase quadratically along the view direction. In general, however, the spacing should increase linearly to match the spacing between shadow map queries, thus avoiding high error. LogPSMs correct the quadratic spacing by applying a logarithmic transformation.

We have conducted an in-depth analysis of aliasing error to determine just how much improvement LogPSMs can give over existing methods. The focus of our analysis has been to minimize the maximum error at any point in the view frustum without requiring information about the scene. Since the perspective factor of aliasing error does not depend on the orien-

tation of the surfaces in the scene, we minimize this factor. We have introduced the storage factor as a metric for perspective aliasing error which takes into account the maximum error along both shadow map dimensions. Minimizing the storage factor yields a parameterization that is near optimal. Unfortunately, this parameterization is quite complex and therefore impractical. We have shown however, that LogPSMs have the same maximum error as this parameterization. This is not the case with existing warping algorithms. This means that the error of LogPSMs is closer to the lower bound.

8.1 Benefits of logarithmic perspective shadow maps

One of the primary benefits of LogPSMs is that they give a more uniform error distribution throughout the view frustum. Other methods may actually have lower error than a LogPSM for different parts of the scene. For instance, a standard shadow map will typically have less error than a LogPSM for regions at the far plane, but this comes at the expense of much higher error at the near plane.

In general, a LogPSM algorithm has lower maximum error than the corresponding algorithm based on perspective warping. We have shown that the storage factor for perspective warping is $O(f/n)$, where f and n are the near and far plane distances. In contrast, the LogPSM parameterization has a storage factor of $O(\log(f/n))$. Therefore, LogPSMs give the greatest improvement over other methods in scenes with high f/n.

The LogPSM parameterization can be used to warp a single shadow map or it can be can be combined with a partitioning technique such as face partitioning or z-partitioning. A single shadow map LogPSM can provide the greatest error reduction of any existing scene-independent warping technique, but only for a subset of all light directions. The lowest error is achieved when the light direction is perpendicular to the view direction. The highest error occurs when the light is behind or in front of the viewer, in which case warping cannot be used. To obtain low error for these light directions, a partitioning scheme must be used. z-partitioning reduces the error for the light behind case, but does not affect the error when the light direction is perpendicular to the view direction. As the number of z-partitions increases the relative difference between LogPSM warping and perspective warping decreases. For a

large number of partitions, warping provides little benefit over a uniform parameterization. Therefore, the LogPSM parameterization gives the greatest relative advantage over a uniform or perspective parameterization when only a few z-partitions are used.

To get the benefit of warping over all light directions, a face partitioning scheme must be used. This increases the number of shadow maps that need to be rendered. We show that for the same number of shadow maps, z-partitioning actually gives lower error than face partitioning combined with perspective warping (unless z-partitioning is also used on the faces, but then face partitioning is better only with a large number of shadow maps, and the relative difference is quite small.) This is not the case with LogPSMs. With face partitioning, LogPSMs always have a lower maximum error over all light directions than z-partitioning. But with a sufficient number of partitions, a z-partitioning scheme can get quite close to the same error as a LogPSM. Section 5.3.6 describes how a LogPSM can be approximated with z-partitioning. For a directional light, 8–16 z-partitions begins to be competitive with LogPSMs, even for the worst case light directions. For an omnidirectional point light, LogPSMs have more of an advantage over z-partitioning because z-partitioning would have to be applied to multiple faces. In the end, the benefit of LogPSMs comes down to the relative difference in cost between rendering a few shadow maps with logarithmic rasterization and rendering a larger number of shadow maps with z-partitioning.

8.2 Limitations

LogPSMs share the same limitations of other scene-independent methods. Because these methods have no knowledge of surface orientation, they cannot handle the projection factor. Therefore they cannot guarantee that aliasing is completely removed from an image. The shadow map must already be several times larger than the image in order to eliminate perspective aliasing. To reduce projection aliasing to acceptable levels on *most* surfaces, the shadow map must typically be several times larger still, but this may still not be enough for *all* surfaces. Many of these samples are wasted on parts of the scene that are not even visible to the light. Compared to an irregular z-buffer that requires no more shadow samples than the number of pixels in the image, scene-independent methods can be quite inefficient

in terms of sample usage.

The LogPSM algorithm that gives the most benefit over all light directions uses face partitioning. While not overly-complex, face partitioning is not as simple as z-partitioning. The number of face partitions varies with the light position, but the number of z-partitions remains constant. Computing the face partitions themselves can require convex polyhedron intersection and clipping routines. Robust implementations of these routines are nontrivial. In addition, face partitioning can exhibit bad shearing artifacts. We have presented some techniques to work around this, but they are not always completely satisfactory, and they burden the algorithm with additional complexity.

Perhaps the biggest limitation of LogPSMs is that logarithmic rasterization is not available on current GPUs. It can be simulated either using an adaptive tessellation of the scene or with brute force rasterization in the fragment program, but both of these approaches add implementation complexity and can degrade performance significantly. We have shown how logarithmic rasterization can be implemented with modifications to graphics hardware. Better algorithms, like the irregular z-buffer, can also be implemented with hardware modifications, but the changes required for logarithmic rasterization are more incremental. However, logarithmic rasterization is less general than the irregular z-buffer. In fact, we have been unable to come up with another application for logarithmic rasterization besides shadow rendering. Because die area is precious, vendors will prefer more general features over specialized ones. Shadows are a very important part of real-time applications but whether the savings in bandwidth and storage that logarithmic rasterization can deliver through LogPSMs will be sufficient to merit a hardware implementation will depend on economic factors and the continued popularity of shadow maps.

8.3 Future Work

In this thesis we have investigated combinations of warping with z-partitioning and face partitioning. Because of our focus on simpler algorithms with few render passes, we have not looked extensively at the combination of warping with adaptive partitioning. Some existing algorithms apply quadtree refinement to a warped shadow map. Giegl and Wimmer (2007a,

2007b) report improvements by creating a quadtree refinement of an LiSPSM. We have performed some preliminary experiments with combining warping with RMSMs (Lefohn et al., 2007). We expected to see a drop in the number of partitions required. Surprisingly, the number of partitions actually went up. The warping shifts samples away from the far end of the view frustum towards the viewer. This means that the distant areas need to be refined to a higher level in the quadtree than before, while areas near the viewer do not need to be refined as much. It seems that because the distant areas cover a larger spatial extent, more new partitions are required than are saved by warping. But this seems to conflict with the results of Giegl and Wimmer. It might be that we observe these results because RMSMs use a much finer granularity (32×32 resolution shadow maps vs. $2K \times 2K$ resolution). We would like to conduct a more thorough investigation and provide more theoretical analysis in order to understand exactly what is happening. Combinations of warping and adaptive partitioning are of particular interest because these algorithms can handle both perspective and projective aliasing.

LogPSMs and existing warping algorithms are scene independent. These algorithms assume that the scene geometry can appear anywhere inside the view frustum. However, it is often the case that large regions of the view frustum are empty. We have shown that a pseudo-near plane can be used to provide rough information about the expected location of objects in the scene, which can be used to further optimize the shadow map warping. Currently we set the pseudo-near plane manually. We would like to investigate algorithms that set the pseudo-near plane adaptively by computing an approximation of the closest visible surface in the scene.

In this thesis we have analyzed the various algorithms in terms of maximum error. But other error metrics are possible. No single algorithm for choosing the warping parameter will be sufficient for all applications. As mentioned in Chapter 5, we would like to create a tool that could be used to interactively manipulate z-partition locations and/or warping parameter shaping curves to select error profiles across the view frustum that are best suited for a user's particular needs.

More analysis needs to be done on the hardware modifications we propose for logarithmic

rasterization. We would like to build a prototype of the system in order to obtain more detailed performance data.

Log rasterization is part of a larger context of greater generalization for the rasterizer. A number of important lighting and rendering effects can be implemented with nonlinear rasterization, such as reflections, refraction, caustics, multi-perspective rendering (Hou et al., 2006), and omnidirectional shadows (Brabec et al., 2002b). Since we have seen increased programmability in just about every other part of the GPU in recent years, it would be interesting to look at the possibility of a programmable rasterizer that could handle these effects in a more general way.

LogPSMs can provide significant improvements over existing warping algorithms, but more work remains to be done to provide shadows that are artifact-free. Because the constraints of real-time graphics remain the same (30-60 frames per second) and display resolutions seem to increase fairly slowly, we can hope that the ever increasing computational power of graphics hardware will eventually enable the use of more sophisticated shadowing algorithms with real-time performance. Until then, high quality shadow generation for real-time applications remains a challenge and continues to be an area of open research.

APPENDIX A

Light image plane orientation

Figure A.1 shows a 1D shadow map in simple 2D scene. The domain of the shadow map is defined over a range $[v_n, v_f]$ on the light image plane (line) $\boldsymbol{\pi}_l$. The points z_c, z_n, and z_f on a line $\boldsymbol{\pi}$ correspond to v_c, v_n, and v_f on $\boldsymbol{\pi}_l$. The mapping of an arbitrary point z on $\boldsymbol{\pi}$ to a point v on $\boldsymbol{\pi}_l$ is a 1D projective transformation. In general, a 1D projective mapping from one line to another can be computed by utilizing the fact that under a projective mapping the cross ratio of any four corresponding points on the lines is preserved. The cross ratio can be defined as:

$$[x_1, x_2, x_3, x_4] = \frac{(x_1 - x_2)(x_3 - x_4)}{(x_1 - x_4)(x_3 - x_2)}. \tag{A.1}$$

Therefore the mapping from z to v can be computed by solving $[v, v_n, v_f, v_c] = [z, z_n, z_f, z_c]$ for v in terms of z. The point v is mapped to a point $t \in [0, 1]$ in the shadow map by a function $F(v)$. Suppose $F(v)$ is also a projective function. To compute $F(v)$ we can solve the following equation for t in terms of v:

$$[t, t_n, t_f, t_c] = [v, v_n, v_f, v_c], \tag{A.2}$$

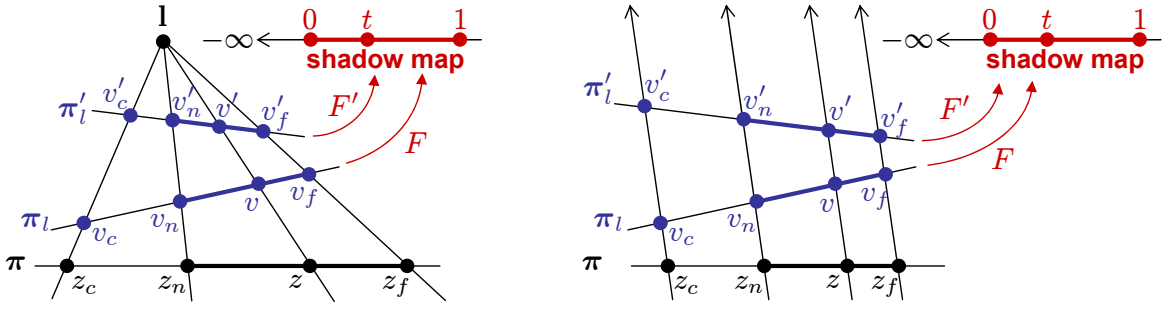

Figure A.1: ***Varying the orientation of the light image plane.*** *(Left) Point light. (Right) Directional light (point light at infinity).*

with $t_n = 0$ and $t_f = 1$. But what should t_c be? We observe that the cross ratio $[t, 0, 1, \pm\infty]$ is the identity function. If we choose $t_c = \pm\infty$, the left hand side of Equation A.2 becomes t. Thus $F(v)$ is simply the cross ratio $[v, v_n, v_f, v_c]$:

$$\begin{aligned} F(v) &= [v, v_n, v_f, v_c] \\ &= \frac{(v - v_n)(v_f - v_c)}{(v - v_c)(v_f - v_n)}. \end{aligned} \tag{A.3}$$

Choosing $t_c = -\infty$ preserves the relative orientation of v_c, v_n, and v_f under F. From this we can also see that if $v_c < v_n < v_f$, then the cross ratio $[v, v_n, v_f, v_c]$ can be thought of as the function that maps v_n to 0, v_f to 1 and v_c to $-\infty$.

The composite mapping from z to v and v to t is also a projective transformation given by $t = [z, z_n, z_f, z_c]$. Now consider a light image plane $\boldsymbol{\pi}'_l$ with a different orientation than $\boldsymbol{\pi}_l$. So long as v'_c, v'_n, and v'_f are chosen to lie on the lines $\overline{\mathbf{p}_l v_c}$, $\overline{\mathbf{p}_l v_n}$, and $\overline{\mathbf{p}_l v_f}$ respectively, then z_c, z_n, and z_f remain unchanged and the overall mapping from z to t in the shadow map does not change. This allows us to pick a convenient light plane orientation and location for analyzing aliasing error and another orientation when fitting the light frustum to the scene without changing the overall warping of the shadow map. The same is true of 2D shadow maps in 3D.

When $\mathbf{p}_l$ lies on the line at infinity, the lines $\overline{\mathbf{p}_l v_c}$, $\overline{\mathbf{p}_l v_n}$, and $\overline{\mathbf{p}_l v_f}$ become parallel. The relative distance between corresponding points on $\boldsymbol{\pi}_l$ and $\boldsymbol{\pi}'_l$ differ by a constant and $[v, v_n, v_f, v_c] = [v', v'_n, v'_f, v'_c]$. In other words, F and F' are the same. This means that for a light at ∞, changing the orientation of the light image plane effects neither the overall warping nor the mapping from the light image plane to the shadow map.

In Equation A.3, v_n and v_f are fixed by the extents of the scene to be included in the shadow map. The projective warping is determined by the location of v_c alone. v_c is the center of projection of the 1D projective transformation that controls the warping. When

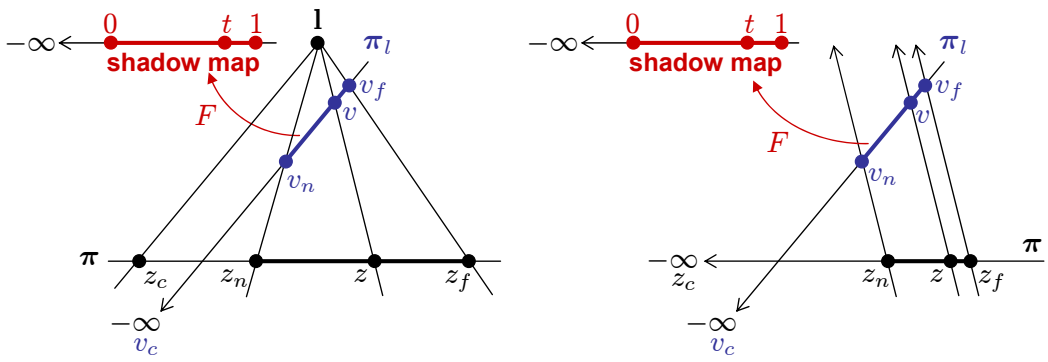

Figure A.2: ***Affine mapping to shadow map.*** *With $v_c = -\infty$, the mapping F from the light image plane to the shadow map becomes affine. (Left) For a point light, the overall warping function can be non-affine. (Right) For a directional light, the overall warping function must also be affine.*

$v_c = \pm\infty$, $F(v)$ becomes:

$$\begin{aligned} F(v) &= [v, v_n, v_f, \pm\infty] \\ &= \frac{v - v_n}{v_f - v_n}. \end{aligned} \tag{A.4}$$

This affine transformation is the same as the uniform parameterization used by a standard shadow map. Chong (2003) formulates shadow map warping in terms of the angle of the light image plane of a standard shadow map with respect to the light's view direction. As shown in Figure A.2, an infinite v_c can produce a finite z_c when the light is not at infinity, and thus a nonuniform projective mapping from z to t. However, when the light is at infinity, z_c is also infinite. In this case the overall transformation from z to t becomes affine and does not induce a nonuniform warping. Therefore, it is more general to use v_c to control the warping rather than image plane angle.

APPENDIX B

Analysis of generalized polygon offset approximation

We analyze the maximum error of our generalized polygon offset approximation using Figure B.1. The functions f_1 and f_2 are the depth slopes $|\partial z/\partial x|$ and $|\partial z/\partial y|$, respectively. f_1 has a constant value c and f_2 varies linearly from a to b. f_3 is our approximation from Equation 7.13 with $m_{z0} = a$ and $m_{z1} = c$:

$$f_1(y) = c \tag{B.1}$$

$$f_2(y) = \frac{a(y_1 - y) + b(y - y_0)}{(y_1 - y_0)} \tag{B.2}$$

$$f_3(y) = \frac{a(y_1 - y) + c(y - y_0)}{(y_1 - y_0)}. \tag{B.3}$$

The maximum difference d between the standard polygon offset, $\max(f_1, f_2)$, and our approximation, f_3, occurs at y_i where $f_1 = f_2$:

$$d = f_3(y_i) - c = \frac{(a - c)(c - b)}{a - b} \tag{B.4}$$

$$y_i = \frac{y_0(c - b) + y_1(a - c)}{a - b}. \tag{B.5}$$

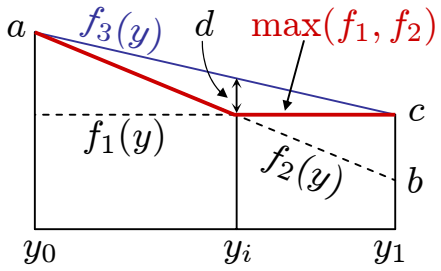

Figure B.1: ***Polygon offset approximation.***

The maximum relative difference $\max_c(d/c)$ occurs when $c = \sqrt{ab}$. Plugging this value we get:

$$\max_c \left(\frac{d}{c}\right) = \frac{\sqrt{a} - \sqrt{b}}{\sqrt{a} + \sqrt{b}}. \tag{B.6}$$

The maximum relative difference is largest when the difference between a and b is largest, which occurs when $y_0 = 0$ and $y_1 = 1$. Plugging these values into Equation 7.12 we get:

$$a = \left|\frac{\partial z}{\partial y}\right| \left|\frac{1}{c_0 c_1}\right| = \left|\frac{\partial z}{\partial y}\right| \frac{f/n \log(f/n)}{f/n - 1} \tag{B.7}$$

$$b = \left|\frac{\partial z}{\partial y}\right| \left|\frac{c_1 + 1}{c_0 c_1}\right| = \left|\frac{\partial z}{\partial y}\right| \frac{\log(f/n)}{f/n - 1} \tag{B.8}$$

$$\max_c \left(\frac{d}{c}\right) = \frac{\sqrt{f/n} - 1}{\sqrt{f/n} + 1}. \tag{B.9}$$

Large values of f/n give the maximum possible relative difference, which approaches 1.

BIBLIOGRAPHY

Aila, T. & Laine, S. (2004a). Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004* (pp. 161–166).: Eurographics Association.

Aila, T. & Laine, S. (2004b). Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004* (pp. 161–166).: Eurographics Association.

Akenine-Möller, T., Munkberg, J., & Hasselgren, J. (2007). Stochastic rasterization using time-continuous triangles. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (pp. 7–16).: Eurographics Association.

Annen, T., Mertens, T., Bekaert, P., Seidel, H.-P., & Kautz, J. (2007). Convolution shadow maps. In J. Kautz & S. Pattanaik (Eds.), *Rendering Techniques 2007: Eurographics Symposium on Rendering*, volume 18 of *Eurographics / ACM SIGGRAPH Symposium Proceedings* (pp. 51–60). Grenoble, France: Eurographics.

Arvo, J. (2004). Tiled shadow maps. In *Proceedings of Computer Graphics International 2004* (pp. 240–247).: IEEE Computer Society.

Brabec, S., Annen, T., & Seidel, H.-P. (2002a). Practical shadow mapping. *Journal of Graphics Tools*, 7(4), 9–18.

Brabec, S., Annen, T., & Seidel, H.-P. (2002b). Shadow mapping for hemispherical and omnidirectional light sources. In J. Vince & R. Earnshaw (Eds.), *Advances in Modelling, Animation and Rendering (Proceedings Computer Graphics International 2002)* (pp. 397–408). Bradford, UK: Springer.

Chan, E. & Durand, F. (2004). An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering* (pp. 185–195).: Eurographics Association.

Chong, H. (2003). *Real-Time Perspective Optimal Shadow Maps.* Senior Thesis, Harvard University.

Chong, H. & Gortler, S. (2004). A lixel for every pixel. In *Proceedings of the Eurographics Symposium on Rendering* (pp. 167–172).: Eurographics Association.

Chong, H. & Gortler, S. (2006). *Scene Optimized Shadow Mapping.* Technical Report TR-11-06, Harvard University.

Cook, R. L. (1986). Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1), 51–72.

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., & Knuth, D. E. (1996). On the lambert w function. *Advances in Computational Mathematics*, 5, 329–359.

Crow, F. C. (1977). Shadow algorithms for computer graphics. *ACM Computer Graphics*, 11(3), 242–248.

Crow, F. C. (1984). Summed-area tables for texture mapping. In H. Christiansen (Ed.), *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18 (pp. 207–212).

Dally, W. J. & Poulton, J. W. (1998). Digital systems engineering. In *Cambridge University Press*.

DeRoo, J., Morein, S., Favela, B., & Wright, M. (2002). Method and apparatus for compressing parameter values for pixels in a display frame. *US Patent 6,476,811*.

Donnelly, W. & Lauritzen, A. (2006). Variance shadow maps. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (pp. 161–165). New York, NY, USA: ACM Press.

Engel, W. (2007). Cascaded shadow maps. In W. Engel (Ed.), *ShaderX*^{5} (pp. 197–206). Charles River Media.

Fernando, R., Fernandez, S., Bala, K., & Greenberg, D. (2001). Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001* (pp. 387–390).

Forsyth, T. (2006). Making shadow buffers robust using multiple dynamic frustums. In W. Engel (Ed.), *ShaderX*^{4} (pp. 331–346). Charles River Media.

Giegl, M. & Wimmer, M. (2007a). Fitted virtual shadow maps. In *Proceedings of Graphics Interface 2007* (pp. 159–168).

Giegl, M. & Wimmer, M. (2007b). Queried virtual shadow maps. In *Proceedings of ACM SIGGRAPH 2007 Symposium on Interactive 3D Graphics and Games* (pp. 65–72).: ACM Press.

Glassner, A. S. (1995). *Principles of Digital Image Synthesis*. Morgan Kaufman Publishers, Inc.

Govindaraju, N., Lloyd, B., Yoon, S., Sud, A., & Manocha, D. (2003). Interactive shadow generation in complex environments. *Proc. of ACM SIGGRAPH/ACM Trans. on Graphics*, 22(3), 501–510.

Grant, C. W. (1992). *Visibility Algorithms in Image Synthesis*. PhD thesis, University of California, Davis.

Hasselgren, J. & Akenine-Möller, T. (2006). Efficient depth buffer compression. In *Proceedings of Graphics Hardware 2006* (pp. 103–110).: Eurographics Association.

Hou, X., Wei, L.-Y., Shum, H.-Y., & Guo, B. (2006). Real-time multi-perspective rendering on graphics hardware. In *Proceedings of the Eurographics Symposium on Rendering 2006* (pp. 93–102).: Eurographics Association.

Hourcade, J. C. & Nicolas, A. (1985). Algorithms for antialiased cast shadows. *Computers and Graphics*, 9(3), 259–265.

Hu, H. H., Gooch, A. A., Thompson, W. B., Smits, B. E., Rieser, J. J., & Shirley, P. (2000). Visual cues for imminent object contact in realistic virtual environment. In *Proceedings of the Conference on Visualization '00* (pp. 179–185). Los Alamitos, CA, USA: IEEE Computer Society Press.

Hubona, G. S., Wheeler, P. N., Shirah, G. W., & Brandt, M. (1999). The relative contributions of stereo, lighting, and background scenes in promoting 3d depth visualization. *ACM Transactions on Computer-Human Interaction*, 6(3), 214–242.

Johnson, D. E. & Cohen, E. (2004). Unified distance queries in a heterogeneous model environment. In *ASME DETC*.

Johnson, G., Mark, W., & Burns, C. (2004). The irregular z-buffer and its application to shadow mapping. In *The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-04-09*.

Johnson, G. S., Lee, J., Burns, C. A., & Mark, W. R. (2005). The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph.*, 24(4), 1462–1482.

Kersten, D., Knill, D., Mamassian, P., & Bülthoff, I. (1996). Illusory motion from shadows. *Nature*, 379, 31.

Kersten, D., Mamassian, P., & Knill, D. (1997). Moving cast shadows induce apparent motion in depth. *Perception*, 26, 171–192.

Kozlov, S. (2004). Perspective shadow maps: Care and feeding. In R. Fernando (Ed.), *GPU Gems* (pp. 214–244). Addison-Wesley.

Lefohn, A. E., Sengupta, S., & Owens, J. D. (2007). Resolution matched shadow maps. *ACM Transactions on Graphics*, 26(4), 20:1–20:17.

Lloyd, B., Tuft, D., Yoon, S., & Manocha, D. (2006). Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006* (pp. 215–226).: Eurographics Association.

Martin, T. & Tan, T.-S. (2004). Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (pp. 153–160).: Eurographics Association.

McCool, M. (2000). Shadow volume reconstruction from depth maps. *ACM Trans. on Graphics*, 19(1), 1–26.

McCool, M., Wales, C., & Moule, K. (2001). Incremental and hierarchical hilbert order edge equation using polygon rasteization. *Eurographics Workshop on Graphics Hardware*, (pp. 65–72).

McCormack, J. & McNamara, R. (2000). Tiled polygon traversal using half-plane edge functions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (pp. 15–21).

NVIDIA Corp. (2004). UltraShadowII: Accelerated shadow calculations. *Technical Brief*.

NVIDIA Corp. (2006). NVIDIA GeForce 8800 GPU architecture overview. *Technical Brief*.

Olano, M. & Greer, T. (1997). Triangle scan conversion using 2d homogeneous coordinates. *Eurographics Workshop on Graphics Hardware*, (pp. 89–95).

Ornstein, D., Peled, G., Sperber, Z., Cohen, E., & Malka, G. (2005). Z-compression mechanism. *US Patent 6,580,427*.

Pineda, J. (1988). A parallel algorithm for polygon rasterization. In J. Dill (Ed.), *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22 (pp. 17–20).

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press.

Reeves, W., Salesin, D., & Cook, R. (1987). Rendering antialiased shadows with depth maps. In *Computer Graphics (ACM SIGGRAPH '87 Proceedings)*, volume 21 (pp. 283–291).

Scherzer, D., Jeschke, S., & Wimmer, M. (2007). Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In J. Kautz & S. Pattanaik (Eds.), *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)* (pp. 45–50).: Eurographics.

Schüler, C. (2006). Eliminating surface acne with gradient shadow mapping. In W. Engel (Ed.), *ShaderX*^{4} (pp. 289–297). Charles River Media.

Segal, M. & Akeley, K. (2006). The OpenGL graphics system: A specification. *http://www.opengl.org/*.

Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., & Haeberli, P. (1992). Fast shadows and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26 (pp. 249–252).

Sen, P., Cammarano, M., & Hanrahan, P. (2003). Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3), 521–526.

Shoemaker, B. (2003). John Carmack speaks on doom 3. *GameSpot, http://www.gamespot.com/pc/action/doom3/news.html?sid=6027989*.

Stamminger, M. & Drettakis, G. (2002). Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002* (pp. 557–562).

Tadamura, K., Qin, X., Jiao, G., & Nakamae, E. (1999). Rendering optimal solar shadows using plural sunlight depth buffers. In *Computer Graphics International 1999* (pp. 166).

Van Dyke, J. & Margeson, J. (2005). Method and apparatus for managing and accessing depth data in a computer graphics system. *US Patent 6,961,057*.

Wald, I., Mark, W. R., Günther, J., Boulos, S., Ize, T., Hunt, W., Parker, S. G., & Shirley, P. (2007). State of the Art in Ray Tracing Animated Scenes. In *Eurographics 2007 State of the Art Reports* (pp. To appear).

Wang, Y. & Molnar, S. (1994). *Second-Depth Shadow Mapping*. Technical Report TR94-019, University of North Carolina at Chapel Hill.

Wanger, L. (1992). The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics* (pp. 39–42). New York, NY, USA: ACM Press.

Weiskopf, D. & Ertl, T. (2003). Shadow Mapping Based on Dual Depth Layers. In *Proceedings of Eurographics '03 Short Papers* (pp. 53–60).

Whitted, T. (1979). An improved illumination model for shaded display. volume 13 (pp. 1–14).

Williams, L. (1978). Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12 (pp. 270–274).

Williams, L. (1983). Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17 (pp. 1–11).

Wimmer, M., Scherzer, D., & Purgathofer, W. (2004). Light space perspective shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (pp. 143–152).: Eurographics Association.

Woo, A. (1992). The shadow depth map revisited. In D. Kirk (Ed.), *Graphics Gems III* (pp. 338–342, 582). Boston, MA: Academic Press.

Woop, S., Schmittler, J., & Slusallek, P. (2005). RPU: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph.*, 24(3), 434–444.

Zhang, F., Sun, H., Xu, L., & Lun, L. K. (2006a). Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of ACM International Conference on Virtual Reality Continuum and Its Applications 2006* (pp. 311–318).: ACM SIGGRAPH.

Zhang, F., Xu, L., Tao, C., & Sun, H. (2006b). Generalized linear perspective shadow map reparameterization. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications* (pp. 339–342). New York, NY, USA: ACM Press.