Logarithmic Perspective Shadow Maps

D. BRANDON LLOYD[1,2], NAGA K. GOVINDARAJU[2], CORY QUAMMEN[1],
STEVEN E. MOLNAR[3], and DINESH MANOCHA[1]
[1] University of North Carolina at Chapel Hill
[2] Microsoft Corporation
[3] NVIDIA Corporation

We present a novel shadow map parameterization to reduce perspective aliasing artifacts for both point and directional light sources. We derive the aliasing error equations for both types of light sources in general position. Using these equations we compute tight bounds on the aliasing error. From these bounds we derive our shadow map parameterization, which is a simple combination of a perspective projection with a logarithmic transformation. We formulate several types of logarithmic perspective shadow maps (LogPSMs) by replacing the parameterization of existing algorithms with our own. We perform an extensive error analysis for both LogPSMs and existing algorithms. This analysis is a major contribution of this paper and is useful for gaining insight into existing techniques. We show that compared with competing algorithms, LogPSMs can produce significantly less aliasing error. Equivalently, for the same error as competing algorithms, LogPSMs can produce significant savings in both storage and bandwidth. We demonstrate the benefit of LogPSMs for several models of varying complexity.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: shadow maps, antialiasing

1. INTRODUCTION

The shadow map algorithm [Williams 1978] is a popular approach for hard shadow computation in interactive applications. It uses a depth buffer rendered from the viewpoint of the light to compute shadows in an image. Shadow maps are relatively easy to implement, deliver good performance on commodity GPUs, and can handle complex, dynamic scenes. Other alternatives for hard shadows, such as shadow volumes or ray tracing, can produce high-quality shadows, but may exhibit poor performance on complex models or dynamic scenes.

A major disadvantage of shadow maps is that they are prone to aliasing artifacts which give rise to jagged shadow edges. Aliasing error can be classified as perspective or projection aliasing [Stamminger and Drettakis 2002]. Perspective aliasing on a surface depends on its position of a surface relative to the light and eye, and projection aliasing depends on the surface orientation. Possible solutions to overcome both kinds of aliasing include very high resolution shadow maps, adaptive shadow maps that support local variation in shadow map resolution, or irregularly sampled shadow maps. However, these techniques have major disadvantages; they can have high memory bandwidth and storage costs, they can be difficult to implement efficiently within the current graphics pipeline, and/or they can require substantial changes to the current rasterization architectures.

Among the fastest shadow mapping solutions are those that reduce perspective aliasing by reparameterizing the shadow map to allocate more samples to the undersampled regions of a scene. Several warping algorithms, such as perspective shadow maps (PSMs) [Stamminger and Drettakis 2002] and their variants [Wimmer et al. 2004; Martin and Tan 2004], have been proposed to alter a shadow map's sample distribution. However,

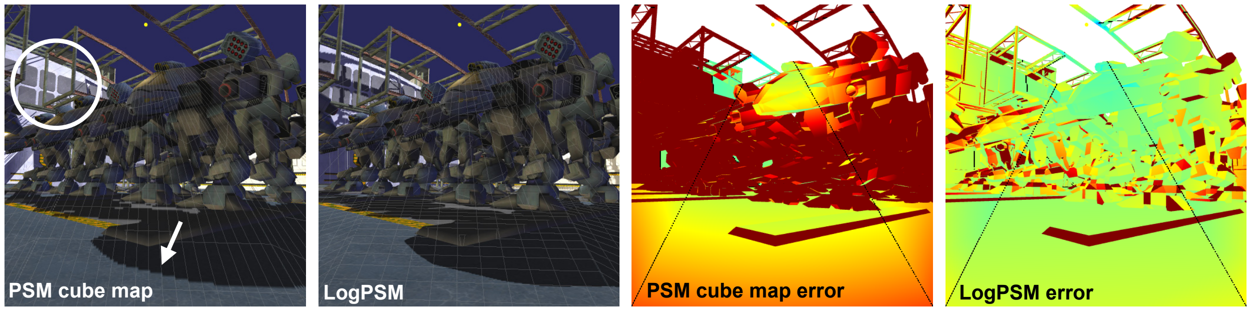


Fig. 1: *Night-time scene of robots in a hangar with a point light. We compare our algorithm (LogPSM) to Kozlov's improved perspective shadow map (PSM) algorithm [Kozlov 2004]. Both algorithms use cube maps whose faces have a combined resolution of about* 1024×1024*. The images have a resolution of* 512×512*. (Left) Compared to a standard cube map, the PSM cube map greatly reduces aliasing artifacts near the viewer, but some aliasing is still visible. The shadows are severely stretched on the back wall. LogPSMs provide higher quality both near the viewer and in the distance. The shadow map grid has been superimposed to aid visualization (grid lines every 20 texels). (Right) An error visualization for both algorithms. We use an error metric, m, that is essentially the area of a shadow map texel projected into the image. Green represents no aliasing (*$m = 1$*) and dark red (*$m > 11$*) represents high aliasing. LogPSMs provide significantly lower maximum error and the error is more evenly distributed.*

the parameterization used by these algorithms can produce a poor approximation to the optimal sample distribution. Therefore existing warping algorithms can still require high shadow map resolution to reduce aliasing. Recent work suggests that a logarithmic parameterization is closer to optimal [Wimmer et al. 2004; Lloyd et al. 2006; Zhang et al. 2006]. Other algorithms produce a discrete approximation of a logarithmic parameterization by partitioning the view frustum along the view vector and using a separate shadow map for each sub-frustum [Tadamura et al. 1999; Zhang et al. 2006; Engel 2007]. To reduce the error to acceptable levels, however, these algorithms can require a large number of partitions, which hurts performance.

Main Results

In this paper, we present logarithmic perspective shadow maps (LogPSMs). LogPSMs are extensions of existing perspective warping algorithms which use a small number of partitions in combination with our improved warping function to achieve significantly less error than competing algorithms. Some of the novel aspects of our work include:

(1) **Error analysis for general configurations.** We compute aliasing error for point and directional lights in general configurations. The error analysis performed in previous work has typically been restricted to directional lights in a few special configurations [Stamminger and Drettakis 2002; Wimmer et al. 2004; Zhang et al. 2006]. The methods used in this analysis can be used for evaluating future algorithms, as well as for gaining insight into existing ones.

(2) **LogPSM parameterization.** Based on the error analysis, we derive a new shadow map parameterization with tight bounds on the perspective aliasing error. We show that the error is $O(\log(f/n))$ where f/n is the ratio of the far to near plane distances of the view frustum. In contrast, the error of existing warping algorithms is $O(f/n)$. The parameterization is a simple combination of a logarithmic transformation with a perspective projection.

(3) **LogPSM algorithms.** Our algorithms are the first to use a continuous logarithmic parameterization and can produce high quality shadows with less shadow map resolution than that required by similar algorithms.

We also discuss an improvement to the LiSPSM algorithm, as well as various implementation details of LogPSMs that are also useful for other shadow map algorithms. We perform a detailed comparison of LogPSMs with other algorithms using several different techniques. We demonstrate significant error reductions on several models of varying complexity.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents our derivation of the aliasing error equations. In Section 4 we use these equations to derive tight bounds on the error. In Section 5 we derive the LogPSM parameterization. Section 6 describes the various LogPSM algorithms. We present our results and comparisons with other algorithms in Section 7 and then conclude with some ideas for future work.

2. PREVIOUS WORK

In this section we briefly review the approaches currently used to handle shadow map aliasing. The approaches can be classified as sample redistribution techniques, improved reconstruction techniques, and hybrid approaches. Besides shadow mapping, other algorithms for hard shadow generation include ray tracing and shadow volume algorithms. However, these approaches are currently unable to handle complex dynamic environments at high resolution and high frame rates.

2.1 Sample redistribution techniques

Focussing the shadow map on just the visible portion of the scene [Brabec et al. 2002] or increasing the resolution reduces aliasing error by uniformly increasing the shadow map sampling density. Further error reductions can be obtained by redistributing the samples nonuniformly, placing more samples where the aliasing error is highest. We classify sample redistribution techniques into several categories: warping, partitioning, combinations of warping and partitioning, and irregular sampling.

Warping. Warping techniques use a nonuniform parameterization of the shadow map to redistribute the samples. The use of warping to handle aliasing was introduced with perspective shadow maps (PSMs) [Stamminger and Drettakis 2002]. A PSM is created by rendering the scene in the post-perspective space of the camera. Light-space perspective shadow maps (LiSPSMs) [Wimmer et al. 2004] are a generalization of PSMs which avoid some of the problems of rendering in post-perspective space. Trapezoidal shadow maps (TSMs) [Martin and Tan 2004] are similar to LiSPSMs, except that the parameter for the perspective projection is computed with a heuristic. Chong and Gortler [2007] propose an optimization framework for computing a perspective projection that minimizes aliasing in the image.

Partitioning. Partitioning techniques use multiple shadow maps to provide local control over the sample distribution. Plural sunlight buffers [Tadamura et al. 1999] and cascaded shadow maps [Engel 2007] use a simple partitioning approach that splits the frustum along the view vector at intervals that increase geometrically with distance from the eye. Parallel-split shadow maps [Zhang et al. 2006] also partition along the view vector, but use a combination of geometric and uniform distances to compute the split locations. Adaptive shadow maps [Fernando et al. 2001] and resolution matched shadow maps [Lefohn et al. 2007] represent the shadow map as a quadtree of fixed resolution tiles. While the quadtree-based approaches can deliver high quality, they can also require a large number of render passes. Tiled shadow maps [Arvo 2004] partition a single, fixed-resolution shadow map into tiles of different sizes guided by an error measurement heuristic. Forsyth [2006] proposes a partitioning technique that uses a greedy clustering of objects into multiple shadow frusta.

Warping + partitioning. Warping can be combined with multiple shadow map partitions. Chong [2003] presents an algorithm for 2D flatland which performs perspective warping combined with partitioning. 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, which can be considered a form of partitioning. Kozlov [2004] uses a cube map to render the shadow map in the post-perspective space of the camera. Queried virtual shadow maps [Giegl and Wimmer 2007b] and fitted virtual shadow maps [Giegl and Wimmer 2007a] combine LiSPSMs with an adaptive partitioning scheme. Lloyd et al. [2006] analyze various combinations of warping and partitioning and suggest that a combination of LiSPSMs with cascaded shadow maps gives the lowest error.

Irregular sampling. Instead of inferring sample locations from a regular grid using a parameterization, irregular shadow maps [Johnson et al. 2004; Aila and Laine 2004] sample explicitly at the locations from which the shadow map is queried during image rendering. The results are equivalent to ray tracing. Because GPUs are heavily optimized for rasterization and coherent access to samples stored on a regular grid, irregular shadow maps can be difficult to implement efficiently on current graphics architectures. Several implementations exists [Arvo 2007; Sintorn et al. 2008], but their performance is significantly less than that of the simpler warping techniques. A hardware architecture for implementing irregular shadow maps has been proposed [Johnson et al. 2005], but requires significant modifications of current GPUs in order to support efficient data scattering. The recently proposed Larrabee architecture has better support for irregular data structures [Seiler et al. 2008]. Irregular shadow maps have been implemented on the architecture, but it is still unclear how well they perform relative to regular rasterization because no absolute performance numbers have been released.

2.2 Filtering techniques

Two sampling processes are involved in shadow mapping – the initial sampling of the scene used to render the shadow map, and the sampling of the shadow map that occurs when rendering the image. None of the aforementioned techniques directly address aliasing that occurs while rendering the shadow map. Aliasing during image rendering can occur in the distance away from the viewer where the shadow map is under-sampled. Filtering techniques, such as mip-mapping or summed area tables, are typically used to handle these kinds of artifacts. The jagged shadow edges addressed by this paper arise from the use of nearest-neighbor sampling when reconstructing the visibility "signal" from the sampled scene representation stored in the shadow map. A more accurate reconstruction filter (e.g. bilinear interpolation) replaces the jagged edges with a blurred edged. On one hand this can sometimes be desirable because it resembles soft-shadow penumbra, although it is not physically correct. On the other hand, the blur can be excessive, in which case a higher initial sampling frequency is required. When the frequency is high enough, nearest-neighbor sampling can be used without introducing jagged edges. The higher sampling frequency can be obtained by increasing the shadow map resolution or by sample redistribution. Filtering and sample redistribution are orthogonal and can often be used together.

Standard filtering techniques, such as interpolation and mip-mapping, cannot be applied directly to a depth-based shadow map. Percentage closer filtering (PCF) [Reeves et al. 1987] first computes visibility at sample locations and then applies a filter to the results. Several algorithms use either a statistical representation of depth [Donnelly and Lauritzen 2006; Lauritzen and McCool 2008; Salvi 2008] or a representation of the visibility function [Annen et al. 2007; Annen et al. 2008] to which standard texture filtering techniques can be applied. Scherzer et al. [2007] reuse the shadowing information from previous frames to provide both temporal smoothing and more accurate reconstruction of hard shadow edges.

2.3 Hybrid approaches

Hybrid techniques combine image-based shadow maps with object-space methods. Silhouette shadow maps [Sen et al. 2003] reconstruct a more accurate shadow edge by augmenting the shadow map with extra geometric information about the location of the silhouette edges of shadow casters. The shadow volume algorithm [Crow 1977] is a popular object-space method that does not suffer from aliasing artifacts, but can require very high fill rate to maintain performance. Shadow maps have been combined with shadow volumes to address the fill rate problems. McCool [2001] constructs a shadow volume from edges in a shadow map, yielding a simplified shadow volume. Chan and Durand [2004] use a shadow map to limit shadow volume rendering to only the regions that are near shadow boundaries.

3. SHADOW MAP ALIASING ERROR

In this section we describe how to quantify aliasing error and derive the equations for the error in a 2D scene. 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 and Drettakis 2002; Wimmer et al. 2004; Lloyd et al. 2006]. A notable exception [Zhang et al. 2006] computes perspective aliasing error for directional lights over a range of angles, but only along a single line through the view frustum. After deriving the equations for 2D, we extend our analysis to 3D.

3.1 Quantifying aliasing error

The aliasing error that results in jagged shadow edges is caused when the sampling frequency used to render the shadow map from the viewpoint of the light is lower than the frequency at which the shadow map is sampled when rendering the image from the viewpoint of the eye. Ideally, the eye sample locations should coincide exactly with the light sample locations, as with ray tracing or the irregular z-buffer, in which case the sampling frequency match implicitly. For our derivations we choose to work with the spacing between samples because it is more intuitive geometrically than frequency. Figure 2 shows how the spacing between a pair of eye and light samples changes as they are traced through through a simple 2D scene. Aliasing error occurs when the light sample spacing is greater than the eye sample spacing. If we define $j \in [0, 1]$ and $t \in [0, 1]$ as normalized image and shadow map coordinates, then aliasing error can be quantified as:

$$m = \frac{r_j}{r_t} \frac{\mathrm{d}j}{\mathrm{d}t}, \tag{1}$$

where r_j and r_t are the image and shadow map resolutions.[1] Aliasing occurs when $m > 1$.

3.2 Deriving aliasing error

To derive $\mathrm{d}j/\mathrm{d}t$ in Equation 1, we first compute j as a function of t from Figure 2 by tracing a sample from the shadow map to its corresponding location in the image. We begin by introducing some notation. A point $\mathbf{p}$ and a vector $\vec{\mathbf{v}}$ are expressed as a 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 $\hat{\mathbf{n}}$. $\boldsymbol{\pi}\mathbf{p}$ gives the signed distance of $\mathbf{p}$ from the plane and $\boldsymbol{\pi}\hat{\mathbf{v}} = \hat{\mathbf{n}} \cdot \hat{\mathbf{v}} = \cos\alpha$, where α is the angle between $\hat{\mathbf{n}}$ and $\hat{\mathbf{v}}$. Please refer to Table I for the meaning of specific symbols used in this section.

The inverse of the shadow map parameterization F is a function G that 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:

$$\mathbf{p}_l = \mathbf{p}_{l0} + vW_l\hat{\mathbf{y}}_l, \tag{2}$$

[1]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.

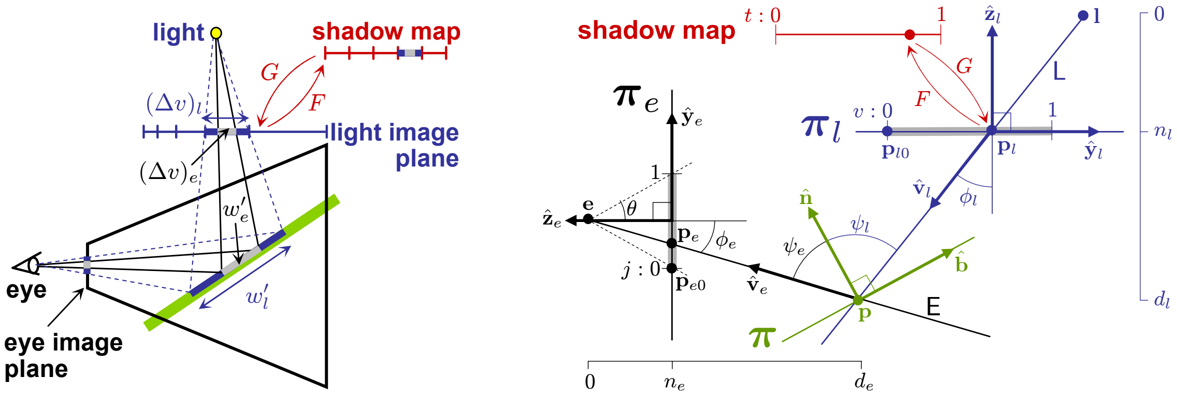


Fig. 2: ***Computing aliasing error.*** *(Left) Spacing between locations corresponding to a pair of samples from the light (shadow map texels) and the eye (image pixels). Aliasing error can be quantified as the ratio of the spacing between the light and eye sample locations. (Right) The sample spacing is related to the derivatives of the function $j(t)$ 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 $\boldsymbol{\pi}$ in the scene, and projects $\mathbf{p}$ through the eye to a point $\mathbf{p}_e$ on the eye's image plane $\boldsymbol{\pi}_e$.*

F, G	shadow map parameterization and its inverse	ϕ_l, ϕ_e	angle between image plane normals and rays of light and eye
$(i,j)^\top$	eye image plane coordinates	ψ_l, ψ_e	angle between surface normal and light and eye rays
$(s,t)^\top$	shadow map coordinates	β_l, β_e	angle between surface normal and light and eye image planes
$(u,v)^\top$	light image plane coordinates	d_l, d_e	distance along $\hat{\mathbf{z}}_l$ and $\hat{\mathbf{z}}_e$ from light and eye to a point in the scene
$r_i \times r_j$	image resolution	n_l, n_e	distance from light and eye to respective image plane
$r_s \times r_t$	shadow map resolution	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 axes	w_l, w_e	widths of light and eye beams
$\hat{\mathbf{x}}_e, \hat{\mathbf{y}}_e, \hat{\mathbf{z}}_e$	eye space coordinate axes	w'_l, w'_e	widths of light and eye beams projected on a surface
$\hat{\mathbf{n}}$	surface normal	$w_e^{\perp \hat{\mathbf{z}}_e}$	width of an eye beam measured perpendicular to $\hat{\mathbf{z}}_e$
$\hat{\mathbf{b}}$	vector along the surface	$(\Delta v)_l, (\Delta v)_e$	sample spacing in v on light image plane
$\boldsymbol{\pi}_l, \boldsymbol{\pi}_e$	light and eye image planes	δ_l, δ_e	spacing distribution function for the light and eye
$\boldsymbol{\pi}$	a planar surface in the scene	$\tilde{\delta}_e$	perspective factor of δ_e
$\mathbf{p}_{l0}, \mathbf{p}_{e0}$	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		
$\mathbf{p}$	point on a surface		
$\mathbf{l}, \mathbf{e}$	light and eye positions		
L, E	rays from the light and eye		
$\hat{\mathbf{v}}_l, \hat{\mathbf{v}}_l$	direction of rays from light and eye		
θ	half of the frustum field of view		

Table I: ***Some symbols used in this section***

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 ray L through the light position $\mathbf{l}$ yields:

$$\mathbf{p} = \mathbf{p}_l - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}(\mathbf{p}_l - \mathbf{l})}(\mathbf{p}_l - \mathbf{l}). \tag{3}$$

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

$$\mathbf{p}_e = \mathbf{p} - \frac{\boldsymbol{\pi}_e\mathbf{p}}{\boldsymbol{\pi}_e(\mathbf{e}-\mathbf{p})}(\mathbf{e}-\mathbf{p}). \tag{4}$$

The eye image plane is parameterized with an equation similar to Equation 2 except that j is used instead of v. The j coordinate can be computed from $\mathbf{p}_e$ as:

$$j = \frac{\hat{\mathbf{y}}_e \cdot (\mathbf{p}_e - \mathbf{p}_{e0})}{W_e} = \frac{\hat{\mathbf{y}}_e^\top (\mathbf{p}_e - \mathbf{p}_{e0})}{W_e}. \tag{5}$$

$\hat{\mathbf{y}}_e$ is transposed to convert the dot product into multiplication by a row vector.

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

$$\frac{\mathrm{d}j}{\mathrm{d}t} = \frac{\partial j}{\partial \mathbf{p}_e}\,\frac{\partial \mathbf{p}_e}{\partial \mathbf{p}}\,\frac{\partial \mathbf{p}}{\partial \mathbf{p}_l}\,\frac{\partial \mathbf{p}_l}{\partial v}\,\frac{\mathrm{d}v}{\mathrm{d}t} \tag{6}$$

$$\frac{\partial j}{\partial \mathbf{p}_e} = \frac{\hat{\mathbf{y}}_e^\top}{W_e} \tag{7}$$

$$\frac{\partial \mathbf{p}_e}{\partial \mathbf{p}} = \mathbf{I} + \frac{\boldsymbol{\pi}_e\mathbf{p}}{\boldsymbol{\pi}_e(\mathbf{e}-\mathbf{p})}\mathbf{I} - \frac{\boldsymbol{\pi}_e\mathbf{e}}{(\boldsymbol{\pi}_e(\mathbf{e}-\mathbf{p}))^2}\left((\mathbf{e}-\mathbf{p})\boldsymbol{\pi}_e\right) \tag{8}$$

$$\frac{\partial \mathbf{p}}{\partial \mathbf{p}_l} = \mathbf{I} - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}\mathbf{I} + \frac{\boldsymbol{\pi}\mathbf{l}}{(\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l}))^2}\left((\mathbf{p}_l-\mathbf{l})\boldsymbol{\pi}\right) \tag{9}$$

$$\frac{\partial \mathbf{p}_l}{\partial v} = W_l\hat{\mathbf{y}}_l \tag{10}$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}G}{\mathrm{d}t}. \tag{11}$$

The notation $\frac{\partial \mathbf{p}}{\partial \mathbf{q}}$ is shorthand for the Jacobian matrix:

$$\frac{\partial \mathbf{p}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial p_0}{\partial q_0} & \cdots & \frac{\partial p_0}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_n}{\partial q_0} & \cdots & \frac{\partial p_n}{\partial q_n} \end{bmatrix}$$

More details on vector partial derivatives can be found in a vector calculus text [Hubbard and Hubbard 2001].

The expression for $\mathrm{d}j/\mathrm{d}t$ is quite complex. In 2D, it can be reduced to a simpler form through judicious substitutions. We first multiply together Equations 9–11:

$$\begin{aligned}
\frac{\partial \mathbf{p}}{\partial t} &= \frac{\partial \mathbf{p}}{\partial \mathbf{p}_l}\,\frac{\partial \mathbf{p}_l}{\partial v}\,\frac{\partial v}{\partial t} \\
&= W_l\frac{\mathrm{d}G}{\mathrm{d}t}\left(\hat{\mathbf{y}}_l - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}\hat{\mathbf{y}}_l + \frac{(\boldsymbol{\pi}\mathbf{l})(\boldsymbol{\pi}\hat{\mathbf{y}}_l)}{(\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l}))^2}(\mathbf{p}_l-\mathbf{l})\right) \\
&= W_l\frac{\mathrm{d}G}{\mathrm{d}t}\left(\frac{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l}) - \boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}\hat{\mathbf{y}}_l + \frac{(\boldsymbol{\pi}\mathbf{l})(\boldsymbol{\pi}\hat{\mathbf{y}}_l)}{(\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l}))^2}(\mathbf{p}_l-\mathbf{l})\right) \\
&= W_l\frac{\mathrm{d}G}{\mathrm{d}t}\frac{\boldsymbol{\pi}\mathbf{l}}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}\left(\frac{\boldsymbol{\pi}\hat{\mathbf{y}}_l}{\boldsymbol{\pi}(\mathbf{p}_l-\mathbf{l})}(\mathbf{p}_l-\mathbf{l}) - \hat{\mathbf{y}}_l\right).
\end{aligned} \tag{12}$$

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 $-d_l/n_l$. Superimposing the light and surface coordinate systems, we see that we can substitute $\boldsymbol{\pi}\hat{\mathbf{y}}_l =$

$-\sin\beta_l$, where β_l is the angle between $\hat{\mathbf{z}}_l$ and $\hat{\mathbf{n}}$. We substitute $||\mathbf{p}_l - \mathbf{l}||\hat{\mathbf{v}}_l$ into the remaining $(\mathbf{p}_l - \mathbf{l})$ terms, where $-\hat{\mathbf{v}}_l$ is the direction from the light. We then replace the resulting $\boldsymbol{\pi}\hat{\mathbf{v}}_l$ term with $-\cos\psi_l$:

$$\frac{\partial \mathbf{p}}{\partial t} = W_l \frac{\mathrm{d}G}{\mathrm{d}t}\frac{d_l}{n_l}\left(\hat{\mathbf{y}}_l - \frac{\sin\beta_l}{\cos\psi_l}\hat{\mathbf{v}}_l\right). \tag{13}$$

We then expand $\hat{\mathbf{y}}_l$ an $\hat{\mathbf{v}}_l$ in terms of the surface coordinate axes $(\hat{\mathbf{b}}, \hat{\mathbf{n}})$:

$$\hat{\mathbf{y}}_l = \cos\beta_l \hat{\mathbf{b}} - \sin\beta_l \hat{\mathbf{n}} \tag{14}$$

$$\hat{\mathbf{v}}_l = -\sin\psi_l \hat{\mathbf{b}} - \cos\psi_l \hat{\mathbf{n}}. \tag{15}$$

Substituting these equations into Equation 13 and utilizing the fact that $\psi_l - \beta_l = \phi_l$ yields:

$$\begin{aligned}\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} &= W_l \frac{\mathrm{d}G}{\mathrm{d}t}\frac{d_l}{n_l}\frac{(\cos\psi_l\cos\beta_l + \sin\psi_l\sin\beta_l)}{\cos\psi_l}\hat{\mathbf{b}} \\ &= W_l \frac{\mathrm{d}G}{\mathrm{d}t}\frac{d_l}{n_l}\frac{\cos(\psi_l - \beta_l)}{\cos\psi_l}\hat{\mathbf{b}} \\ &= W_l \frac{\mathrm{d}G}{\mathrm{d}t}\frac{d_l}{n_l}\frac{\cos\phi_l}{\cos\psi_l}\hat{\mathbf{b}}. \end{aligned} \tag{16}$$

Now we multiply together Equations 7, 8, and 16 and substitute $\boldsymbol{\pi}_e\mathbf{e} = n_e$, $\boldsymbol{\pi}_e(\mathbf{e} - \mathbf{p}) = d_e$, and $(\mathbf{e} - \mathbf{p}) = ||\mathbf{e} - \mathbf{p}||\hat{\mathbf{v}}_e$:

$$\frac{\mathrm{d}j}{\mathrm{d}t} = \frac{\mathrm{d}G}{\mathrm{d}t}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\psi_l}\left(\hat{\mathbf{y}}_e^\top\hat{\mathbf{b}} - \frac{(\hat{\mathbf{y}}_e^\top\hat{\mathbf{v}}_e)(\boldsymbol{\pi}_e\hat{\mathbf{b}})}{\boldsymbol{\pi}_e\hat{\mathbf{v}}_e}\right). \tag{17}$$

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 $\mathrm{d}j/\mathrm{d}t$:

$$\begin{aligned}\frac{\mathrm{d}j}{\mathrm{d}t} &= \frac{\mathrm{d}G}{\mathrm{d}t}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\psi_l}\frac{(\cos\phi_e\cos\beta_e + \sin\phi_e\sin\beta_e)}{\cos\phi_e} \\ &= \frac{\mathrm{d}G}{\mathrm{d}t}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\psi_l}\frac{\cos(\beta_e - \phi_e)}{\cos\phi_e} \\ &= \frac{\mathrm{d}G}{\mathrm{d}t}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\psi_l}\frac{\cos\psi_e}{\cos\phi_e}. \end{aligned} \tag{18}$$

Plugging $\mathrm{d}j/\mathrm{d}t$ into Equation 1 yields the final expression for aliasing error:

$$m = \frac{r_j}{r_t}\frac{\mathrm{d}G}{\mathrm{d}t}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\phi_e}\frac{\cos\psi_e}{\cos\psi_l}. \tag{19}$$

3.2.1 *Geometric derivation.* Some basic intuition for the terms in Equation 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 2). 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:

$$m = \frac{w_l'}{w_e'}. \tag{20}$$

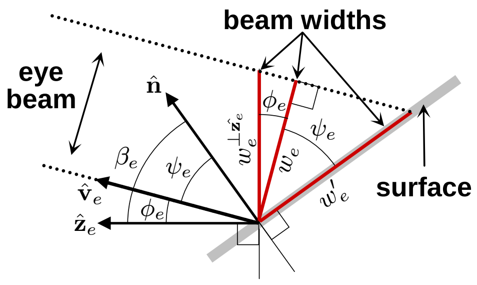


Fig. 3: ***Computing the projected eye beam width on a surface.*** *w_e is the actual width of the beam, w_e' is the width of the projection onto the surface, and $w^{\perp \hat{\mathbf{z}}_e}$ is the width of beam measured perpendicular to $\hat{\mathbf{z}}_e$.*

By the properties of 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:

$$w_e^{\perp \hat{\mathbf{z}}_e} = \frac{W_e}{r_j} \frac{d_e}{n_e} \tag{21}$$

For a narrow beam, we can assume that the sides of the beam are essentially parallel. From Figure 3 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':

$$w_e = w_e^{\perp \hat{\mathbf{z}}_e} \cos\phi_e \tag{22}$$

$$w_e' = \frac{w_e}{\cos\psi_e} = \frac{W_e}{r_j} \frac{d_e}{n_e} \frac{\cos\phi_e}{\cos\psi_e}. \tag{23}$$

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:

$$w_l' = \frac{W_l}{r_t} \frac{\mathrm{d}G}{\mathrm{d}t} \frac{d_l}{n_l} \frac{\cos\phi_l}{\cos\psi_l}. \tag{24}$$

Plugging Equations 23 and 24 into Equation 20 yields Equation 19.

3.2.2 *Directional lights.* As a point light at $\mathbf{l}$ moves away towards infinity it converges to a directional light $\hat{\mathbf{l}}$. Equation 3 converges to:

$$\mathbf{p} = \mathbf{p}_l - \frac{\boldsymbol{\pi}\mathbf{p}_l}{\boldsymbol{\pi}\hat{\mathbf{l}}} \hat{\mathbf{l}}. \tag{25}$$

Equation 9 becomes:

$$\frac{\partial \mathbf{p}}{\partial \mathbf{p}_l} = \mathbf{I} - \frac{1}{\boldsymbol{\pi}\hat{\mathbf{l}}} \hat{\mathbf{l}}\boldsymbol{\pi}. \tag{26}$$

In Equation 19, the n_l/d_l term converges to 1 and the $\cos\phi_l$ term becomes constant.

The formulation for aliasing error in Equation 19 is similar to those used for aliasing error in previous work [Stamminger and Drettakis 2002; Wimmer et al. 2004; Zhang et al. 2006]. 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

We will now consider the significance of the various factors of Equation 19. Because the locations of eye samples on the light image plane are fixed by the viewing parameters, scene geometry, and light position, the only way we can control aliasing error is by modifying the light sample locations. It is helpful to look at the aliasing error in terms of the relative spacing of the light and eye samples in v on the light's image plane, $(\Delta v)_l$ and $(\Delta v)_e$ (see Figure 2), because $(\Delta v)_l$ encapsulates the two factors that affect the light sample locations, the parameterization and the shadow map resolution, while $(\Delta v)_e$ encapsulates all the other terms.[2] The sample spacing can be split into two parts – the *spacing distribution functions* δ_l and δ_e, which control the variation in spacing, and the resolution, which controls the overall scale:

$$m = \frac{(\Delta v)_l}{(\Delta v)_e} = \frac{\frac{1}{r_t}\frac{\mathrm{d}v}{\mathrm{d}t}}{\frac{1}{r_j}\frac{\mathrm{d}v}{\mathrm{d}j}} = \frac{\frac{\delta_l}{r_t}}{\frac{\delta_e}{r_j}} = \frac{r_j}{r_t}\frac{\delta_l}{\delta_e}. \tag{27}$$

δ_l and δ_e which are simply the derivatives $\mathrm{d}v/\mathrm{d}t$ and $\mathrm{d}v/\mathrm{d}j$, respectively. They are related to Equation 19 as follows:

$$m = \frac{r_j}{r_t}\underbrace{\left(\frac{\mathrm{d}G}{\mathrm{d}t}\right)}_{\delta_l}\underbrace{\left(\overbrace{\left(\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\phi_e}\right)}^{1/\tilde{\delta}_e}\frac{\cos\psi_e}{\cos\psi_l}\right)}_{1/\delta_e}. \tag{28}$$

Following Stamminger and Drettakis [2002], δ_e can be further factored into two components – a *perspective factor*, $\tilde{\delta}_e$, and a *projection factor*, $\cos\psi_l/\cos\psi_e$. The perspective factor consists of terms related to the light and eye. It is independent of the scene and is bounded over all points inside the view frustum. The projection factor, on the other hand, depends on the orientation of the surfaces in the scene, and can potentially cause the aliasing error to become unbounded. 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, which we denote as $\tilde{m}$:

$$\tilde{m} = \frac{r_j}{r_t}\frac{\delta_l}{\tilde{\delta}_e} = \frac{w_e}{w_l}. \tag{29}$$

$\tilde{m}$ 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 aligned with the vector 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. Although any method that ignores the projection factor cannot fully remove aliasing error, removing perspective aliasing can significantly improve the overall image quality, while still maintaining good performance

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 using 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 2 becomes:

$$\mathbf{p}_l = \mathbf{p}_{l0} + uW_{lx}\hat{\mathbf{x}}_l + vW_{ly}\hat{\mathbf{y}}_l. \tag{30}$$

[2]For a point light, the orientation of the light image plane also affects the distribution of light samples in the scene. 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. For convenience, we choose a canonical light image plane for our computations that is independent of the actual near plane.

Equations 3 and 4 remain the same. We replace W_e in Equation 5 with the direction-specific W_{ey}. An additional equation is needed to compute i:

$$i = \frac{\hat{\mathbf{x}}_e^\top (\mathbf{p}_e - \mathbf{p}_{e0})}{W_{ex}}. \tag{31}$$

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 $\mathbf{M}_a$:

$$\mathbf{M}_a = \begin{bmatrix} r_i & 0 \\ 0 & r_j \end{bmatrix} \frac{\partial \mathbf{i}}{\partial \mathbf{s}} \begin{bmatrix} \frac{1}{r_s} & 0 \\ 0 & \frac{1}{r_t} \end{bmatrix} \tag{32}$$

$$\frac{\partial \mathbf{i}}{\partial \mathbf{s}} = \begin{bmatrix} \frac{\partial i}{\partial s} & \frac{\partial i}{\partial t} \\ \frac{\partial j}{\partial s} & \frac{\partial j}{\partial t} \end{bmatrix} = \frac{\partial \mathbf{i}}{\partial \mathbf{p}_e} \frac{\partial \mathbf{p}_e}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{p}_l} \frac{\partial \mathbf{p}_l}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{s}} \tag{33}$$

$$\frac{\partial \mathbf{i}}{\partial \mathbf{p}_e} = \begin{bmatrix} \frac{\hat{\mathbf{x}}_e^\top}{W_{ex}} \\ \frac{\hat{\mathbf{y}}_e^\top}{W_{ey}} \end{bmatrix} \tag{34}$$

$$\frac{\partial \mathbf{p}_l}{\partial \mathbf{u}} = \begin{bmatrix} W_{lx}\hat{\mathbf{x}}_l & W_{ly}\hat{\mathbf{y}}_l \end{bmatrix} \tag{35}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{s}} = \frac{\partial \mathbf{G}}{\partial \mathbf{s}}. \tag{36}$$

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:

$$\boldsymbol{\delta}_l = \frac{\partial \mathbf{G}}{\partial \mathbf{s}} = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right)^{-1} \tag{37}$$

$$\boldsymbol{\delta}_e = \left(\frac{\partial \mathbf{i}}{\partial \mathbf{u}}\right)^{-1}. \tag{38}$$

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 is 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 is to take $\tilde{m} = h(\mathbf{M}_a)$ where the normal of the planar surface used to compute $\mathbf{M}_a$ is aligned with the vector 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.

4. COMPUTING A PARAMETERIZATION WITH TIGHT BOUNDS ON PERSPECTIVE ALIASING ERROR

In the last section, we defined the aliasing metric m. In parts of the scene where $m > 1$, aliasing occurs. Where $m < 1$, the shadow map is oversampled. Ideally we would like to ensure that $m = 1$ everywhere in the scene, thus eliminating aliasing while minimizing the resolution of the shadow map. From Equation 27 we see that if we choose a shadow map parameterization that generates a spacing distribution function δ_l on the light image plane that is proportional to δ_e, then the error becomes uniform throughout the frustum. We can then scale the overall magnitude of the error to 1 by choosing the resolution $r_t = \rho_e r_j$, where ρ_e is the constant of proportionality of δ_l to δ_e. In a general scene, however, multiple surface points may be associated

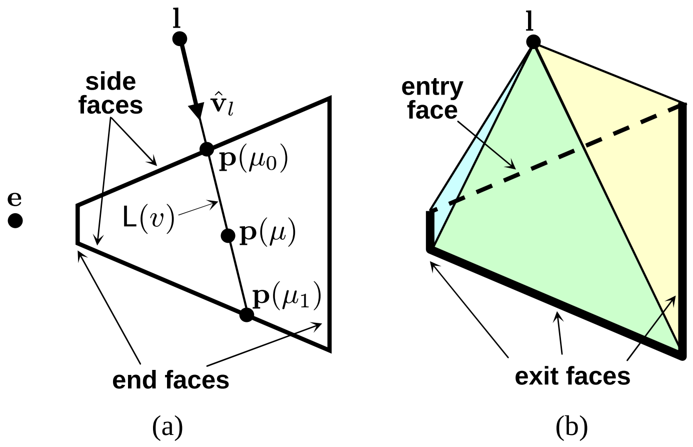


Fig. 4: ***Face partitioning.*** *(a) The minimum perspective factor $\tilde{\delta}_e$ along any light ray L through the view frustum can be tightly bounded by a function B whose minimum value occurs either at the faces where L enters or exits the frustum. Which set of faces depends on the position of the light and is the same for all light rays. We choose δ_b as the function B along the appropriate faces. (b) Because δ_b is different on each face, we use a separate the shadow map to cover the regions corresponding to each of the appropriate view frustum faces. For this light position, we use a shadow map for each exit face.*

with the same location on the light image plane. Therefore, δ_e may not be a single-valued function. If instead we choose $\delta_l(v) \sim \min(\delta_e(v))$ we can still guarantee that no aliasing is visible, but some portions of the scene may be oversampled (i.e. $m \leq 1$). Unfortunately, $\min(\delta_e(v))$ may be very complex and difficult to compute. For this reason, we seek a simpler, scene-independent approximation.

In this section compute a spacing distribution function δ_b that is a tight lower bound on the perspective factor of δ_e:

$$\delta_b(v) \in \left[\frac{1}{C}\min(\tilde{\delta}_e(v)), \min(\tilde{\delta}_e(v))\right], \tag{39}$$

where $C > 1$ is a small constant. Choosing $\delta_l \sim \delta_b$ and $r_t = \rho_b r_j$, where ρ_b is the constant of proportionality, ensures that $\tilde{m} \leq 1$, canceling out the perspective aliasing error in the scene. The δ_b function we derive changes at the boundaries of the view frustum faces, leading naturally to a partitioning of the shadow map into regions corresponding to the faces. We describe how to compute a shadow map parameterization F_b from δ_b in both 2D and 3D. The resulting parameterizations are still too complicated for practical use, but they do provide a good baseline for evaluating the simpler parameterizations in the next section.

4.1 Computing δ_b

The points in the view frustum along a ray $\mathsf{L}(v)$ passing through the location corresponding to v on the light image plane can be parameterized as shown in Figure 4a:

$$\mathbf{p}(\mu) = \mathbf{l} + \mu\hat{\mathbf{v}}_l(v). \tag{40}$$

We want to find $\min_\mu(\tilde{\delta}_e(\mu))$ on the interval $\mu \in [\mu_0, \mu_1]$ inside the view frustum. From Equation 28 we derive $\tilde{\delta}_e(\mu)$:

$$\tilde{\delta}_e(\mu) = c\frac{d_e(\mu)}{d_l(\mu)} \cos\phi_e(\mu) \tag{41}$$

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

$$d_e(\mu) = -\hat{\mathbf{z}}_e \cdot (\mathbf{p}(\mu) - \mathbf{e}) = -\mu(\hat{\mathbf{z}}_e \cdot \hat{\mathbf{v}}_l) - \hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e}) \tag{42}$$

$$d_l(\mu) = -\hat{\mathbf{z}}_l \cdot (\mathbf{p}(\mu) - \mathbf{l}) = -\mu(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l) \tag{43}$$

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

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, $\cos\theta$:

$$B(\mu) = c\frac{d_e(\mu)}{d_l(\mu)}\cos\theta \le \tilde{\delta}_e(\mu). \tag{44}$$

$\tilde{\delta}_e$ can be at most $1/\cos\theta$ times larger than B, i.e. when $\cos\phi_e = 1$. 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:

$$\frac{\mathrm{d}B}{\mathrm{d}\mu} = (c\cos\theta)\ \frac{\hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e})(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l)}{(\mu(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l))^2}. \tag{45}$$

The first term and the denominator of the second term are strictly positive, and the $(\hat{\mathbf{z}}_l \cdot \hat{\mathbf{v}}_l)$ 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 $\mu_{B_{\min}}$ of $B_{\min} = \min_\mu(B(\mu))$ must be at one of the boundaries of the interval:

$$\mu_{B_{\min}} = \underset{\mu\in[\mu_0,\mu_1]}{\operatorname{argmin}}\,(B(\mu)) = \begin{cases} \mu_0 & \hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e}) > 0, \\ \mu_1 & \hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e}) < 0. \end{cases} \tag{46}$$

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 vector $\hat{\mathbf{l}}$). Equation 46 shows that $B_{\min}$ 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. Because there is no dependence on the ray itself, the face selection criterion applies to all rays passing through the view frustum. $B \le \tilde{\delta}_e$ implies that if $B_{\min} = \tilde{\delta}_e$, as is the case when $\mu_{B_{\min}}$ is on a side face (because $\cos\phi_e = \cos\theta$ on side faces), then $B_{\min} = \tilde{\delta}_{e,\min}$. Thus the only time that $B_{\min} \ne \tilde{\delta}_{e,\min}$ is when $\mu_{B_{\min}}$ is on an end face, in which case $B_{\min}$ is still a tight lower bound. Therefore $B_{\min}$ is a suitable approximation for $\tilde{\delta}_{e,\min}$. We use this approximation for δ_b:

$$\begin{aligned}\delta_b &= B(\mu_{B_{\min}}) \\ &= \frac{W_e}{W_l}\frac{n_l}{n_e}\frac{d_e}{d_l}\frac{\cos\theta}{\cos\phi_l}.\end{aligned} \tag{47}$$

d_e and d_l are the values for points along the appropriate faces chosen according to Equation 46. As the light and eye move around, $\mu_{B_{\min}}$ transitions from entry to exit faces or vice versa. When the transitions occur, $B_{\min}$ is the same on both sets of faces. Thus the transitions do not cause temporal discontinuities in δ_b.

4.2 Computing a parameterization in 2D

Now that we have derived δ_b, we need to compute the parameterization F that generates $\delta_l \sim \delta_b$. First we need to rewrite δ_l in terms of F using Equation 28:

$$\delta_l = \frac{\mathrm{d}G}{\mathrm{d}t} = \left(\frac{\mathrm{d}F}{\mathrm{d}v}\right)^{-1}. \tag{48}$$

Now we can set $\delta_l = \rho_b \delta_b$ and solve for F, which we rename to F_b to emphasize that it was derived from δ_b:

$$\left(\frac{\mathrm{d}F_b}{\mathrm{d}v}\right)^{-1} = \rho_b \delta_b(v)$$

$$\frac{\mathrm{d}F}{\mathrm{d}v} = \frac{1}{\rho_b \delta_b(v)} \tag{49}$$

$$F_b(v) = \frac{1}{\rho_b} \int_0^v \frac{1}{\delta_b(\nu)} \mathrm{d}\nu. \tag{50}$$

We solve for the constant of proportionality ρ_b by normalizing F_b to the range $[0, 1]$ on $v \in [0, 1]$:

$$\rho_b = \int_0^1 \frac{1}{\delta_b(v)} \mathrm{d}v. \tag{51}$$

The same process can be used to find the parameterization corresponding to any arbitrary $\delta(v)$, so long as $\delta(v) > 0$ over the domain of integration.

4.3 Computing a parameterization in 3D

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

$$\frac{\partial^2 s}{\partial u \partial v} = \frac{\partial^2 s}{\partial v \partial u} \quad \text{and} \quad \frac{\partial^2 t}{\partial u \partial v} = \frac{\partial^2 t}{\partial v \partial u}. \tag{52}$$

Because $\partial \mathbf{F}_b / \partial \mathbf{u} = (\boldsymbol{\delta}_b)^{-1}$, if $\partial(\delta_b)^{-1}_{00} / \partial v \neq \partial(\delta_b)^{-1}_{01} / \partial u$ and $\partial(\delta_b)^{-1}_{10} / \partial v \neq \partial(\delta_b)^{-1}_{11} / \partial u$ does not hold, then $(\boldsymbol{\delta}_b)^{-1}$ is not the gradient of a function $\mathbf{F}_b$. Finally, even if $\boldsymbol{\delta}_b^{-1}$ 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 compute scalar functions $\delta_{b,s}$ and $\delta_{b,t}$ derived from Equation 47 for each face and integrate their multiplicative inverses w.r.t. u and v, respectively, to obtain s and t. We choose a coordinate system on each face as shown in Figure 5. For the W_l term, we use the width of the parameterized region in the appropriate direction and for $\cos \phi_l$ we use $\cos \phi_{lx}$ and $\cos \phi_{ly}$, the angles between $\hat{\mathbf{z}}_l$ and the projection of $\hat{\mathbf{v}}_l$ into the $x_l z_l$ and $y_l z_l$ planes, respectively. To obtain scalars 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 a conservative bound $W_e = \min(W_{ex}, W_{ey})$ 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.

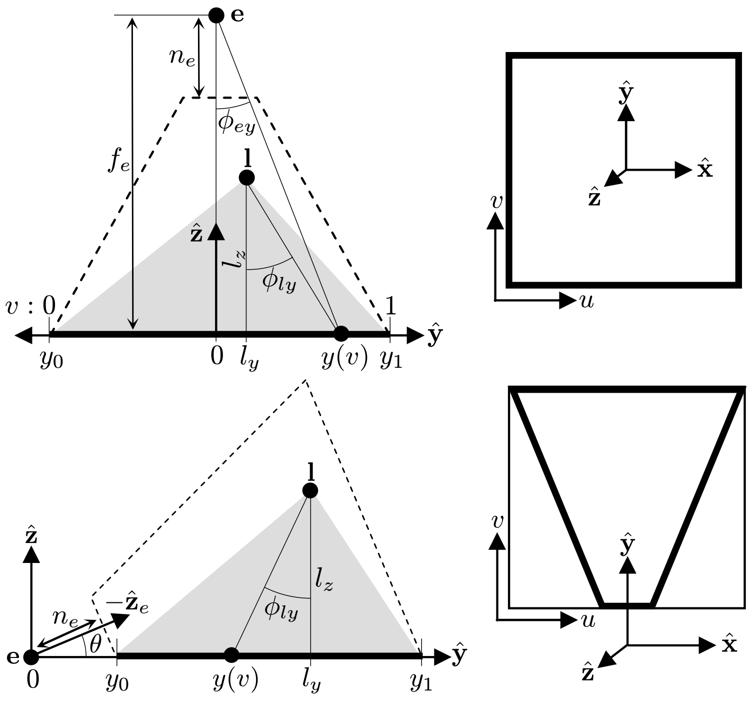


Fig. 5: ***Frustum face coordinate systems.*** *For each row, a side view appears on the left and a view from above the face appears on the right. (Top row) End face. The far face is shown here. The coordinate axes are aligned with the eye space coordinate axes with the origin at the center of the face. (Bottom row) Side face. The y-axis is aligned with the center line of the face, and the z-axis is aligned with the face normal. The origin is at the eye.*

4.3.1 *End faces.* We will first look at $\delta_{b,t}^{\text{end}}$. 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:

$$\cos\phi_{ly}(v) = \frac{l_z}{\sqrt{(y(v) - l_y)^2 + l_z^2}}. \tag{53}$$

Plugging these equations into Equation 47 we get:

$$\delta_{b,t}^{\text{end}}(v) = \frac{\cos\theta}{\cos\phi_{ly}(v)}. \tag{54}$$

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

4.3.2 *Side faces.* 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:

$$\frac{d_e}{n_e} = \frac{y}{y_0}. \tag{55}$$

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

$$x_0(y) = -\frac{W_e}{2}\frac{y}{y_0}, \quad x_1(y) = \frac{W_e}{2}\frac{y}{y_0}. \tag{56}$$

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:

$$\begin{aligned} x(u) &= (x_1(y_1) - x_0(y_1))u + x_0(y_1) \\ y(v) &= (y_1 - y_0)v + y_0 \end{aligned}$$

Putting all these equations together yields:

$$\delta^{\text{side}}_{b,s}(u,v) = \frac{y(v)}{y_1}\frac{\cos\theta}{\cos\phi_{lx}(u)}. \tag{57}$$

$$\delta^{\text{side}}_{b,t}(v) = \frac{W_e}{(y_1 - y_0)}\frac{y(v)}{y_0}\frac{\cos\theta}{\cos\phi_{ly}(v)}. \tag{58}$$

Note that because $\delta^{\text{side}}_{b,s}$ is a function of both u and v, $\mathbf{F}^{\text{side}}_b(\mathbf{u})$ is not, strictly speaking, just two 1D parameterizations, one for s and one for t. Therefore it must satisfy the more stringent conditions outlined at the beginning of this section. $\mathbf{F}^{\text{side}}_b$ must be integrable and one-to-one. It is trivial to show that if we obtain $F^{\text{side}}_{b,s}(u,v)$ by integrating $1/\delta^{\text{side}}_{b,s}$ w.r.t. u and $1/\delta^{\text{side}}_{b,t}$ w.r.t. v, then the mixed partials of both the s and t components of $\mathbf{F}^{\text{side}}_b(\mathbf{u})$ are equal. $\mathbf{F}^{\text{side}}_b$ is also invertible, and thus one-to-one. Because $1/\delta^{\text{side}}_{b,t}(v)$ is strictly positive in valid configurations, $F_{b,t}(v)$ is monotonic and invertible, so we can obtain v from t using $(F_{b,t})^{-1}$. $1/\delta^{\text{side}}_{b,s}(u,v)$ is also strictly positive, so we can plug $v(t)$ into $F_{b,s}(u,v)$ and invert to find u.

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

$$F^{\text{end}}_{b,t}(v) = \frac{1}{\rho^{\text{end}}_{b,t}}\zeta^{\text{end}}_{b,t}\Big|_0^v \qquad \rho^{\text{end}}_{b,t} = \zeta^{\text{end}}_{b,t}\Big|_0^1 \tag{59}$$

$$F^{\text{side}}_{b,s}(u,v) = \frac{1}{\rho^{\text{side}}_{b,s}(v)}\zeta^{\text{side}}_{b,s}\Big|_{u_0(v)}^u \qquad \rho^{\text{side}}_{b,s}(v) = \zeta^{\text{side}}_{b,s}\Big|_{u_0(v)}^{u_1(v)} \tag{60}$$

$$F^{\text{side}}_{b,t}(v) = \frac{1}{\rho^{\text{side}}_{b,t}}\zeta^{\text{side}}_{b,t}\Big|_0^v \qquad \rho^{\text{side}}_{b,t} = \zeta^{\text{side}}_{b,t}\Big|_0^1 \tag{61}$$

$$\zeta_{b,t}^{\text{end}}(v) = C_{b,t}^{\text{end}}\ l_z \sinh^{-1}\left(\frac{y(v)-l_y}{l_z}\right) + C \tag{62}$$

$$\zeta_{b,s}^{\text{side}}(u,v) = C_{b,s}^{\text{side}}\ \frac{l_z}{y(v)} \sinh^{-1}\left(\frac{x(u)-l_x}{l_z}\right) + C \tag{63}$$

$$\zeta_{b,t}^{\text{side}}(v) = -C_{b,t}^{\text{side}}\ \frac{l_z}{\sqrt{l_y^2+l_z^2}} \log\left(\frac{2}{y(v)}\left(\frac{l_y(l_y-y(v))+l_z^2}{l_z\sqrt{l_y^2+l_z^2}} + \frac{1}{\cos\phi_{ly}(v)}\right)\right) + C \tag{64}$$

$$\begin{aligned}
C_{b,t}^{\text{end}} &= \frac{1}{(y_1-y_0)\cos\theta} = \frac{n_e}{W_e d_e\cos\theta}\\
C_{b,s}^{\text{side}} &= \frac{1}{(x_1(y_1)-x_0(y_1))\cos\theta} = \frac{n_e}{W_e f_e\cos\theta}\\
C_{b,t}^{\text{side}} &= \frac{y_0}{W_e\cos\theta} = \frac{n_e}{W_e\cos^2\theta}
\end{aligned}$$

Note that for $F_{b,s}^{\text{side}}$ 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 $u_1(v)$ are found by solving $x(u) = x_0(y(v))$ and $x(u) = x_1(y(v))$ for u:

$$u_0(v) = \frac{y_1 - y(v)}{2y_1}, \quad u_1(v) = \frac{y_1 + y(v)}{2y_1}. \tag{65}$$

Unfortunately, the parameterizations derived from δ_b are too complex for practical use. In the next section, we will examine approximations to δ_b that have simpler parameterizations. (The Appendix includes further analysis of the parameterizations derived from δ_b).

5. DERIVING THE LOGPSM PARAMETERIZATION

We have now derived tight bounds δ_b on the perspective factor and the parameterizations based on δ_b that can cancel out the perspective aliasing in the scene. In this section, we seek spacing distribution functions that produce error that is nearly as low as that produced by $\delta_l \sim \delta_b$, but that have simpler parameterizations. We first discuss the global error metrics that we use to compare different parameterizations. We then analyze several parameterizations to identify those with error on the same order as the parameterizations for $\delta_{b,t}^{\text{end}}$, $\delta_{b,s}^{\text{side}}$, and $\delta_{b,t}^{\text{side}}$. We use these simpler parameterizations to formulate our LogPSM parameterization.

5.1 Global error measures

To compare the error of different parameterizations we would like a metric for the error over the entire view frustum. If we replace $\tilde{\delta}_e$ in Equation 29 with δ_b, we obtain a metric $\widetilde{M}$ that is a tight upper bound on the perspective error $\tilde{m}$ over all points that map to a point v on the light image plane:

$$\widetilde{M}(v) = \frac{r_j}{r_t}\frac{\delta_l(v)}{\delta_b(v)}. \tag{66}$$

For a given $\delta_l(v)$, the 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:

$$R_t = \max_v\left(\frac{\delta_l(v)}{\delta_b(v)}\right) \tag{67}$$

m	aliasing error
$\tilde{m}$	perspective aliasing error
$\widetilde{M}$	tight upper bound on maximum $\tilde{m}$ along a line L through the light
R	critical resolution factor. Related to maximum $\widetilde{M}$ over the view frustum
S	storage factor. Equal to $R_s \times R_t$
R	maximum R over all light positions
S	maximum S over all light positions

Table II: ***Summary of error metrics***

We call R_t the *critical resolution factor*. R_t is the smallest when $\delta_l \sim \delta_b$, in which case $R_t = \rho_b$. R_t can also be thought of as a measure of the maximum error that results when using a shadow map with the same resolution as the image. In 3D we combine the critical resolution factors for s and t:

$$S = R_s \times R_t. \tag{68}$$

We call S the *storage factor* because it represents the total number of texels in a critical resolution shadow map relative to the total number of pixels in the image. The use of the critical resolution and storage factor as error measures was originally introduced by Lloyd et al. [2006]. Note that $\delta_{b,s}^{\text{side}}$ is a function of u and v, so R_s^{side} is defined as:

$$R_s^{\text{side}} = \max_{(u,v)\in\mathcal{F}} \left(\frac{\delta_l(u,v)}{\delta_b^{\text{side}}(u,v)} \right), \tag{69}$$

where $\mathcal{F}$ is the set of points covered by the face. It is also useful to define measures of maximum perspective error over all possible light positions Ω:

$$\mathsf{R} = \max_{\Omega} R \tag{70}$$

$$\mathsf{S} = \max_{\Omega} S. \tag{71}$$

R and S can be evaluated simply by computing R and S for a light at infinity directly above the face. With the light in this position, the $\cos\phi_l$ factor of δ_b is 1 over the entire face and R and S reach their maximum values. Table II summarizes the error metrics we use.

We analyze various algorithms in terms of maximum error. The justification for this is that it is difficult to know *a priori* where shadows may appear on in the view, especially for interactive applications where the user has control over the view. The maximum error metric provides guaranteed bounds on perspective aliasing over the entire view frustum. But the maximum error may be overly conservative. Other error metrics may be more appropriate for different applications or for algorithms that use more information about scene geometry.

5.2 Maximum error of various parameterizations

Table III summarizes the error of four different parameterizations on end faces and in both directions on a side face (the values for the table are derived in the Appendix). The first is the error bound parameterization F_b derived in the last section. The next three are approximations for δ_b:

—**Uniform.** The simplest parameterization is the uniform parameterization:

$$\begin{bmatrix} s \\ t \end{bmatrix} = \mathbf{F}_{un}(u,v) = \begin{bmatrix} u \\ v \end{bmatrix}. \tag{72}$$

Parameterization	$\mathsf{R}_s^{\text{end}}$	$\mathsf{R}_s^{\text{side}}$	$\mathsf{R}_t^{\text{side}}$	S^{side}
Error bound (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 (F_{un})	$O(1)$	$O\left(\frac{f_e}{n_e}\right)$	$O\left(\frac{f_e}{n_e}\right)$	$O\left(\left(\frac{f_e}{n_e}\right)^2\right)$
Perspective (F_p)	–	$O(1)$	$O\left(\sqrt{\frac{f_e}{n_e}}\right)$	$O\left(\frac{f_e}{n_e}\right)$
Logarithmic (F_l)	–	–	$O\left(\log\left(\frac{f_e}{n_e}\right)\right)$	–
LogPSM (F_{lp})	–	$O(1)$	$O\left(\log\left(\frac{f_e}{n_e}\right)\right)$	$O\left(\log\left(\frac{f_e}{n_e}\right)\right)$

Table III: ***Maximum error***. *This table shows measures of the maximum error over all light directions* R *and* S *for our error bound parameterization* F_b *and several simpler ones. The perspective and logarithmic parameterizations have error that is on the same order as* $F_{b,s}$ *and* $F_{b,t}$ *for side faces. These parameterizations form the basis of the LogPSM parameterization, which has the same maximum error bounds as* F_b*. Entries marked with – indicate uses of the parameterizations that are not generally useful and thus have not been analyzed.*

This parameterization is used for standard shadow maps, although a standard shadow map is fit to the entire frustum instead of a face.

—**Perspective.** The next parameterization is a perspective projection along the y-axis of the side face:

$$\begin{bmatrix} s \\ t \end{bmatrix} = \mathbf{F}_p(u,v) = \begin{bmatrix} \dfrac{p_0 x(u) + p_1(y(v)+a)}{y(v)+a} \\ \dfrac{p_2(y(v)+a)+p_3}{y(v)+a} \end{bmatrix} \tag{73}$$

$$\begin{aligned} p_0 &= \frac{(y_1+a)}{W_e(y_1/y_0)} \\ p_1 &= \frac{1}{2} \\ p_2 &= \frac{y_1+a}{y_1-y_0} \\ p_3 &= -p_2(y_0+a), \end{aligned}$$

where a is a free parameter that translates the center of projection to a position of $y=-a$. $a=0$ yields the standard perspective projection that is used along z when rendering a perspective image from the eye. As $a \to \infty$, the parameterization degenerates to $\mathbf{F}_{un}$. The perspective parameterization is used by existing shadow map warping algorithms [Stamminger and Drettakis 2002; Wimmer et al. 2004; Martin and Tan 2004]. The primary difference between these algorithms is the way they choose a. The parameter used by LiSPSMs expressed in this coordinate system is $a=\sqrt{y_0 y_1}$. For the values in Table III, we have used the parameter that is optimal with respect to the error metric listed for each column.

—**Logarithmic.** The last parameterization is logarithmic:

$$t = F_{\log}(v) = \frac{\log\left(\frac{y(v)+a}{y_0+a}\right)}{\log\left(\frac{y_1+a}{y_0+a}\right)}. \tag{74}$$

A logarithmic parameterization has been suggested as a good fit for perspective aliasing [Wimmer et al. 2004; Zhang et al. 2006; Lloyd et al. 2006], but has not yet been used in an actual algorithm. It produces a spacing distribution that increases linearly in v and can thus match the $y(v)$ term in $\delta_{b,t}^{\text{side}}$. Equation 74 is a generalized version of a logarithmic parameterization with a free parameter a similar to that of $\mathbf{F}_p$.

The values in Table III are expressed in terms of the ratio of the far to near plane distances of the view frustum. It is this term which predominantly determines the error. There is also some dependence on the field of view, but the typical range of values for the field of view used in interactive applications is fairly limited. Further analysis of these parameterizations is found in the Appendix.

We are looking for approximate parameterizations that produce error that is at most on the same order as that of F_b. From Table III we see that suitable approximations to δ_b can be obtained by using a uniform parameterization for the end faces, a perspective projection for s on a side face, and a logarithmic parameterization for t. Our LogPSM parameterization combines a logarithmic transformation with a perspective projection to get good bounds for both s and t on a side face.

5.3 Logarithmic + perspective parameterization

To derive our logarithmic perspective parameterization, we start with $\mathbf{F}_p$ (Equation 73). $F_{p,s}$ is sufficient for the s direction. We need to transform $F_{p,t}$ into $F_{\log}$ (Equation 74). If we multiply $F_{p,t}$ by $-(y_1-y_0)/(y_1+a)$ and add 1, we get $(y_0+a)/(y(v)+a)$, which is the multiplicative inverse of the argument of the logarithm in the numerator of $F_{\log}$. We can then use the fact that negating the logarithm inverts its argument (i.e $\log(c/d) = -\log(d/c)$) to complete the transformation. The full parameterization is given by:

$$\begin{bmatrix} s \\ t \end{bmatrix} = \mathbf{F}_{lp}(u,v) = \begin{bmatrix} F_{p,s}(u,v) \\ c_0 \log(c_1 F_{p,t}(u,v) + 1) \end{bmatrix} \tag{75}$$

$$c_0 = \frac{-1}{\log\left(\frac{y_1+a}{y_0+a}\right)}$$

$$c_1 = -\frac{y_1 - y_0}{y_1 + a}.$$

With $a = 0$, this parameterization provides the same good error bounds in the s and t directions as the perspective and logarithmic parameterizations, respectively. See the Appendix for more details.

6. LOGPSM ALGORITHMS

In this section we outline several algorithms that use the LogPSM parameterization. These algorithms are basically extensions of LiSPSMs [Wimmer et al. 2004], cascaded shadow maps [Zhang et al. 2006; Engel 2007], and perspective-warped cube maps [Kozlov 2004]. The first algorithm uses a single shadow map to cover the entire view frustum. The second algorithm partitions the frustum along its z-axis into smaller sub-frusta and applies a single shadow map to each one. The third algorithm uses separate shadow maps for partitions corresponding to the view frustum faces. The first two algorithms are best suited for directional lights or spot lights, while the third algorithm can support both directional and omnidirectional point lights. Each algorithm consists of several steps:

(1) Compute the parameterizations for the shadow maps corresponding to each partition. The partitions are polyhedra defined by the entire view frustum, z-partitioned sub-frusta, or the intersection of the view frustum with convex hull of the light and the appropriate frustum faces.
(2) Error-based allocation of texture resolution.
(3) Rendering the shadow maps.
(4) Rendering the image with multiple shadow maps.

We will discuss each of these steps in detail. We will then describe some modifications to handle shearing artifacts that can sometimes arise with the face partitioning algorithm. Note that many of the details in this section also apply to other shadow map algorithms.

6.1 Parameterizing partitions

The perspective portion of the LogPSM parameterization determines the shape of the light frustum used to render the shadow map. To maximize the use of the available depth and shadow map resolution, the light frustum should be fit as tightly as possible to the relevant portions of its corresponding partition, P. If depth clamping is available, the near and far planes can be set to bound only the surfaces inside P that are visible to the eye [Brabec et al. 2002]. This will cause the depth of occluders between the light and the near plane of its frustum to be clamped to 0. However, this is sufficient to produce correct shadowing because depth information is only needed to prevent false self-shadowing on visible occluders. (Actually, the near plane should be backed off slightly towards the light so as to avoid self-shadowing artifacts near the edge of the view frustum). If depth clamping is not available, the light frustum must be expanded to include all potential occluders. This can be done by by taking the convex hull of P and the light position $\mathbf{l}$. 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 correctly. 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, could eliminate the unshadowed region.

6.1.1 *Entire view frustum.* We build on the LiSPSM algorithm [Wimmer et al. 2004] to apply the LogPSM parameterization to a single shadow map covering the entire view frustum. As the angle γ between the light and view directions approaches $0°$ or $180°$, LiSPSMs degenerate to a uniform parameterization in order to avoid excessive error. The uniform parameterization is produced when the near plane distance of the perspective parameterization n' diverges to ∞. To accomplish this, the LiSPSM algorithm modulates the optimal n' parameter for $\gamma = 90°$ with $1/\sin\gamma$. Given that for an overhead directional light $n' = n_e + a/\cos\theta$ and that a for LiSPSMs is $\sqrt{y_0 y_1}$, we have:

$$n' = \frac{n_e + \sqrt{n_e f_e}}{\sin\gamma}.$$

A LogPSM could use the same strategy, using the optimal parameter $a = 0$:

$$n' = \frac{n_e}{\sin\gamma}.$$

Unfortunately, for some values of γ this can result in error that is worse than the error of standard shadow maps. Essentially, the problem is that $\sin\gamma$ does not go to 0 fast enough for $\gamma \in [0°, \theta]$ and $\gamma \in [180°, 180° - \theta]$. LiSPSMs have the same problem, but the problem is worse for LogPSMs because n' has farther to go to get to ∞. Instead, we use a different falloff function. We first parameterize n' by η using the equation from Lloyd [2007] (see Equation 117 in the Appendix). Then we use the following function to compute η for

LogPSMs:

$$\eta = \begin{cases} 0 & \gamma < \gamma_a \\ -1 + (\eta_b + 1)\frac{\gamma - \gamma_a}{\gamma_b - \gamma_a} & \gamma_a < \gamma < \gamma_b \\ \eta_b + (\eta_c - \eta_b)\sin\left(\frac{\gamma-\gamma_b}{\gamma_c-\gamma_b} 90^\circ\right) & \gamma_b < \gamma < \gamma_c \\ \eta_c & \gamma < \gamma_c \end{cases} \tag{76}$$

$$\gamma_a = \frac{\theta}{2}, \quad \gamma_b = \theta, \quad \gamma_c = \theta + 0.3(90^\circ - \theta)$$

$$\eta_b = -0.2, \quad \eta_c = 1.$$

$\eta \in [-1, 1]$ corresponds to $n' = \infty$ at $\eta = -1$, $n' = n_e + \sqrt{n_e f_e}$ at $\eta = 0$, and $n' = n_e$ at $\eta = 1$. The shaping function for η in Equation 76 rises smoothly from -1 to η_b over the interval $[\gamma_a, \gamma_b]$ and then continues to rise until it smoothly transitions to η_c at $\gamma = \gamma_c$. Like the LiSPSM algorithm, this function provides a continuous transition to ∞, but with more control than the simple $\sin\gamma$ function. Note that this shaping function can also be used to improve the LiSPSM algorithm by using $\gamma_a = \theta/3$ and $\eta_c = 0$. A more complete analysis of this function and our choice of parameters is found in the Appendix.

The perspective portion of the LogPSM parameterization is encoded in a matrix $\mathbf{M}_s$ that is used during shadow map rendering. This matrix is computed the same as it is with LiSPSMs, but using our modified n'. Note that the light space z-axis used in the LiSPSM algorithm corresponds to our y-axis.

6.1.2 *z-partitioning.* The basic idea of z-partitioning schemes is to subdivide the view frustum along its z-axis so that tighter fitting shadow maps may be applied to each sub-frustum. The main difference between the various z-partitioning algorithms is where subdivisions are made. The maximum error over all partitions is minimized when the split points are chosen such that the far to near plane distance ratio of each partition is the same:

$$n_i = n_e \left(\frac{f_e}{n_e}\right)^{(i-1)/k} \tag{77}$$

$$f_i = n_{i+1} = n_e \left(\frac{f_e}{n_e}\right)^{i/k} \tag{78}$$

$$\frac{f_i}{n_i} = \left(\frac{f_e}{n_e}\right)^{1/k}. \tag{79}$$

where k is the number of partitions and $i \in \{1, 2, ..., k\}$. We use this partitioning scheme in this paper. The scheme can be viewed as a discrete approximation of $F_{\log}$ (Equation 74). This can be seen by splitting the t domain of the shadow map into k uniform partitions and mapping these through $G_{\log}$, the inverse of $F_{\log}$, with $a = 0$:

$$y = G_{\log}(t) = y_0 \left(\frac{y_1}{y_0}\right)^t. \tag{80}$$

Substituting n_e for y_0 and f_e for y_1 yields the split points of this partitioning scheme. Zhang et al. [2006] use a combination of this scheme with a uniform partitioning. When the same resolution is used for each partition, this algorithm favors regions further from the viewer (see Lloyd [2007] for more details). Once the z-partitions have been computed, applying the LogPSM parameterization to each partition proceeds just as in the single shadow map case.

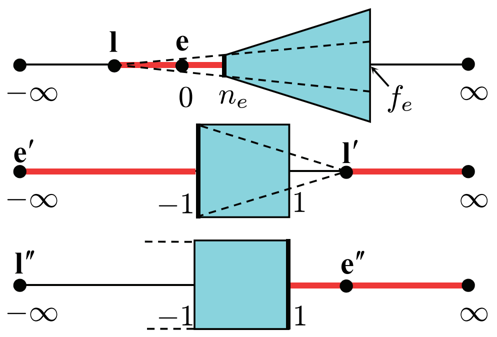


Fig. 6: ***Post-perspective parameterization of frustum faces.*** (Top) *With the light* $\mathbf{l}$ *behind the eye* $\mathbf{e}$ *we parameterize the entry face. The red points along the line contain potential occluders and must be included in the shadow map.* (Middle) *In post-perspective space the frustum becomes a unit cube, the eye goes to* $-\infty$ *and the light wraps around* ∞*. The entry face is now an exit face.* (Bottom) *After applying the light's perspective transform,* $\mathbf{l}''$ *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.*

6.1.3 *Face partitioning.* Face partitioning is motivated by the analysis in Section 5, which showed that δ_b changes at the boundaries of the frustum faces. We use a uniform parameterization for the end faces of the view frustum and the LogPSM parameterization for the side faces. The uniform and the perspective portions of the parameterizations can be handled using the PSM cube map algorithm of Kozlov [2004]. 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 separate projection matrix for the light $\mathbf{M}_l$ is fit to each exit face of the unit cube with each light frustum's near plane oriented parallel to the face. $\mathbf{M}_l$ 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 $\mathbf{M}_l$ corresponding to the z-axis should be scaled by -1 (see Figure 6). The inversion occurs whenever the face selection criterion in Equation 46 selects the entry faces (i.e. $\hat{\mathbf{z}}_e \cdot (\mathbf{l} - \mathbf{e}) > 0$). Thus parameterizing exit faces in post-perspective space always selects the same faces as Equation 46. The face partition polyhedra can be computed by taking the convex hulls of the light and exit faces of the unit cube in post-perspective space, intersecting them with the unit cube, and transforming them back to world space. The final matrix used to render each shadow map is obtained by concatenating the camera and light projection matrices, $\mathbf{M}_s = \mathbf{M}_l\mathbf{M}_c$.

6.2 Error-based allocation of texture resolution

We compute the shadows in the image in a single pass, storing all the shadow maps together in memory. We use a fixed texture resolution budget that must be distributed somehow among the shadow maps. The simplest approach is to allocate a fixed amount of resolution in each direction for each shadow map. One problem with this strategy is that as the light position varies relative to the eye, the maximum error can vary dramatically between partitions and shadow map directions (see Figure 9a). To keep the overall error as low as possible, we distribute the error evenly between the different partitions and between the s and t directions by adjusting the resolution as described by Lloyd et al. [2006]. The idea is to first compute R_s, R_t, and S for each partition. Each partition P_i then receives a fraction $\sigma_i = S_i/\Sigma_{j=1}^{k} S_j$ of the total texel budget T. The texel allocation for a partition is then divided between the s and t directions in the shadow

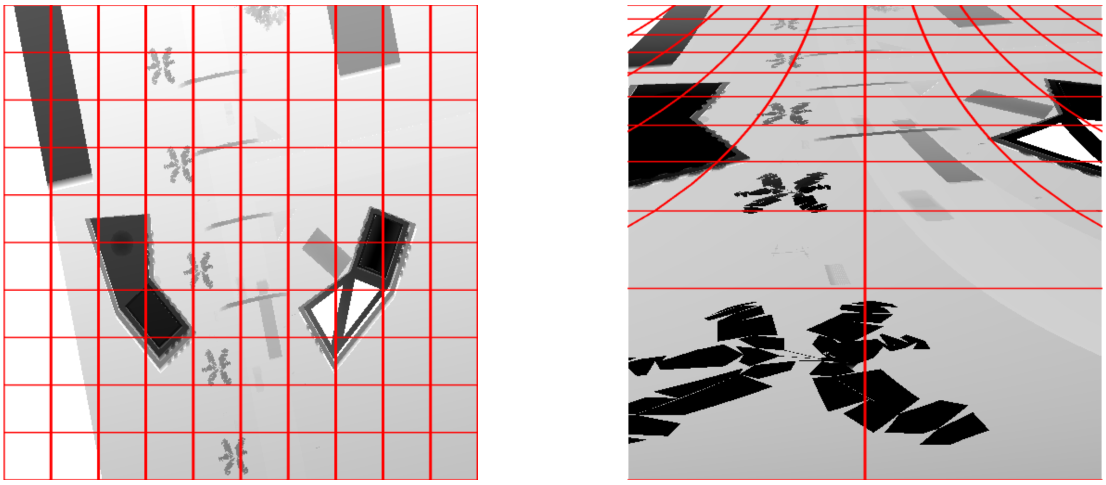


Fig. 7: *Standard and logarithmic perspective shadow maps.* *These shadow maps were rendered from overhead in the town scene. The viewpoint is at the bottom of the shadow map looking along the road (up in the shadow map). A uniform grid is superimposed on the shadow maps to highlight the warping from the LogPSM parameterization that causes areas near the viewer to occupy more of the shadow map. Note that straight lines become curved*

map in proportion to R_s and R_t:

$$r_s = \sqrt{\frac{(1-\alpha)}{\alpha}\frac{R_s}{R_t}\sigma T}, \qquad r_t = \sqrt{\frac{\alpha}{(1-\alpha)}\frac{R_t}{R_s}\sigma T}. \tag{81}$$

The parameter $\alpha \in [0, 1]$ gives the user control over how much the resolution allocation is biased towards the t direction. At $\alpha = 0.5$ there is no bias. The effect of unbiased resolution redistribution is shown in Figure 9b.

We compute R_s and R_t with Equation 67, treating each direction independently, as we did in Section 4. For R_s we use $W_l = W_{lx}$ and $\cos\phi_l = \cos\phi_{lx}$, and for R_t we use $W_l = W_{ly}$ and $\cos\phi_l = \cos\phi_{ly}$. With 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. $(\mathrm{d}F_{lp,s}/\mathrm{d}u)^{-1}$ and $(\mathrm{d}F_{lp,t}/\mathrm{d}v)^{-1}$ are monotonic, therefore 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. As an approximation, we evaluate the error at the point on each face that is closest to the light, in addition to the face vertices. For applications where a point light enters and exits the view frustum, it can sometimes be better to not use resolution redistribution at all because the LogPSM parameterization does not do a very good job at approximating δ_b when the light is very close to a face. The Appendix discuss a variation on the LogPSM parameterization that may address this problem. Resolution redistribution is highly recommended for directional lights, however.

Current GPU 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 allocated the shadow maps in a single fixed-sized texture atlas large enough to accommodate the changing shadow map dimensions.

6.3 Rendering the shadow maps

Each shadow map is rendered in a separate pass. In order to maximize rendering performance for each pass, objects should only be rendered into the shadow maps in which they actually lie. We call this *partition*

```
void logRasterize(
  varying float2 stPos,
  varying float3 edgeEq0,
  varying float3 edgeEq1,
  varying float3 edgeEq2,
  varying float3 depthEq,
  uniform float  d[3],
  out float depth : DEPTH )
{
  // invert logarithmic transform
  float3 p( stPos, 1 );
  p.y = d[0]*exp( d[1]*p.y )+d[2];

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

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

Fig. 8: ***Fragment shader for simulating logarithmic rasterization.***

culling. Partition culling can be accomplished by testing an object against the planes through the light and the partition silhouette edges using straightforward extensions of the existing view frustum culling techniques typically employed in graphics applications. For simple scenes, partition culling is of only small benefit and may not be worth the extra overhead, but for geometrically complex scenes this can be an important optimization. GPUs that support DirectX 10 features can render to multiple render targets in a single pass, but if the vertex data already resides on the GPU, this does little to reduce the overhead of multipass rendering. So partition culling can still be an important optimization here as well.

6.3.1 *Logarithmic rasterization.* The LogPSM parameterization requires logarithmic rasterization. Logarithmic rasterization causes planar primitives to become curved (see Figure 7). Current GPUs only support projective transformations. The logarithmic transformation could be approximated by computing it only for vertices using a vertex program, but when the f_e/n_e ratio is high, this would require a fine tessellation of the scene to avoid error. The tessellation could be performed adaptively, but this adds complexity to the algorithm. We currently simulate logarithmic rasterization by performing brute-force rasterization in a fragment program. We first render the scene using $\mathbf{M}_s$ with the viewport set to the range $[0, 0] \times [1, 1]$ and read back the 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 t coordinates using $\mathbf{F}_{lp}$ and render the axis-aligned bounding quad of the resulting triangle vertices. Using the fragment program in Figure 8, we invert $F_{lp,t}$ in order to undo the logarithmic warping, compare against the linear triangle edge equations, and discard fragments that fall outside the triangle. The equation for the inverse of $F_{lp,t}(v)$ is given by:

$$v = d_0 e^{d_1 t} + d_2 \tag{82}$$

$$d_0 = \frac{1}{c_1(y_1 - y_0)}, \qquad d_1 = \frac{1}{c_0}, \qquad d_2 = -\frac{c_1 y_0 + 1}{c_1(y_1 - y_0)}.$$

With our unoptimized simulator we observe frame rates of 2–3 fps on a PC with a GeForce 7800GT graphics card and a 2 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 implemented this yet. Even with a geometry shader, however, our simulation of logarithmic rasterization would be considerably slower than linear shadow map rasterization. GPUs have a number of optimizations, such as z-culling and higher rasterization rates for depth-only rendering, which are disabled when a fragment program outputs depth information. To test the difference in speed between standard rasterization and our brute-force method, we rendered a series of screen-sized quads at different depths. With decreasing depths, z-culling provides no benefit, and our brute force rasterization is only 6 times slower than standard linear rasterization. With increasing depths, however, z-culling discards fragments for all but the first quad, and our brute force rasterization is 24 times slower.

Hardware support for logarithmic rasterization can be implemented through incremental enhancements to current hardware [Lloyd et al. 2007]. The LogPSM parameterization is especially practical for hardware implementation because the standard graphics pipeline can be used for the perspective portion of the parameterization. Only the rasterizer needs to be modified to handle the logarithmic part. For the same error as other algorithms, LogPSMs can require less memory bandwidth and storage. This is important because shadow map rendering is often bandwidth limited, and high resolution shadow maps increase contention for limited GPU memory resources.

Hardware support would allow LogPSMs to be rendered at speeds comparable to other algorithms that are currently used for real-time shadow generation. For example, the frame rates for the three algorithms shown in Figure 12 that use standard rasterization are 87, 72, and 74 frames per second (from left to right).

The proposed Larrabee architecture [Seiler et al. 2008] performs rasterization completely in software. It should be relatively easy to modify an existing Larrabee rasterizer to support logarithmic rasterization. In particular, the table-based technique described by Lloyd et al. should work well with the binning approach used on Larrabee. This technique avoids the expensive computation of exponentials for each pixel by using a small precomputed table.

6.4 Rendering the image

We render the image using all the shadow maps in a single pass. The LogPSM algorithm used on the entire frustum is the simplest because it uses only one shadow map. We compute texture coordinates for the perspective part of the parameterization at the vertices using the transformation $\mathbf{M}_n\mathbf{M}_s\mathbf{p}$, where $\mathbf{M}_n$ maps the $[-1,-1] \times [1,1]$ range of $\mathbf{M}_s$ to $[0,0] \times [1,1]$ and $\mathbf{p}$ is the world position. We linearly interpolate these texture coordinates over the triangle. In the fragment program we perform the perspective divide on the texture coordinates and apply the logarithmic transformation in Equation 75 to the y coordinate to get the final texture coordinates for the shadow map lookup.

z-partitioning introduces additional shadow maps. In the fragment program we must determine in which partition the fragment lies. This can be done efficiently with a conditional and a dot product [Engel 2007]:

```
float4 zGreater = (n1_4 < eyePos.z);
float  partitionIndex = dot(zGreater, 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. If a sufficient number of interpolators are available, we can interpolate a set of texture coordinates for each shadow map and use the partition index to select the appropriate set. Otherwise we use the index to select the appropriate element from an array containing the matrices $\mathbf{M}_s$ for each partition and perform the matrix multiplication on the world position in the fragment program. Once we have the right set of texture coordinates, the computation proceeds just as in the single shadow map case.

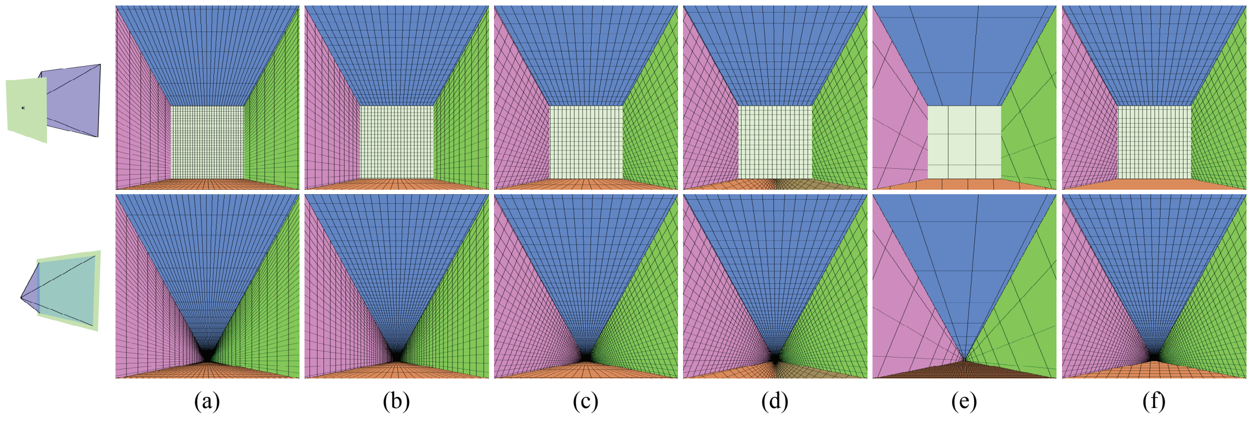


Fig. 9: ***The effects of resolution redistribution and shear handling.*** *The top row shows the texel grid of the shadow map projected onto a plane oriented perpendicular to the view direction that is near the viewer. The bottom row uses a plane at the far end of the view frustum. (a) The same resolution 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. Redistributing the resolution according to error leaves little resolution for the other partitions. (f) Using a limited field of view for the parameterization but allocation resolution as if the original field of view were used leads to high error close to viewer but acceptable results further away.*

For face partitioning we compute the partition index using a small cube map. In each face of the cube map we store the index of the partition corresponding to that face. We perform the cube map lookup 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 the cube map indicating whether the face is a side face or an end face so we know whether or not to apply the logarithmic transformation.

6.5 Handling shear

The PSM cube map algorithm that we extend for our face partitioning algorithm uses a parameterization that is affixed to the face. For some light positions, the cross-section of the light beams become overly sheared, which can lead to disturbing artifacts (see Figure 10). With any shadow map algorithm some shearing can occur on surfaces that are nearly parallel to the light. However, this shearing is usually less noticeable because diffuse shading reduces the intensity difference between shadowed and unshadowed parts of these surfaces. Warping algorithms fit a rectangular shadow map to the trapezoidal view frustum, which can introduce some shearing on surfaces that are directly facing the light. This shearing is even more pronounced with face partitioning because the trapezoidal faces, as seen from the light, can become extremely sheared and flattened for some light positions. In addition, the shearing in one partition may not be consistent with that of adjacent partitions, and is therefore more noticeable. Because our error metrics do not take this shearing into account, redistributing the resolution according to error may actually make this shearing more noticeable. One possibility for incorporating shear into the metrics is to simply weight the error computed at the face vertices (shear is usually worst at the vertices) by a measure of the shear. But this will only change the distribution of resolution between faces. It does not change the parameterization itself.

One way to mitigate shearing is to unbind the parameterization from the face. Consider the view of a side

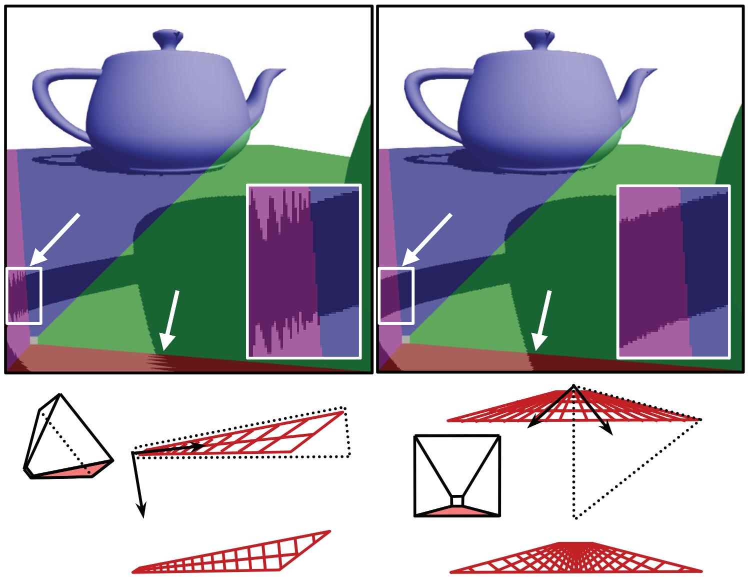


Fig. 10: ***Handling shear artifacts.*** (Top-left) *Shear artifacts resulting from face partitioning indicated by arrows. We use a low-resolution shadow map to make them easier to see. Regions are colored according to the face partition to which they belong.* (Top-right) *Shear artifacts removed by coordinate frame adjustment.* (Bottom) *View frustum as seen from the light in cases in which shearing occurs. To handle the first case, we realign the projection axis with the bisector of the side edges of the face and 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 then apply the coordinate frame adjustment to each of the sub-faces. The corrected faces are shown below the images.*

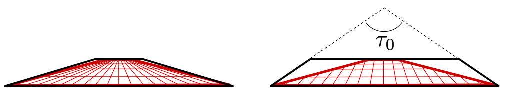


Fig. 11: τ***-Limiting.*** *(Left) a face partition with excessive shearing caused by a large angle* τ *between the edges of the trapezoidal region to which to parameterization is applied (black). (Right) Limiting* τ *to some constant* τ_0 *reduces the shearing at the expensive of error at the narrow end of the face partition.*

face from a directional light. The shearing of the light beam cross-sections can be minimized by fitting the parameterization to the symmetric trapezoid that bounds the face in the light's view. We align the midline of the trapezoid with the bisector of the two side edges of the face (see Figure 10). The resulting projection is no longer a tight fit, but greatly reduces the shear artifacts. However, when the angle between the two edges is high, this coordinate frame adjustment may provide little improvement (see the bottom face in Figure 9c). Therefore we 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 $\tau_0 > 90°$ ensures that no face will be split more than once and that no more than two faces will be split at the same time.

The image rendering pass needs to be modified slightly to handle split faces. In the cube map we store the indices of the shadow maps for both halves of the face. The first and second index are identical for unsplit faces. We compute the equations of the splitting planes containing the light and split lines and pass these to the fragment program. We determine which face a fragment belongs to as before, but then test the fragment's world position against the face's splitting plane to determine which index to use. We adopt the convention that the first index corresponds to the negative side of the plane. For faces without a split, it does not matter which side is selected. Once the appropriate index has been computed, the calculations proceed in the same way as before.

Another possibility for handling shear is to simply place a limit τ_0 on the angle between the sides of the trapezoid used to parameterize the face (see Figure 11). This approach does not require additional shadow maps. However, it leads 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 9e). We could also use the resolution that would have been allocated to the face had we not changed the parameterization. For surfaces near the view the error may be extremely high at the narrow end of the face partition, but acceptable for surfaces farther away (Figure 9f). 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.

For both of these approaches we find it easier to work in light space rather than the post-perspective space of the camera. We can compute the partitions in post-perspective space and transform them back into light space. Then we parameterize the partitions on a light image plane that is perpendicular to the light direction. For a point light there is no single light direction. Due to this and other complications we currently only handle shear for directional lights.

7. RESULTS AND ANALYSIS

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

—P: Perspective warping. Unless indicated otherwise, we use the LiSPSM warping parameter with our new shaping function for the falloff.

—Po: Perspective warping with original $1/\sin\gamma$ falloff.

—LogP: Logarithmic + perspective warping.

—ZPk: z-partitioning using k partitions.

—FP: Face partitioning.

—FPc: Face partitioning with coordinate frame adjustment.

—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. When ZPk appears without P or LogP, a uniform parameterization is used.

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 error, surfaces must pass through these regions. Moreover, shadow edges must also be present in these regions. We project grid lines from the shadow map onto the scene to more easily visualize the projection of shadow map texels themselves. In addition, we generate color coded images using an aliasing error metric. Figure 12 shows several of the possible metrics. (The color mapping used for the comparison images is shown in Figure 13). One possibility

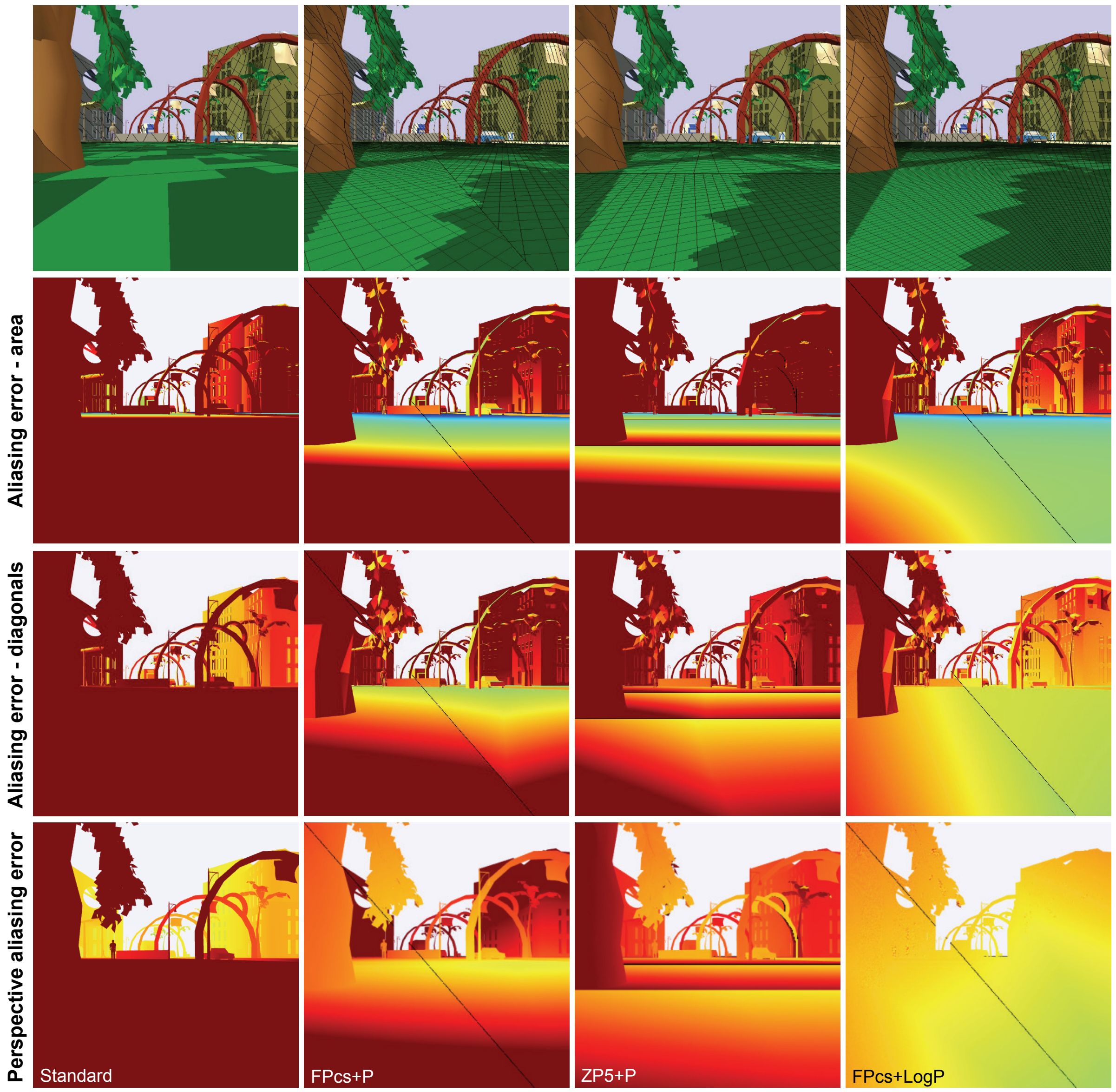


Fig. 12: ***Comparison of various algorithms using different error metrics.*** *The viewer is positioned below a tree in a town scene. In the top row, grid lines for every 5 texels are projected onto the scene. In subsequent rows, aliasing error is shown in terms of projected shadow map texel area in pixels, maximum extent of texel diagonals, and perspective aliasing error. 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. Both 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 resolutions are* 512×512*. For this view, the storage factor (a measure of maximum over the whole view frustum) is* 1.6×10^6, 1.8×10^3*, 69, and 13 for the* Standard*,* FPcs+P*,* ZP5+P*, and* FPcs+LogP *algorithms, respectively. The LogPSM produces the lowest overall error and has the most uniform distribution.* $(f/n = 1000, \theta = 30°)$.

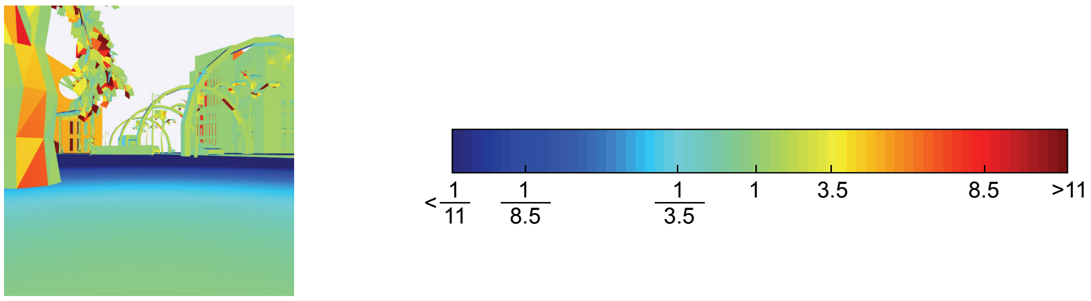


Fig. 13: ***Projection error and color mapping for error comparison images.*** *The image on the left shows a measure of the projection term of the aliasing error for the images in Figure 12. The LogPSM almost completely cancels out the perspective error.*

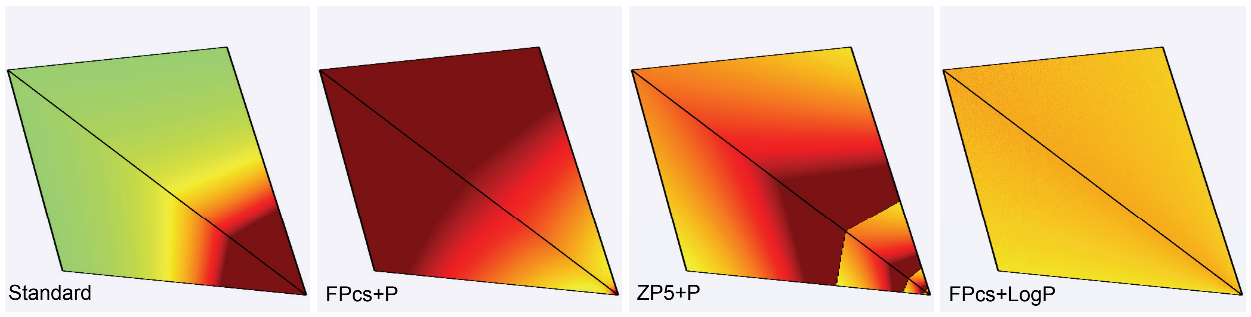


Fig. 14: ***Perspective error along faces of view frustum.*** *This is a third-person view of the view frustum used in Figure 12. The range of the color mapping has been scaled for this figure (green* $= 1$*, red* $= 32$*)*

is to use the area of the projected texels in the image:

$$m_a = \det\left(\begin{bmatrix} r_i & 0 \\ 0 & r_j \end{bmatrix} \frac{\partial \mathbf{i}}{\partial \mathbf{s}} \begin{bmatrix} \frac{1}{r_s} & 0 \\ 0 & \frac{1}{r_t} \end{bmatrix}\right) = \left|\mathbf{ds}\;\; \mathbf{dt}\right| \tag{83}$$

$$\mathbf{ds} = \left(\frac{r_i}{r_s}\frac{\partial i}{\partial s}, \frac{r_j}{r_s}\frac{\partial j}{\partial s}\right)^\top, \quad \mathbf{dt} = \left(\frac{r_i}{r_t}\frac{\partial i}{\partial t}, \frac{r_j}{r_t}\frac{\partial j}{\partial t}\right)^\top. \tag{84}$$

ds and **dt** are the change in screen space coordinates for a texel sized step in s and t in the shadow map. The availability of screen space derivatives in fragment programs make this metric easy to evaluate. The metric takes into account the error in both directions, but has the drawback that it does not capture shearing of the projected texels. Another possibility is to measure the maximum extent of a texel in the image using maximum of the diagonals:

$$m_d = \max\left(||\mathbf{ds} + \mathbf{dt}||, ||\mathbf{ds} - \mathbf{dt}||\right). \tag{85}$$

This metric also takes both directions into consideration, and to some extent it captures shearing. Unless stated otherwise, we use this metric for the rest of the comparison images in this section. It is also possible to visualize the components of aliasing error. Figure 12 shows $\max(\tilde{m}_s, \tilde{m}_t)$, the maximum of the perspective aliasing error in both directions. This is the error that a global shadow map reparameterization seeks to reduce. Figure 14 shows the same measure of perspective aliasing error along the entry faces of the view

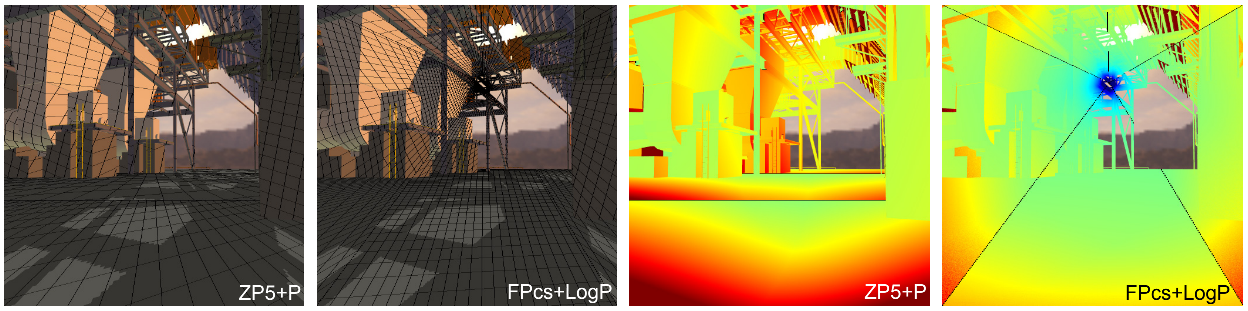


Fig. 15: ***Comparison in a power plant model.*** *The light is placed almost directly in front of the viewer. This is the dueling frustum case that is difficult for single shadow map algorithms to handle. Both FP and ZP algorithms can handle this situation well, though FPcs+LogP produces less error than ZP5+P. A face split is visible in the error image for FPcs+LogP. The image resolution is 512×512 and the total shadow map resolution is 1024×1024. ($f/n = 1000$, $\theta = 30°$)*

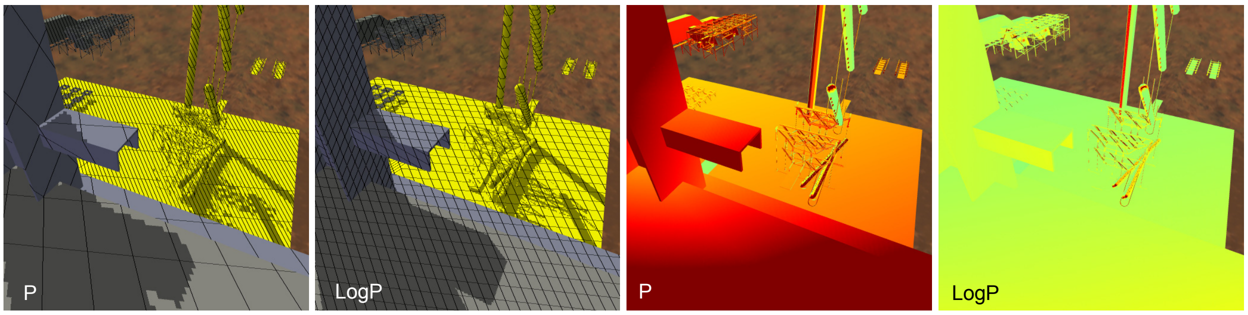


Fig. 16: ***Comparison with single shadow map in a power plant model.*** *Here we compare the P and LogP parameterizations. The image resolution is 512×512 and the shadow map resolution is 1024×1024. Grid lines are shown for every 10 texels. ($f/n = 500$, $\theta = 30°$)*

frustum. These are the faces selected for this light position by Equation 46 to compute δ_b for the error bound. Figure 13 shows the projection error that depends on surface orientation relative to the light and eye. We calculate the projection error as $\cos\psi_e / \cos\psi_l$. Note that we do not show the storage factor S directly because it is a measure of error over the entire view frustum.

Figure 12 shows a comparison between various shadow map algorithms. The aliasing is extremely high near the viewer for the standard shadow map, but improves with distance from the viewer. The FPc+P algorithm is comparable to Kozlov's perspective warped cubemap algorithm [2004] except that the LiSPSM parameter is used for warping instead of the PSM parameter. This gives a more uniform error distribution. FPcs+LogP has lower error than FPcs+P due to the better parameterization. ZP5+P is similar to cascaded shadow maps [Engel 2007], but adds warping for further error reductions. ZP5+P always renders 5 shadow maps while FPcs+LogP can render anywhere from 1 to 7 shadow maps. For this view, FPcs+LogP renders only 3. FPcs+LogP has the most uniform distribution of error.

Figure 15 shows a dueling frustum situation which is especially difficult for single shadow map algorithms to handle. Here FPc+LogP produces less error than ZP5+P. The portion of the image around the light direction is oversampled for surfaces far from the viewer.

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
Po	865	3.65×10^6	4.22×10^3	2.53×10^5	2.32×10^4	1
P	865	2.00×10^6	2.30×10^3	1.59×10^5	1.45×10^4	1
LogP	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+LogP	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+LogP	5.98	101	16.8	38.4	3.51	7
FP+LogP	5.98	14.8	2.48	10.9	1	3.6
FPc+LogP	5.98	25.4	4.24	15.4	1.40	3.6
FPcs+LogP	5.98	27.7	4.62	15.6	1.43	3.8

Table IV: ***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+LogP and its variations also have the lowest maximum, and mean storage factor. The values in the table do not include the $1/\cos^2\theta$ factor that arises in the storage factor. ($f/n = 1000, \theta = 30°$)*

Figure 16 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 1 shows FP+LogP used with point lights. We compare with Kozlov's algorithm [Kozlov 2004] (essentially FP+P but using the PSM parameter for the perspective warping). The LogP parameterization also provides lower error for point lights.

Table IV shows the variation in perspective aliasing error in terms of 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 Po, P, 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 17 that LogP provides significant improvement 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 all k. With this light position,

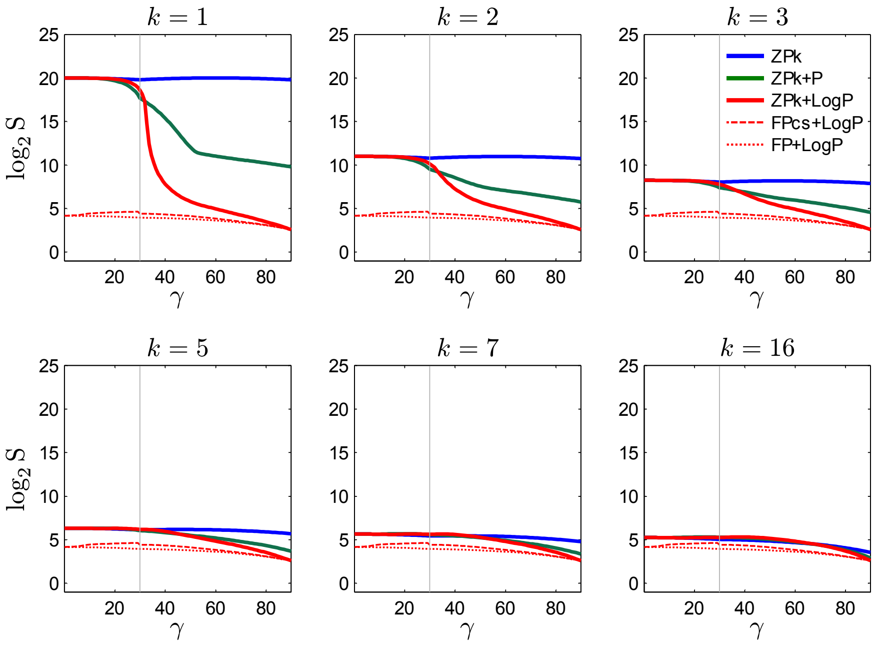
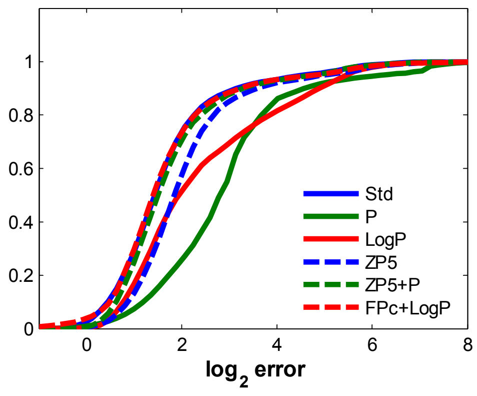
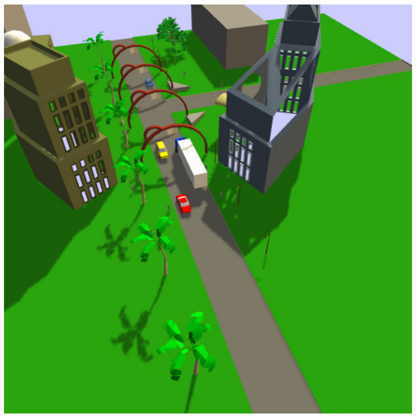


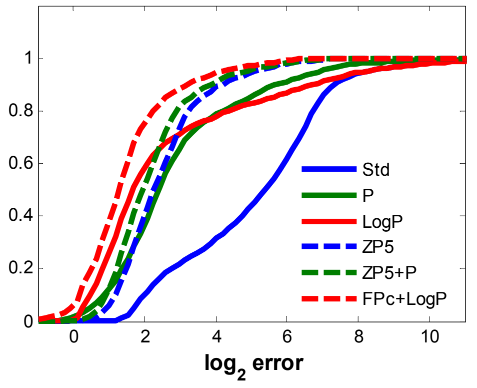
Fig. 17: ***z-partitioning using various parameterizations.*** *z-partitioning leads to significant error reductions but requires many partitions to converge to the same error as a face partitioning scheme that uses a logarithmic+perspective parameterization.* ($f/n = 1024$, $\theta = 30°$)

increasing the number z-partitions has no effect on the parameterization. This is not the case for uniform and perspective parameterizations. Figure 17 shows that for other light positions, increasing k initially produces drastic reductions in error that then trail off. With large k, warping makes less of a difference. For comparison, the error for FP+Log and FPcs+LogP are also shown in Figure 17. A small rise in FPcs+LogP can be seen at $\gamma = \theta$ where a new face partition appears and is split. As the number of z-partitions increases, the error begins to approach that of the FP+LogP. We have shown the error for the ZPk algorithms with $k = 5$ and $k = 7$ because these are the maximum number of shadow maps required for the FP and FPc algorithms (5) and for the FPcs algorithm (7). On average, however, the number of shadow maps used by FP, and FPc is 3.6, and for FPcs the average number is only 3.8.

The statistics reported in Table IV 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 resulting error in the image for a fixed view. Figure 18 shows the cumulative distribution of the error. The value y at a position x on the curve is the fraction of pixels with error less than x (i.e. the further the curve is shifted left, the better). 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 18, 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, FPc+LogP can produce significantly less error than other parameterizations.







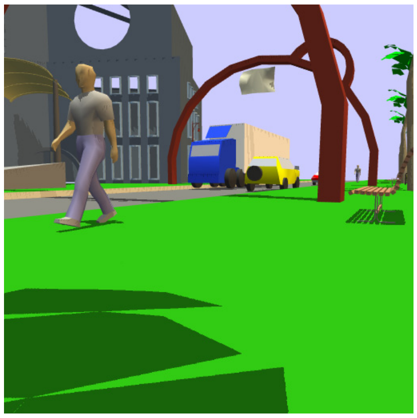


Fig. 18: ***Cumulative aliasing error distribution over randomly sampled light directions for several algorithms.*** *The graphs on the left were generated for the views on the right. The area of the projected texel was used for the error metric. The actual benefit of warping and partitioning methods depends on the location and orientation of surfaces within the view frustum.*

7.1 Discussion

Having examined the detailed analysis for LogPSMs, we can now discuss the advantages of each LogPSM algorithm (LogP, ZP+Log, and FP+LogP). Because rendering multiple shadow maps incurs a performance cost, we want an algorithm that gives the most error reduction with the fewest number of shadow maps.

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 just 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.

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 some parts of the frustum, but this comes at the expense of higher error elsewhere. For instance, a standard shadow map will typically have less error than a LogPSM at the far plane, but much higher error at the near plane.

The analysis in this section provides additional supporting evidence for the conclusion of Lloyd et al. [2006] that in the absence of logarithmic rasterization, ZPk+P is the best algorithm for achieving the low error with just a few shadow maps. Figure 17 shows that when even with just 2 partitions, ZP2 has error over the entire range of γ that is nearly as low as that of the minimum error of a single shadow map perspective warping scheme (ZP1+P). When k is small, the additional complexity of adding warping (ZPk+P) is justified by substantial error reductions over a large range of γ, but the benefit diminishes as k increases. With a large enough k, ZPk can come fairly close to matching the error of FPcs+LogP.

One of the most important things that can be done to improve the error of all the algorithms examined in this paper is to decrease the f_e/n_e ratio. This means that the near plane of the view frustum should be pushed away from the viewer as far as possible. In order to avoid near plane clipping of scene geometry, it is often desirable, however, to keep the near plane as close to the viewer as possible, though this approach can be overly conservative because the regions near the viewer are often empty. One possibility for reconciling these competing requirements is to use a conservatively small n_e, but compute the optimal shadow map parameterization/partitioning using a *pseudo-near plane* n_p, where $n_p > n_e$. For example, we could compute the partitions of ZPk+P using n_p instead of n_e, expand partition 1 to include n_e, and use a uniform parameterization instead of a perspective one in partition 1 so as to concentrate most of the error towards n_e. This allows us to choose n_p using a heuristic or a cheap estimate of the closest point in the view. For the rare cases when our n_p is not small enough, we still have shadow map coverage due the more conservative n_e, albeit with higher error, while maintaining low error in the rest of the frustum. (For details on using a pseudo-near plane to optimize parameterizations see [Lloyd 2007]). In essence, n_p gives an otherwise scene-independent algorithm some information about the parts of the frustum that the scene is likely to occupy.

7.1.1 *Limitations.* The biggest limitation for LogPSMs is that logarithmic rasterization is not available on current GPUs. It can be simulated with a fragment program, but it is considerably slower than linear rasterization. Logarithmic rasterization may be implemented efficiently in hardware through incremental modifications to existing rasterizers [Lloyd et al. 2007] or in software on an architecture such as Larrabee [Seiler et al. 2008].

To get the most benefit out of the LogPSM parameterization over all light directions, a face partitioning scheme must be used. 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.

The other limitations of LogPSMs are common to other warping techniques. Warping creates a non-uniform distribution of depth values, which can make it more difficult to select a suitable constant bias to remove self-shadowing artifacts. This is less of an issue for slope-scaled bias. Warping tends to increase the instability of texture coordinate derivative calculations on surfaces nearly parallel to the light (some noise can be seen in Figure 12). The distortion of filter kernels by the parameterization can also make spatial filtering more complicated.

Another limitation of all the scene-independent techniques examined in this paper is that they only handle perspective aliasing error. This simplification makes the algorithms easier to implement for real-time applications, but because they have no knowledge of surface orientation, they cannot handle the projection factor. Therefore they cannot guarantee that aliasing is completely removed from an image. In order to offset any remaining perspective aliasing, the shadow map must already be several times larger than the image. To reduce projection aliasing to acceptable levels on *most* surfaces, the shadow map must typically be several times larger still, but even this may 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 shadow map 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.

7.1.2 *Reducing projection aliasing.* In order to handle projection aliasing, the orientation of surfaces in the scene must be taken into account, i.e. the projection factor. In general, this requires partitioning. ZP and FP algorithms perform partitioning, but only according to the perspective factor. Adaptive methods partition according to both the perspective and projective factors and can thus produce much lower error. The more finely the shadow map is partitioned, the better it can adapt to the projection factor. In the limit each partition consists of a single pixel, resulting in the irregular shadow map algorithm.

But partitioning incurs the extra cost of both determining where to partition the shadow map and handling the irregularity introduced by dependence on arbitrary scene geometry. For example, samples in irregular shadow maps no longer correspond to locations on a regular grid, which means more work is required to access them. Sample locations and connectivity have to be stored explicitly, which requires more storage per sample and makes filtering operations more complicated. Adaptive shadow maps partition at a coarser granularity than the irregular shadow map, which leads to higher error, but their samples can be computed and accessed more cheaply, which makes it easier to create more samples to offset the error. We would like to further investigate the trade-offs between performance and error with varying partition granularity, and at what error/performance threshold it makes more sense to use an irregular shadow map instead of an adaptive shadow map.

Combining warping with adaptive shadow maps can reduce the number of partitions required to reduce the error in the scene. Geigl and Wimmer [2007b; 2007a] report a reduction in error by using an adaptive algorithm on top of a global perspective parameterization. We would also like to investigate whether replacing the perspective parameterization with a LogPSM parameterization would yield further improvements.

Conclusion

We have presented a logarithmic perspective parameterization for shadow maps and have shown how it can be used to replace perspective or uniform parameterizations in existing algorithms. With proper hardware support, LogPSMs would have the same good performance as the algorithms upon which they are based, but with lower error. We have conducted an in-depth analysis of aliasing error to determine just how much improvement LogPSMs can give over existing methods. We have shown that the error for perspective warping is $O(f/n)$, where f and n are the near and far plane distances. In contrast, the LogPSM parameterization has an error that is $O(\log(f/n))$, which is close to the lower bound for global, scene-independent parameterizations. We have also discussed various implementation details for LogPSMs and shadow maps in general, and based on the error analysis make recommendations of which algorithms are best to use under various circumstances. For future work we would like to investigate the use of the logarithmic perspective parameterizations with algorithms such as adaptive shadow maps, which can handle projection aliasing.

Acknowledgements

We would like to thank Ben Cloward for the robot model, Aaron Lefohn and Taylor Holiday for the use of the town model, as well as David Feng, Nico Galoppo, and the anonymous reviewers for their helpful comments. This work was supported in part by an NSF Graduate Fellowship, an NVIDIA Fellowship, ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583 and 0404088, DARPA/RDECOM Contract N61339-04-C-0043 and the Disruptive Technology Office.

REFERENCES

Aila, T. and Laine, S. 2004. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*. Eurographics Association, 161–166.

Annen, T., Mertens, T., Bekaert, P., Seidel, H.-P., and Kautz, J. 2007. Convolution shadow maps. In *Rendering Techniques 2007: Eurographics Symposium on Rendering*, J. Kautz and S. Pattanaik, Eds. Eurographics / ACM SIGGRAPH Symposium Proceedings, vol. 18. Eurographics, Grenoble, France, 51–60.

Annen, T., Mertens, T., Seidel, H.-P., Flerackers, E., and Kautz, J. 2008. Exponential shadow maps. In *Proceedings of the 2008 Conference on Graphics Interface*. 155–161.

Arvo, J. 2004. Tiled shadow maps. In *Proceedings of Computer Graphics International 2004*. IEEE Computer Society, 240–247.

Arvo, J. 2007. Alias-free shadow maps using graphics hardware. *Journal of Graphics Tools 12,* 1, 47–59.

Brabec, S., Annen, T., and Seidel, H.-P. 2002. Practical shadow mapping. *Journal of Graphics Tools 7,* 4, 9–18.

Chan, E. and Durand, F. 2004. An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering*. Eurographics Association, 185–195.

Chong, H. 2003. *Real-Time Perspective Optimal Shadow Maps*. Senior Thesis, Harvard University.

Chong, H. and Gortler, S. 2004. A lixel for every pixel. In *Proceedings of the Eurographics Symposium on Rendering*. Eurographics Association, 167–172.

Chong, H. and Gortler, S. 2007. Scene optimized shadow mapping. Tech. Rep. TR-07-07, Harvard University.

Crow, F. C. 1977. Shadow algorithms for computer graphics. *ACM Computer Graphics 11,* 3, 242–248.

Donnelly, W. and Lauritzen, A. 2006. Variance shadow maps. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM Press, New York, NY, USA, 161–165.

Engel, W. 2007. Cascaded shadow maps. In *ShaderX*^{5}, W. Engel, Ed. Charles River Media, 197–206.

Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. 2001. Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001*. 387–390.

Forsyth, T. 2006. Making shadow buffers robust using multiple dynamic frustums. In *ShaderX*^{4}, W. Engel, Ed. Charles River Media, 331–346.

Giegl, M. and Wimmer, M. 2007a. Fitted virtual shadow maps. In *Proceedings of Graphics Interface 2007*. 159–168.

Giegl, M. and Wimmer, M. 2007b. Queried virtual shadow maps. In *Proceedings of ACM SIGGRAPH 2007 Symposium on Interactive 3D Graphics and Games*. ACM Press, 65–72.

Hubbard, J. H. and Hubbard, B. B. 2001. *Vector Calculus, Linear Algebra, and Differential Forms: A Unified Approach (2nd Ed.)*. Prentice Hall.

Johnson, G., Mark, W., and 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., and Mark, W. R. 2005. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph. 24,* 4, 1462–1482.

Kozlov, S. 2004. Perspective shadow maps: Care and feeding. In *GPU Gems*, R. Fernando, Ed. Addison-Wesley, 214–244.

Lauritzen, A. and McCool, M. 2008. Layered variance shadow maps. In *Proceedings of the 2008 Conference on Graphics Interface*. 139–146.

LEFOHN, A. E., SENGUPTA, S., AND OWENS, J. D. 2007. Resolution-matched shadow maps. *ACM Transactions on Graphics 26,* 4, 20.

LLOYD, B., TUFT, D., YOON, S., AND MANOCHA, D. 2006. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006.* Eurographics Association, 215–226.

LLOYD, D. B. 2007. Logarithmic perspective shadow maps. Ph.D. thesis, University of North Carolina at Chapel Hill.

LLOYD, D. B., GOVINDARAJU, N. K., MOLNAR, S., AND MANOCHA, D. 2007. Practical logarithmic rasterization for low-error shadow maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. Eurographics Association, 17–24.

MARTIN, T. AND TAN, T.-S. 2004. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering.* Eurographics Association, 153–160.

MCCOOL, M., WALES, C., AND MOULE, K. 2001. Incremental and hierarchical hilbert order edge equation using polygon rasteization. *Eurographics Workshop on Graphics Hardware*, 65–72.

REEVES, W., SALESIN, D., AND COOK, R. 1987. Rendering antialiased shadows with depth maps. In *Computer Graphics (ACM SIGGRAPH '87 Proceedings).* Vol. 21. 283–291.

SALVI, M. 2008. Rendering filtered shadows with exponential shadow maps. In *ShaderX*^{6}, W. Engel, Ed. Charles River Media.

SCHERZER, D., JESCHKE, S., AND WIMMER, M. 2007. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, J. Kautz and S. Pattanaik, Eds. Eurographics, 45–50.

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics 27,* 3, To appear.

SEN, P., CAMMARANO, M., AND HANRAHAN, P. 2003. Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003) 22,* 3 (July), 521–526.

SINTORN, E., EISEMANN, E., AND ASSARSSON, U. 2008. Sample based visibility for soft shadows using alias-free shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2008.* To appear.

STAMMINGER, M. AND DRETTAKIS, G. 2002. Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002.* 557–562.

TADAMURA, K., QIN, X., JIAO, G., AND NAKAMAE, E. 1999. Rendering optimal solar shadows using plural sunlight depth buffers. In *Computer Graphics International 1999.* 166.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings).* Vol. 12. 270–274.

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Proceedings of the Eurographics Symposium on Rendering.* Eurographics Association, 143–152.

ZHANG, F., SUN, H., XU, L., AND LUN, L. K. 2006. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of ACM International Conference on Virtual Reality Continuum and Its Applications 2006.* ACM SIGGRAPH, 311–318.

ZHANG, F., XU, L., TAO, C., AND SUN, H. 2006. Generalized linear perspective shadow map reparameterization. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications.* ACM Press, New York, NY, USA, 339–342.

Received September 2007; revised May 2008; accepted June 2008

APPENDIX

In this appendix we analyze various shadow map parameterizations for the faces of the view frustum. We begin with the parameterizations F_b for the three varieties of δ_b. We also discuss the uniform, perspective, logarithmic, and logarithmic+perspective parameterizations described in Section 5. We then analyze the error of parameterizations applied to the entire view frustum.

Parameterizations based on δ_b

We begin by looking at some graphs of the various spacing distributions functions δ_b that are based on the bound for maximum perspective aliasing error. Figure 19 shows how $\delta_{b,t}^{\text{end}}$ on the far face changes with light position. The $\cos\phi_{ly}$ term is responsible for the variation in the shape of the function. Close to the face, $\delta_{b,t}^{\text{end}}$

resembles an absolute value function with a rounded tip. As $l_z \to \infty$, $\cos\phi_{ly}$ converges to a constant and the spacing distribution function converges to uniform. Moving the light in y simply translates the function along the v-axis.

Figure 20 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 $\delta^{\text{end}}_{b,t}$ extruded along the y direction and scaled by y.

Figure 21 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 distribution 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 distribution function converges to a linear function. Once again, it is the $\cos\phi_{ly}$ term close when the light is close to the face that accounts for the variation in the shape of the spacing distribution function.

From Equation 67 we can see that if $\delta_l = \rho_b\delta_b$, then $R_b = \rho_b$. We note that $\rho^{\text{side}}_{b,s}$ varies with v. Therefore we take $R^{\text{side}}_{b,s} = \max_v(\rho^{\text{side}}_{b,s}(v))$. Figure 22 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. This is due to the effect of the $\cos\phi_l$ term. R_b is related to the integral of $\cos\phi_l$ over the face. When the light is close to the face or off to the side, the $\cos\phi_l$ term becomes smaller. It is largest for points directly under the light. For an overhead directional light, $\cos\phi_l = 1$ everywhere. It is for this position that R_b reaches its maximum value. Therefore, the maximum critical resolution factors, R_b, can be computed from $\lim_{l_z\to\infty}\rho_b$, or equivalently, by plugging $\cos\phi_l = 1$ into Equations 54, 57, and 58, and computing the corresponding ρ_b:

$$\mathsf{R}^{\text{end}}_{b,t} = \frac{1}{\cos\theta} \tag{86}$$

$$\mathsf{R}^{\text{side}}_{b,s} = \frac{1}{\cos\theta} \tag{87}$$

$$\begin{aligned}\mathsf{R}^{\text{side}}_{b,t} &= \frac{y_0\log(y_1/y_0)}{W_e\cos\theta} = \frac{\log(f_e/n_e)}{2\tan\theta\cos^2\theta}\\ &= \frac{\log(f_e/n_e)}{\sin 2\theta}\end{aligned} \tag{88}$$

We have substituted $W_e = 2n_e\tan\theta$. The critical resolution factors are at most $O(1)$ for an end face and $O(\log(f_e/n_e))$ for a side face.

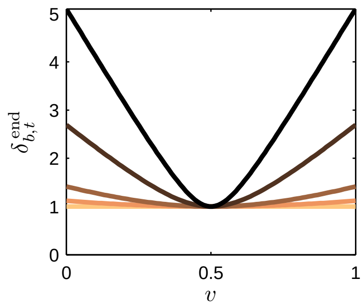


Fig. 19: ***Error bound spacing distribution on an end face.*** *This graph shows $\delta^{end}_{b,t}$ for the light at $l_y = 0$ and $l_z = \sigma(y_1 - y_0)$. From dark to light, $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The $\cos\theta$ term has been factored out.*

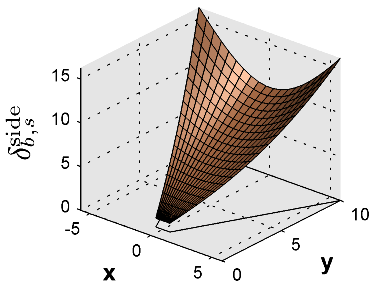


Fig. 20: ***Error bound spacing distribution for*** s ***on a side face.*** *This graph shows* $\delta^{side}_{b,s}$ *with the light centered over the face.*

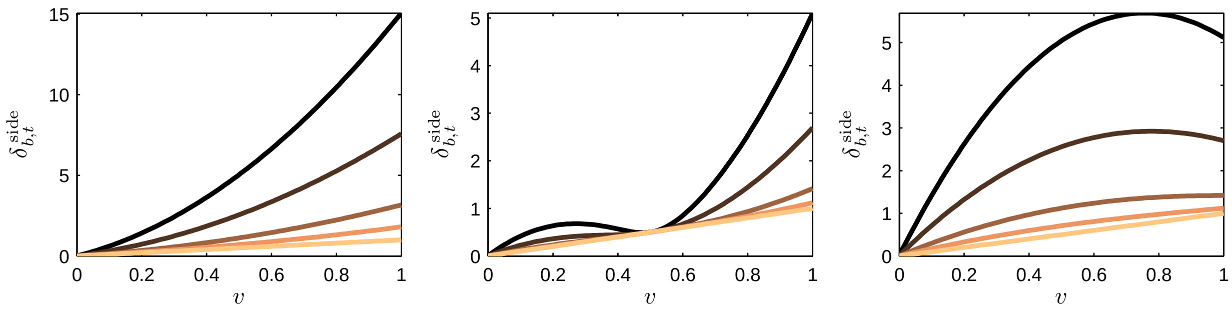


Fig. 21: ***Error bound spacing distribution for*** t ***on a side face.*** *These graphs show* $\delta^{side}_{b,t}$ *for the light at* $l_y = \kappa(y_1 - y_0)$ *and* $l_z = \sigma(y_1 - y_0)$*. From left to right,* $\kappa = -0.5, 0.5, 1$*. From dark to light,* $\sigma = 0.1, 0.2, 0.5, 1, \infty$*. The frustum parameters are* $\theta = 45°$*,* $n_e = 1$*, and* $f_e = 1000$*.*

Uniform parameterization

A standard shadow map uses a parameterization that has a uniform spacing distribution function:

$$\delta_{un} = 1 \tag{89}$$

$$\rho_{un} = 1. \tag{90}$$

From Equation 67 we get for an end face:

$$R^{\text{end}}_{un,t} = \max_v \left(\frac{1}{\delta^{\text{end}}_{b,t}(v)} \right) = \max_v \left(\frac{\cos \phi_{ly}(v)}{\cos \theta} \right) \tag{91}$$

Figure 23 shows R_{un} on an end face and a side face for various light positions. Note that R_{un} reaches its maximum whenever 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 23 to Figure 22, 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 scaled differs by several orders of magnitude. This difference arises from the $y(v)$ term in $\delta^{\text{side}}_{b,s}$ and $\delta^{\text{side}}_{b,t}$, which varies by a factor of f_e/n_e over the face.

R_{un} can be computed most simply by setting the $\cos \phi_l$ term in δ_b to 1 and applying the equivalent of Equation 91 for the appropriate face. $\mathsf{R}^{\text{side}}_{un}$ has its maximum when the $y(v)$ term of $\delta^{\text{side}}_{b,s}$ reaches its minimum

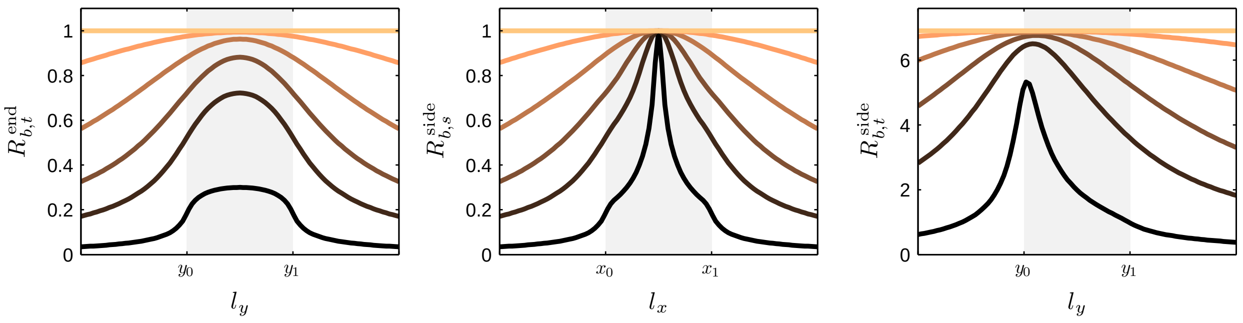


Fig. 22: ***Critical resolution factor for error bound parameterizations.*** *From left to right, $R^{end}_{b,t}$, $R^{side}_{b,s}$, and $R^{side}_{b,t}$ for the light at varying positions. The plots show $l_z = \sigma(y_1 - y_0)$, where from dark to light, $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The grayed 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 $R^{side}_{b,s}$.*

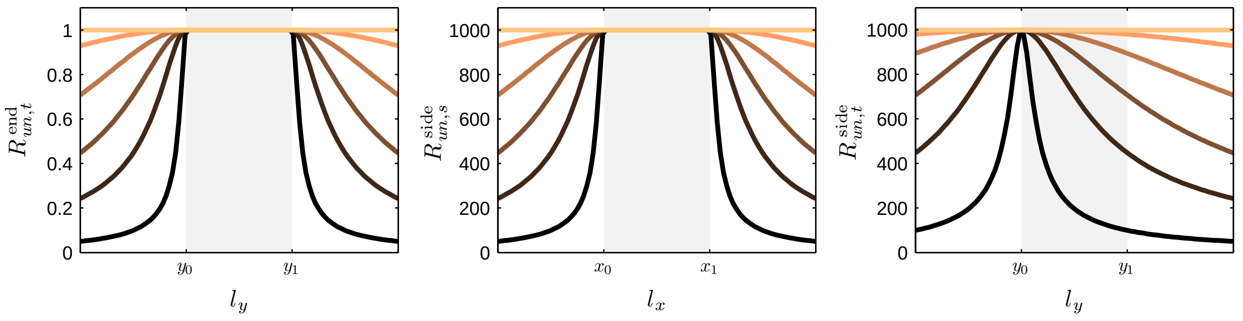


Fig. 23: ***Critical resolution factor for uniform parameterization.*** *From left to right, $R^{end}_{un,t}$, $R^{side}_{un,s}$, and $R^{side}_{un,t}$ for various light positions. For all graphs $l_z = \sigma(y_1 - y_0)$, where from dark to light, the plots are for $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The grayed out region indicates light positions over the face. The frustum parameters are $\theta = 45°$, $n_e = 1$, and $f_e = 1000$. The $\cos\theta$ term has been factored out of $R^{end}_{un,t}$ and $R^{side}_{un,s}$.*

value of y_0. Putting all of this together, we get:

$$\mathsf{R}^{\text{end}}_{un,t} = \frac{1}{\cos\theta} \tag{92}$$

$$\mathsf{R}^{\text{side}}_{un,s} = \frac{y_1}{y_0\cos\theta} = \frac{f_e}{n_e\cos\theta} \tag{93}$$

$$\begin{aligned}\mathsf{R}^{\text{side}}_{un,t} &= \frac{(y_1 - y_0)}{W_e\cos\theta} = \frac{f_e - n_e}{2n_e\tan\theta\cos^2\theta} \\ &= \frac{(f_e/n_e) - 1}{\sin 2\theta}.\end{aligned} \tag{94}$$

The maximum critical resolution for the uniform parameterization is identical to that of the error bound parameterization for the end faces.

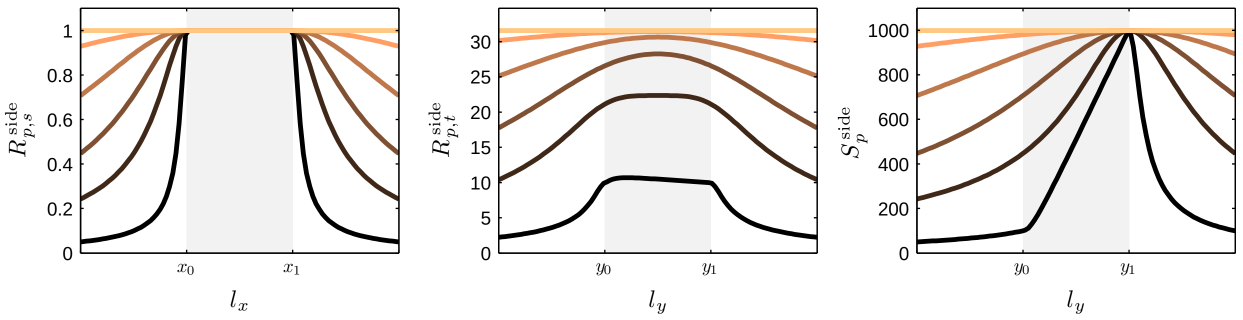


Fig. 24: ***Optimal critical resolution factors for perspective parameterization.*** *From left to right $R_{p,s}^{side}$, $R_{p,t}^{side}$, and S_p^{side} for various light positions. When parameterizing with a perspective matrix, $\delta_{p,s}^{side}$ and $\delta_{p,t}^{side}$ are coupled. S_p^{side} 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 $f_e = 1000$. The $\cos\theta$ term has been factored out of $R_{p,s}^{side}$.*

Perspective parameterization

Existing warping methods use perspective projections. The normalized spacing distribution functions on a side face for the perspective parameterization $\mathbf{F}_p$ in Equation 73 can be computed by taking the multiplicative inverse of the derivatives of $\mathbf{F}_{p,s}$ and $\mathbf{F}_{p,t}$ w.r.t. u and v, respectively:

$$\begin{aligned} \delta_{p,s}^{\text{side}} &= \frac{y(v)+a}{p_0\left(x_1(y_1)-x_0(y_1)\right)} \\ &= \frac{(y(v)+a)}{(y_1+a)} \end{aligned} \tag{95}$$

$$\begin{aligned} \delta_{p,t}^{\text{side}} &= -\frac{(y(v)+a)^2}{p_2(y_1-y_0)} \\ &= \frac{(y(v)+a)^2}{(y_0+a)(y_1+a)}. \end{aligned} \tag{96}$$

$\delta_{p,s}^{\text{side}}$ is constant in u and increases linearly in v, while $\delta_{p,t}^{\text{side}}$ is quadratic in v.

Figure 24 shows the optimal $R_{p,s}^{\text{side}}$, $R_{p,t}^{\text{side}}$, and S_p^{side}. The optimal $R_{p,s}^{\text{side}}$ is computed as:

$$\begin{aligned} \min_a R_{p,s}^{\text{side}} &= \min_a \max_{(u,v)\in\mathcal{F}} \left(\frac{\delta_{p,s}^{\text{side}}}{\delta_{b,s}^{\text{side}}}\right) \\ &= \min_a \max_{(u,v)\in\mathcal{F}} \left(\frac{y1}{(y1+a)}\frac{(y(v)+a)}{y(v)}\frac{\cos\phi_{lx}(u)}{\cos\theta}\right). \end{aligned} \tag{97}$$

The maximum in $R_{p,s}^{\text{side}}$ is achieved when $y(v) = y_0$. $R_{p,s}^{\text{side}}$ is minimized when $a = 0$, which leaves only the $\cos\phi_{lx}/\cos\theta$ term in Equation 97. Geometrically, setting $a = 0$ causes the face to be stretched out to fill the entire shadow map, maximizing the use of the available resolution. With $a = 0$, Equation 97 is essentially the same as Equation 91. The optimal a parameter for $R_{p,t}^{\text{side}}$ varies with the position of the light. For a directional light directly above the face, $a_{\text{opt}} = \sqrt{y_0 y_1}$, the same optimal parameter as used by LiSPSMs. The maximum occurs at either $y(v) = y_0$ or $y(v) = y_1$. Using these optimal parameters for s and t for the

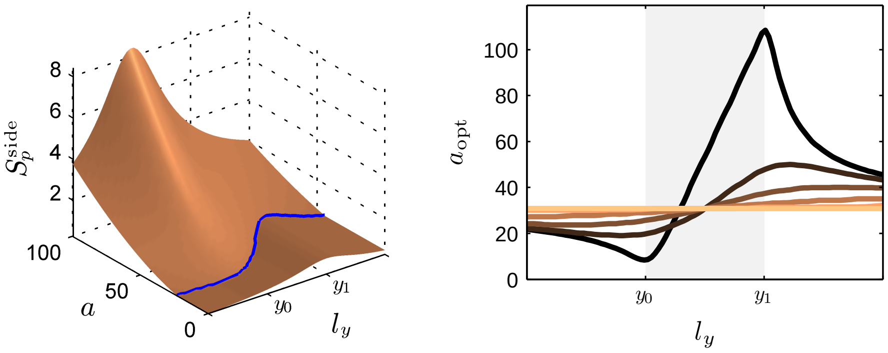


Fig. 25: ***Optimal perspective parameter.*** (Left) *When the errors in the s and t are combined, a range of parameters (those below the curve) produce the same storage factor S_p^{side} for each light position ($l_y = 0.5y_1$.)* (Right) *The maximum parameter value in optimal range for $l_z = \sigma(y_1 - y_0)$, where from dark to light $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The frustum parameters are $\theta = 45°$, $n_e = 1$, and $f_e = 1000$.*

overhead light case gives the optimal $\mathsf{R}_p^{\text{side}}$:

$$\min_a \mathsf{R}_{p,s}^{\text{side}} = \frac{1}{\cos\theta} \tag{98}$$

$$\begin{aligned}\min_a \mathsf{R}_{p,t}^{\text{side}} &= \frac{y_0(y_1 - y_0)}{W_e\sqrt{y_0 y_1}\cos\theta} = \frac{(f_e - n_e)}{2\tan\theta\cos^3\theta\sqrt{n_e f_e}} \\ &= \frac{f_e/n_e - 1}{\sin 2\theta\sqrt{f_e/n_e}\cos\theta} \\ &\approx O(\sqrt{f_e/n_e}).\end{aligned} \tag{99}$$

The parameterization $\mathbf{F}_p$ is usually implemented with a projective matrix, in which case the parameter a is the same for s and t. This means that $R_{p,s}^{\text{side}}$ and $R_{p,t}^{\text{side}}$ can not be optimized separately, as we have done. Rather they should be optimized simultaneously by optimizing the storage factor S_p^{side}. Lloyd et al. [2006] noted that for overhead directional lights, there exists a range of parameter values for which S_p^{side} is constant and is minimal. Figure 25 shows that a range of optimal parameter values for S_p^{side} also exists for point lights. S, however, does not measure the error due to shearing. Shearing decreases with increasing a. Therefore it is advisable to choose the largest a possible on the equivalent range. For an overhead directional light, this corresponds to the LiSPSM parameter. Using this value we can compute $\mathsf{S}_p^{\text{side}}$:

$$\begin{aligned}\min_a \mathsf{S}_p^{\text{side}} &= \frac{y_1 - y_0}{W_e\cos^2\theta} = \frac{(f_e - n_e)}{2n_e\tan\theta\cos^2\theta} \\ &= \frac{(f_e/n_e - 1)}{\sin 2\theta} \\ &\approx O(f_e/n_e).\end{aligned} \tag{100}$$

The optimal range for a includes $a = 0$. If $a = 0$, then $R_{p,s}^{\text{side}}$ is minimized and most of the error is in t.

Logarithmic parameterization

We have now shown that the uniform parameterization gives error for end faces on the same order as that of the error bound parameterization. The perspective parameterization can do the same for the s on side

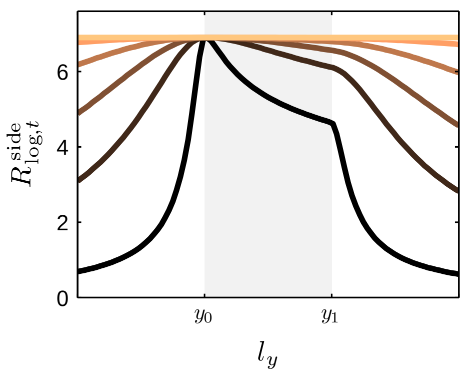


Fig. 26: ***Critical resolution factor for logarithmic parameterization.*** *For all graphs $l_z = \sigma(y_1 - y_0)$, where from dark to light, the plots are for $\sigma = 0.1, 0.2, 0.5, 1, \infty$. The grayed out region indicates light positions over the face. The frustum parameters are $\theta = 45^\circ$, $n_e = 1$, and $f_e = 1000$.*

faces. We will now show that a logarithmic parameterization can do the same for t. The normalized spacing distribution for Equation 74 is:

$$\delta^{\text{side}}_{\log,t} = \log\left(\frac{y_1 + a}{y_0 + a}\right)\frac{y + a}{y_1 - y_0} \tag{101}$$

This spacing distribution is linear, as is that of the $\delta^{\text{side}}_{b,t}$ for a directional light (see Figure 21). Figure 26 shows $R^{\text{side}}_{\log,t}$. Because $\delta^{\text{side}}_{\log,t}$ is not a very good match for the $\cos\phi_l$ factor, the error is not as low as it could be when the light is close to the face, but it is on the same order as $R^{\text{side}}_{b,t}$. When the $\cos\phi_l$ factor in $\delta^{\text{side}}_{b,t}$ becomes 1 for a directional light, the optimal $a_{\text{opt}} = 0$ gives a perfect match for the $y(v)$ term. With this parameter we get:

$$\begin{aligned}\min_a \mathsf{R}^{\text{side}}_{\log,t} &= \frac{y_0 \log(y_1/y_0)}{W_e \cos\theta} \\ &= \frac{\log(f_e/n_e)}{\sin\ 2\theta},\end{aligned} \tag{102}$$

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

Logarithmic + perspective parameterization

The general spacing distribution for $F_{lp,t}$ is found by computing $(\mathrm{d}F_{lp,t}/\mathrm{d}v)^{-1}$:

$$\delta_{lp,t} = \frac{(y(v) + a)\Big((y(v) + a)(c_1 p_2 + c_2) + c_1 p_3\Big)}{c_0 c_1 p_3 (y_1 - y_0)}. \tag{103}$$

$\delta_{lp,t}$ has two real roots (in terms of $y(v)$):

$$\lambda_0 = -a, \qquad \lambda_1 = -\left(a + \frac{c_1 p_3}{c_1 p_2 + c_2}\right). \tag{104}$$

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

$$\delta_{lp,t} \sim (y(v) + a)(y(v) + b). \tag{105}$$

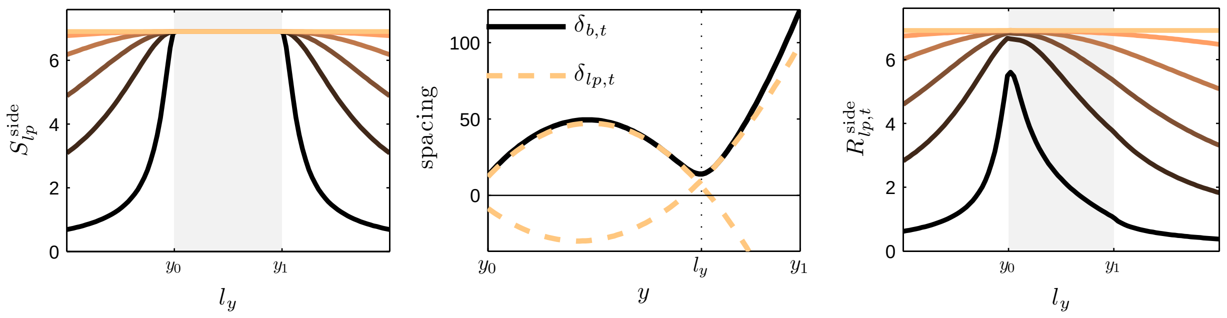


Fig. 27: ***Critical resolution and storage factors for logarithmic+perspective parameterizations.*** (Left) *Storage factor for the linear form of $\delta_{lp,t}^{side}$ that we use for LogPSMs.* (Middle) *The logarithmic + perspective parameterization can also generate parabolic spacing distribution functions that can give a good fit to $\delta_{b,t}^{side}$.* (Right) *The two parabola form gives a storage factor that approaches that of δ_b^{side}. 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 $f_e = 1000$.*

To compute c_1 and c_2 we need an additional constraint, which comes from the requirement that $F_{lp,t}(y_0) = 0$. 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:

$$c_1 = -\frac{(y_0 + a)(a - b)}{(y_0 + b)p_3} \tag{106}$$

$$c_2 = \frac{(y_0 + a)(ap_2 - bp_2 + p_3)}{(y_0 + b)p_3}. \tag{107}$$

Plugging these values into Equation 103 gives us the normalized spacing distribution:

$$\delta_{lp,t} = -\frac{(y(v) + a)(y(v) + b)}{c_0(a - b)(y_1 - y_0)}. \tag{108}$$

The constant c_0 can be computed from the constraint that $F_{lp,t}(y_1) = 1$:

$$c_0 = \frac{1}{\log\left(\frac{(y_0+a)(y_1+b)}{(y_0+b)(y_1+a)}\right)}. \tag{109}$$

When $b \to \infty$, the constants converge to those given in Section 5.3 and $\delta_{lp,t}$ converges to a linear spacing distribution. Figure 27 shows the optimal $S_{lp,t}^{\text{side}}$ with $b = \infty$. The a parameter also affects $R_{lp,s}^{\text{side}}$ and thus must be optimized simultaneously with $R_{lp,t}^{\text{side}}$ by optimizing $S_{lp,t}^{\text{side}}$. We find the optimal a parameter numerically. Over the wide range of parameters we tested, it appears that the optimal parameter is always $a = 0$. Note that when the light is over the face, $R_{lp,t}^{\text{side}}$ is higher than $R_{\log,t}^{\text{side}}$. However $\mathsf{R}_{\log,t}^{\text{side}}$ is still the same.

When b is finite, $\delta_{lp,t}$ is a parabola. Using two parabolas we can get a better fit to the curved portions of $\delta_{b,t}^{\text{side}}$ (see Figure 27). 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 27 also shows $S_{lp,t}^{\text{side}}$ with $y_s = l_y$, $a = 0$, and b chosen to optimize $S_{lp,t}^{\text{side}}$. The curves for $R_{lp,t}^{\text{side}}$ are very similar to those for $R_{b,t}^{\text{side}}$. 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 further explore the additional degrees of freedom in $\mathbf{F}_{lp}$ as future work.

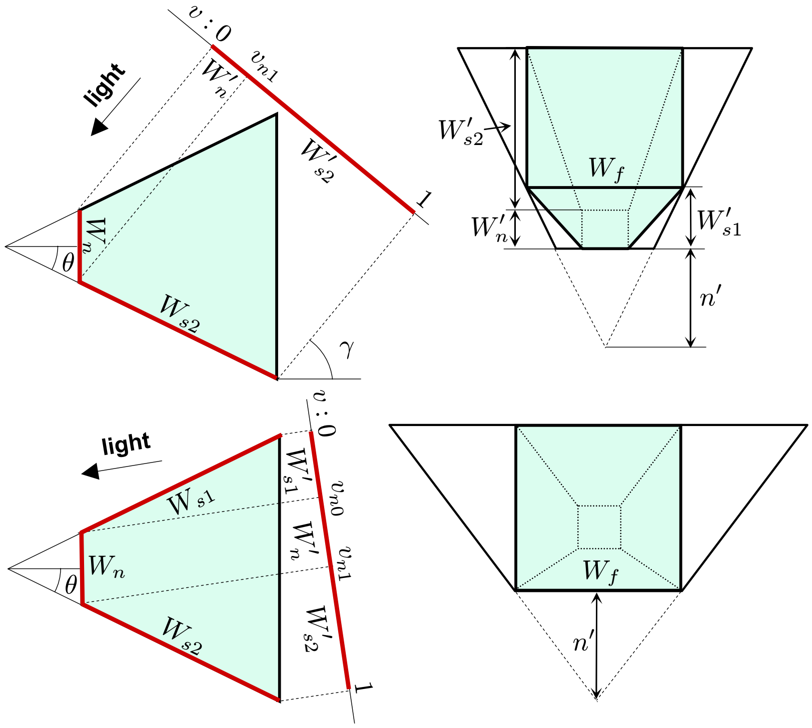


Fig. 28: ***Parameterizing the entire frustum for a directional light.*** *(Left) Side views of the view frustum. The exit faces of the view frustum are projected onto the light image plane, which is oriented perpendicular to the light direction. (Right) The light's view of the view frustum. The perspective warping frustum surrounding the view frustum is shown in black. On the top row $\gamma > \theta$ and on the bottom row $\gamma < \theta$, where γ is the angle between the light and view vector, and θ is half the field of view of the view frustum.*

Error analysis for parameterizations over entire view frustum

The error analysis for point lights that we performed on the frustum faces can be extended for analyzing single shadow map methods. We restrict our attention here, however, to directional lights in order to simplify the discussion. These methods are typically used for directional lights anyway, or for spot lights with a fairly narrow field of view.

We first derive equations that can be used to analyze the error for varying light directions. These equations are similar to those used by Zhang et al. [2006]. The main difference is that Zhang et al. analyze error along the view direction while we evaluate the error along the faces.

Figure 28 shows the light image plane and the light 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, Equation 46 dictates that we compute $\widetilde{M}$ using δ_b along the exit faces of the view frustum. The near face is an exit face for all $\gamma < 90°$. Both side faces are exit faces when $\gamma < \theta$, while only one of them is for $\gamma > \theta$. The various terms of δ_b can be computed using simple geometry on Figure 28:

$$W_{ly} = W'_n + W'_{s2} + \begin{cases} W'_{s1} & \gamma \in [0, \theta] \\ 0 & \gamma \in [\theta, 90^\circ] \end{cases} \tag{110}$$

$$W_{lx} = (n' + W_{ly}) \begin{cases} \frac{W_f}{n'} & \gamma \in [0, \theta] \\ \frac{W_f}{n'+W'_{s1}} & \gamma \in [\theta, 90^\circ] \end{cases} \tag{111}$$

$$d_e(v) = n_e + (f_e - n_e) \max\left(\frac{y(v_{n0}) - y(v)}{W'_{s1}}, 0, \frac{W_{ly} - y(v)}{W'_{s2}} \right) \tag{112}$$

$$y(v) = W_{ly} v \tag{113}$$

$$v_{n0} = \begin{cases} \frac{W'_{s1}}{W_{ly}} & \gamma \in [0, \theta] \\ 0 & \gamma \in [\theta, 90^\circ] \end{cases} \qquad v_{n1} = v_{n0} + \frac{W'_n}{W_{ly}} \tag{114}$$

$$W'_n = W_n \cos\gamma, \qquad W'_{s1} = W_s\,(1 - \cos(\theta - \gamma)) \qquad W'_{s2} = W_s \sin(\theta - \gamma) \tag{115}$$

$$W_n = W_e, \qquad W_f = W_n \frac{f_e}{n_e}, \qquad W_s = \frac{f_e - n_e}{\cos\theta}. \tag{116}$$

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 are 1. The spacing distribution functions computed for the various face parameterizations in the Appendix need to be modified slightly when parameterizing the frustum because the coordinate system we are using is not the same. We make the following substitutions: $y_0 = 0$, $y_1 = W_{ly}$, and $a = n'$.

Equipped with these equations we can now analyze the error distribution of the uniform, perspective, and logarithmic perspective parameterizations for a single frustum. We will assume throughout this discussion that $r_s = r_i$ and $r_t = r_j$. We will also drop the $\cos\theta$ term from δ_b so that the dependence of the error on a function of f_e/n_e can be more easily observed. Because the warping parameter n' which controls $\mathbf{F}_p$ and $\mathbf{F}_{lp}$ can diverge to ∞, we parameterize n' with η, which is defined over the finite range $[-1, 1]$. As given by Lloyd et al. [2006], η is parameterization of n' that corresponds to $n' = \infty$ at $\eta = -1$, $n' = n_e + \sqrt{n_e f_e}$ at $\eta = 0$, and $n' = n_e$ at $\eta = 1$. We modify their equations slightly for use with varying γ:

$$n'(\eta) = \frac{W_{ly}}{\alpha - 1} \begin{cases} \frac{\sqrt{\alpha}+1-\eta(\alpha-1)}{\eta+1} & \eta < 0 \\ \frac{\sqrt{\alpha}+1}{\eta\sqrt{\alpha}+1} & \eta \geq 0 \end{cases} \tag{117}$$

$$\alpha = \frac{f_e}{n_e}.$$

Figure 29 shows the error distribution using $\mathbf{F}_p$ and $\mathbf{F}_{lp}$ over all η. The graphs are shown over all v. $F_{p,s}(u, v)$ is constant in u, which is not shown. Let us consider first the case of an overhead light with $\gamma = 90^\circ$. The value of $\widetilde{M}_{p,s}$ (and $\widetilde{M}_{lp,s}$ because $F_{lp,s} = F_{p,s}$) attains its maximum at $v = 0$, decreases rapidly, and then descends more slowly to its minimum value at $v = 1$. At $\eta = 1$ the warping frustum matches the view frustum and the error is constant and minimal over all v. $\widetilde{M}_{p,t}$ shows a wider range of behavior. At $\eta = -1$, $\widetilde{M}_{p,t}$ degenerates to a uniform parameterization and looks very much like $\widetilde{M}_{p,s}$ with an initial, rapid descent followed by a more gradual one. At $\eta = 1$, $\widetilde{M}_{p,t}$ increases (linearly, in fact) from $v = 0$ to $v = 1$. At $\eta = 0$ the values at $v = 0$ and $v = 1$ are the same and the maximum error over all v is minimal. In general, over some range around $\eta = 0$ the minimum error over v does not occur at the endpoints. For the parameters used to generate Figure 29, this range is approximately $\eta \in [-0.4, 0.4]$. Note that η was derived such that

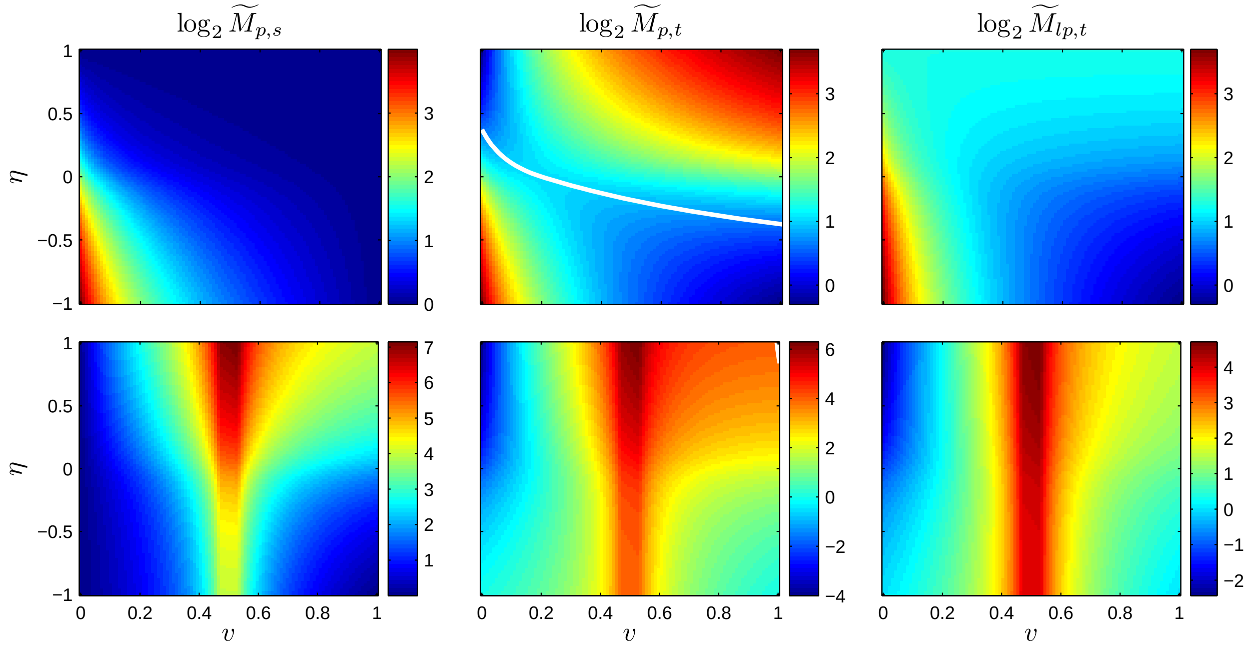


Fig. 29: ***Error distribution for varying warping parameter.*** *(Top) $\gamma = 90°$. (Bottom) $\gamma = 0°$. The white line on the graph for $\widetilde{M}_{p,t}$ is the location of the minimum error over v as a function of η. The frustum parameters are $n = 1$, $f = 16$, and $\theta = 30°$.*

the maximum error over v would decrease linearly along $v = 0$ for $\eta \in [-1, 0]$ and rise again linearly along $v = 1$ for $\eta \in [0, 1]$. This behavior can be seen in the graph. As with $\widetilde{M}_{p,s}$, the maximum of $\widetilde{M}_{lp,t}$ is always at $v = 0$, the minimum is at $v = 1$, and at $\eta = 1$ the error is uniform over all v.

Looking at the graphs for the light directly in front of the view frustum ($\gamma = 0°$) we see that the maximum error occurs on the farthest end of near plane at $v = v_{n1}$. This is true for all $\gamma \in [0°, 90°]$. When $\gamma = 90°$, $v_{n1} = 0$. The minimal maximum error is attained for all parameterizations at $\eta = -1$, which is why the single shadow map warping algorithms all degenerate to a uniform parameterization at $\gamma = 0°$.

In order to compute a function that gives us a smooth degeneration to a uniform parameterization as $\gamma \to 0°$ while maintaining low error, we examine the behavior of these functions over varying γ in Figure 30. Based on the analysis of Figure 29, we plot the error at $v = v_{n1}$, $v = 1$, and for $\widetilde{M}_{p,t}$, at the location of the minimum $v = v_{\min}$. The maximum value of all the plots is the critical resolution factor R and the distance between the maximum and minimum captures the variation in error over v. For an overhead light, both R and the variation in $\widetilde{M}$ 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 $R_{p,s}$ rises significantly, while $R_{lp,t}$ remains practically the same. At $\gamma = \theta$, there is a rapid shift in R for $\eta \in [0, 1]$. $\widetilde{M}_{p,s}$ also sees a rapid change at $v = v_{n1}$ for η in this range, but the shift in $R_{p,t}$ 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 31 shows the error over $\gamma \in [0°, 90°]$ for various shaping functions for the warping parameter. When each parameterization uses its optimal parameter without any shaping, the error for $\gamma < \theta$ is very high. The

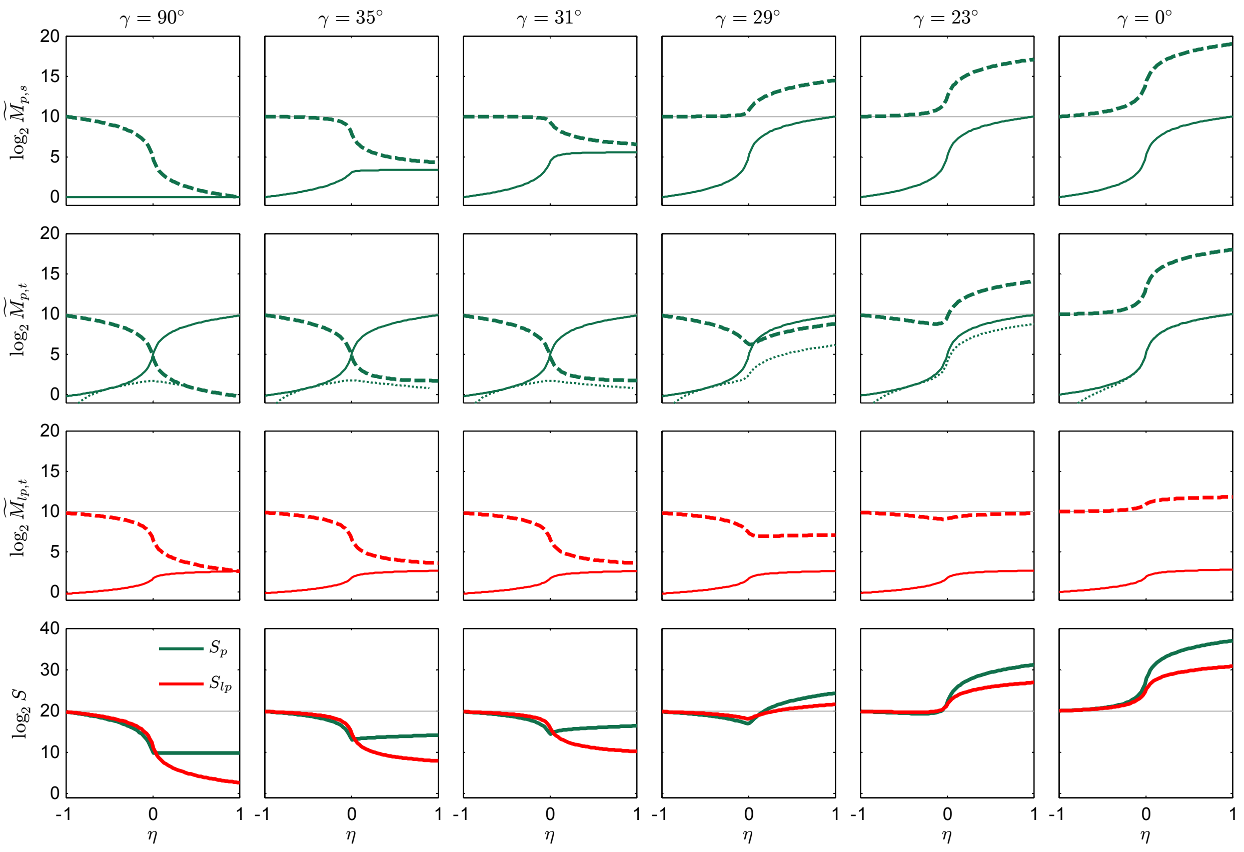


Fig. 30: ***Perspective error over all warping parameters for several light directions.*** *These graphs plot the error at $v = v_{n1}$ (heavy dotted line), $v = 1$ (thin solid line), and for $\widetilde{M}_{p,t}$, $v = v_{\min}$, where $v_{\min}$ is the location of the minimum value. S_p is constant over $\eta \in [0, 1]$ for $\gamma = 90°$. The frustum parameters are $n = 1$, $f = 1024$, and $\theta = 30°$.*

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 $\widetilde{M}_s$ for $\eta \neq -1$. $\widetilde{M}_{lp,s}$ has even higher error because η_{lp} is further from -1 than η_p. If we compute the optimal parameter with respect to 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 parameter that minimizes S_p and S_{lp} is involved. Instead we propose the simple function in Equation 76 that can nearly replicate the S-optimal warping parameter, while giving the user better control over the shape of S curve. 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 $\gamma \in [\gamma_b, \gamma_c]$. We choose $\gamma_b = \theta$ because this gives us precise control over η at the point where the error functions begin to change rapidly. Based on observations of the optimal S curves over the typical range of θ and f_e/n_e, we choose $\gamma_a = \theta/3$ for $\mathbf{F}_p$ and

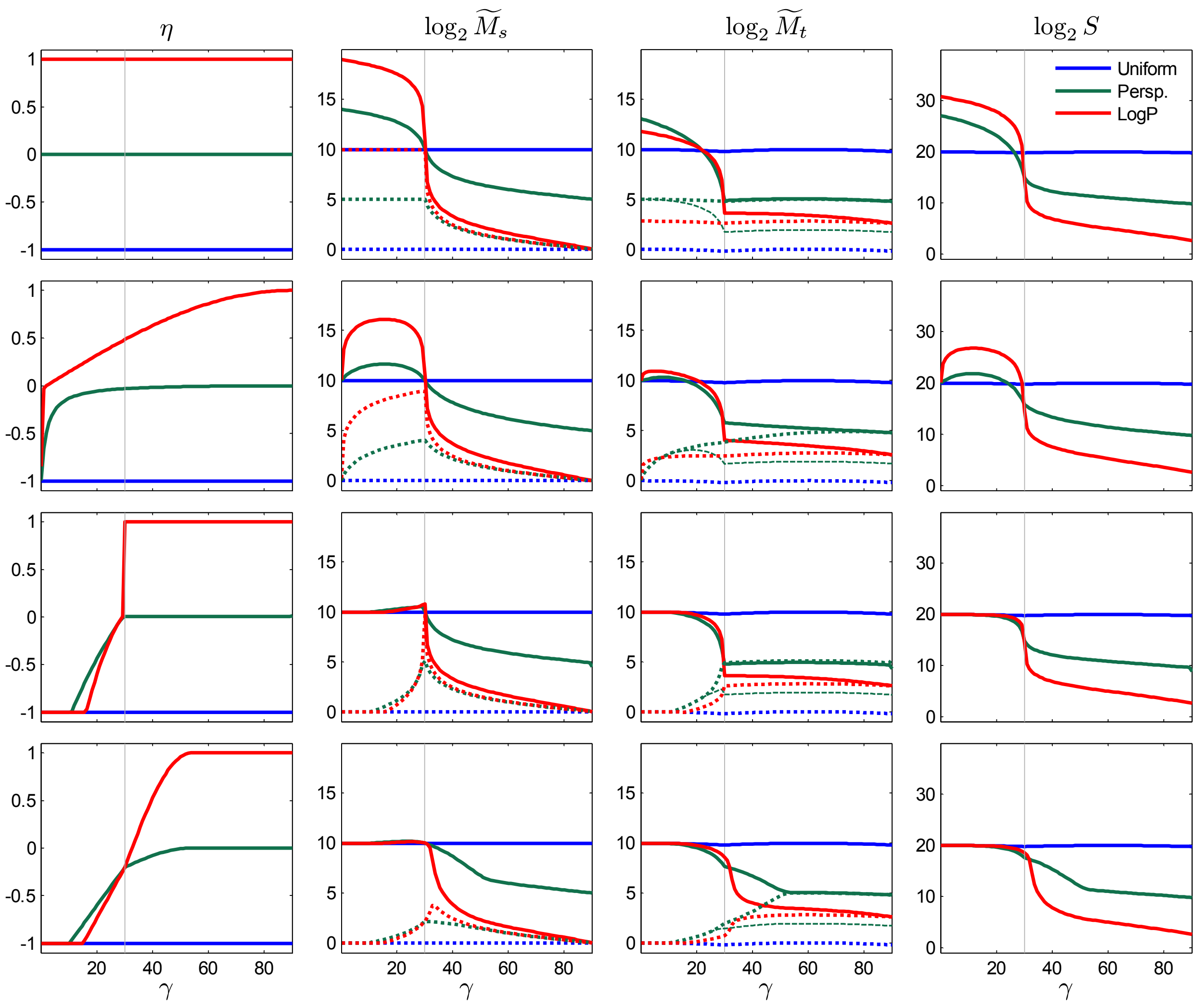


Fig. 31: ***Various shaping functions for the warping parameter.*** *For the $\widetilde{M}$ graphs, the solid lines are $v = v_n$ and dotted lines are $v = 1$. The dashed line is $v = v_{\min}$ for $\widetilde{M}_{p,t}$. (First row) The optimal warping parameter computed for $\gamma = 90^\circ$ is used for all γ without modification by a shaping function. The error is extremely high for $\gamma < \theta$. (Second row) The LiSPSM $1/\sin\gamma$ is used to ramp off n' (n' is converted to η in this graph). (Third row) The warping parameter 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 parameter but the other two parameterizations converge to uniform as $\eta \to -1$. The frustum parameters are $n = 1$, $f = 1024$, and $\theta = 30^\circ$.*

$\gamma_a = \theta/2$ for $\mathbf{F}_{lp}$. γ_c controls how quickly the transition from low to high error occurs as $\gamma \to 0$. We choose $\gamma_c = \theta + 0.3(90 - \theta)$ so that the error is kept low for $\gamma > \theta$ while extending the transition period. η_c should be 1 for optimal S_{lp} and 0 for optimal S_p. η_b should be 0 to replicate the behavior of the S-optimal curve. However, this causes the value of $\widetilde{M}_{p,s}$ at $v = 1$ to rise to the same value as at $v = v_{n1}$. By decreasing η_b a small amount, it is possible to mitigate this effect without affecting S too much. By experimentation we arrive at the value of $\eta_b = -0.2$. The behavior of S when using these parameters to choose the warping parameter is fairly consistent over varying θ and f_e/n_e. We have observed that for some light positions not in the yz plane, the warping can be slightly stronger than necessary.

Our shaping function is based on the optimal parameter using S as an 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 smooth splines. By presenting users with graphs similar to those shown in this paper, they could interactively generate a shaping function that meets their particular needs.