Real-Time Perspective Optimal Shadow Maps

A Thesis presented

by

Hamilton Yu-Ik Chong

to

Computer Science

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April 8, 2003

Abstract

Shadow mapping is perhaps the most pervasive method for creating convincing shadows in computer graphics. However, because it is an image-space algorithm it must contend with aliasing issues caused by finite and discrete sampling.

This exposition differs from previous work on shadow maps in that it pinpoints the one remaining degree of freedom in the shadow mapping process – the angle of the film plane at the light – and uses this understanding to develop a framework for categorizing the effects of film plane angle change on the sampling pattern in pixel space, allowing a quantitative analysis of how to exploit this degree of freedom in mitigating the aliasing problem. This presentation focuses on making a thorough study of the implications for the flatland, or two-dimensional world, case. In the 2D world, our algorithm can always find the optimal setting of the light plane, consistently providing factors of 2-3 improvement over previous methods.

Because no *a priori* assumptions can be made about a dynamically changing scene in a real-time context, an initial low-resolution rendering pass is required to allow informed decisions to be made. The rest of the algorithm can be broken up into two parts:

1. **deciding the optimal angle for a single light frustum** – given a per-pixel measure of sampling "badness" for some initial setting of the light's film plane, we can parameterize by angular offset the sampling error that would be incurred through use of a different film plane orientation. This allows us to solve for the optimal angle using standard calculus techniques.

2. **splitting the light frustum** – we greedily partition the light hull into smaller light frusta and allocate resolutional resources so as to minimize the sum total of the error metric over the multiple frusta.

Another benefit of understanding the degree of freedom as light plane angle is that shadow map rendering can be computed as "normal" without having to deal with the complexities of first moving into a different projective space as suggested by some previous work.

Contents

List of Figures

Chapter 1

Introduction

"To think of shadows is a serious thing."
– Victor Hugo

Shadows play essential roles in human perception. They provide visual cues for identifying shape and allow viewers to extract spatial understanding from two-dimensional images. An image lacking proper shadowing appears artificial because it presents lighting information that conflicts with the natural intuitions we develop from real world experiences. As a consequence, a number of algorithmic approaches for drawing shadows have been developed to address this aspect of realistic image creation.

In the context of real-time rendering (i.e., polygon rasterization), the difficulty lies in the fact that triangles are treated independently and therefore lack more global information about their surrounding environment. A couple techniques have been developed to get around this limited knowledge possessed by each triangle. The presentation of shadowing techniques here assumes a familiarity with the basics of image rendering and texture mapping. For a review, Appendix A provides all the necessary preliminaries.

1.1 Projected Planar Shadows

This approach assumes all shadows will be cast onto a flat floor [1]. Each polygon of a shadow casting object gets its vertices projected onto the plane representing the floor. This defines a polygon on the floor that can be rendered in black or a subtractive color to create the effect of having a shadow on the ground. This method does not scale well for non-uniformly flat shadow receivers because more

tricky intersection tests must take place and possible subdivision of the projected shadow triangle may be required to ensure the shadow is always drawn on the surface top.

1.2 Shadow Volumes

One method that has gained popularity in the real-time rendering contexts is the so-called Shadow Volume technique [1, 2]. Figure 1.1 illustrates the algorithmic process. In this approach, the silhouette of an object is determined from the perspective of the light. A polyhedral volume is then constructed connecting the light source to the silhouette and the volume is extruded to infinity (or far enough to encompass all geometry of potential interest). The geometry in the scene is rendered a first time as normal without regard for shadowing information. The constructed shadow volume is then rendered from the perspective of the viewer's camera, giving back-facing polygons an opposite sign from the front-facing ones (the sign is added into a buffer called the *stencil buffer*). For each pixel, the signs of front-facing and back-facing polygons are added into the stencil buffer only if the polygon depth at that pixel is closer than the value stored in the depth buffer created from the initial rendering pass. Objects in the scene living inside the shadow volume will be behind the front-facing shadow volume polygons, but in front of the back-facing ones. Therefore, the stencil buffer will have a nonzero value for all the pixels making up that object since front-facing polygons will pass the depth test and back-facing ones will not. For pixels corresponding to geometry behind the shadow volume, both front-facing and back-facing polygons will be rendered and the signs will cancel. For pixels corresponding to geometry in front of the shadow volume, neither front-facing nor back-facing polygons will be drawn, so the stencil buffer will remain its initial zero at those locations. A third rendering pass is then made in which a subtractive color is applied to all those pixels whose stencil buffer counterparts are nonzero.

1.2.1 Benefits of Shadow volumes

Because shadow volumes take advantage of geometric knowledge during rendering, they allow for per-pixel accuracy during image creation. Current hardware also provides stencil buffers and other features making them quite practical for real-time rendering.

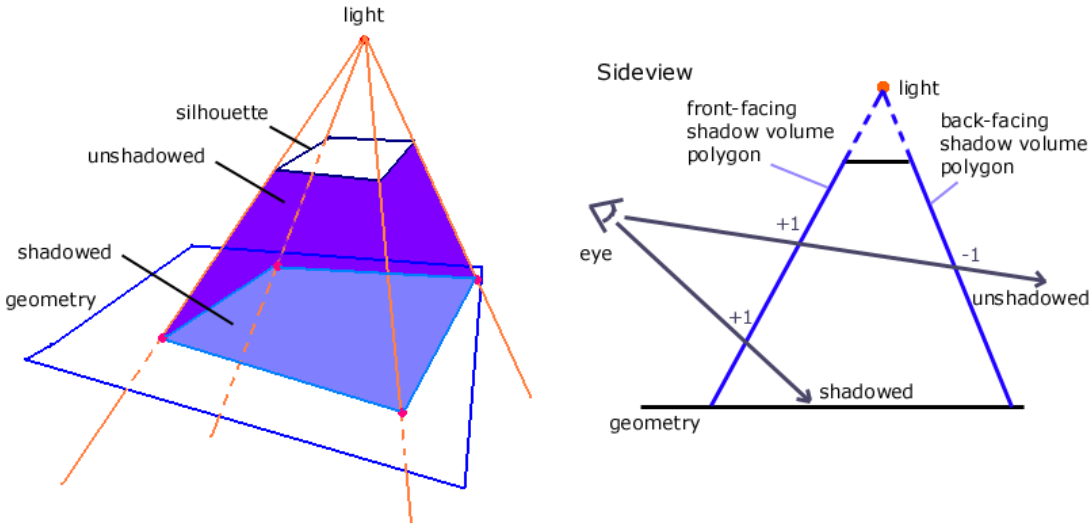


Figure 1.1: *Shadow Volume*
The region in which the viewer sees only the front-facing shadow volume polygons is shadowed; the region in which both front-facing and back-facing shadow volume polygons are visible corresponds to unshadowed scenery

1.2.2 Drawbacks of Shadow Volumes

The requirement that shadow volumes have access to geometric information is limiting. While it allows for pixel level accuracy, it restricts the allowable geometry representations for real-time applications. Shadow volumes mandate efficient access to geometric connectivity or else demand two dot product tests per edge to determine silhouettes as seen from the light. Such demands complicate data structures and tend not to scale well with increased scene complexity. If the scene possesses occluders with intricate silhouettes, shadow volume rendering can easily reach the fill rate limitations of the hardware. The rendering passes over shadow volume extrusions also require "watertight" precision so that rounding errors do not cause any pixels on shadow volume edges to be missed or drawn multiply. Such errors can cause missing or unexpected spots of shadowing to appear in the scene.

1.3 Shadow Maps

In shadow mapping, a picture (more precisely, a depth map) of the scene is first captured from the viewpoint of the light. Subsequently, when rendering the image

from the perspective of the viewer's camera, depth values can be mapped to the light's domain to compare whether such points are lit or are in shadow (i.e., whether those depth values correspond to ones closest to the light or not) [1, 4, 7, 9, 10]. This depth test gives a binary result (shadowed or unshadowed) for each pixel. For convenience, each depth texture sample will henceforth be referred to as a *lexel*[1]. Fortunately, the mapping back per pixel to the lexel domain doesn't have to be as costly as sending out shadow feelers. Given the mapping to the lexel domain of a polygon's vertices, the image of any point in the convex hull can be found through interpolation.

1.3.1 Benefits of Shadow Maps

Shadow mapping is flexible and relatively fast. Being a purely image based approach, it requires no extra geometry connectivity information and can be used in conjunction with any existing algorithm for rendering.

1.3.2 Drawbacks of Shadow Maps

Because finite and discrete sampling occurs in the lexel domain, reconstruction at the camera's viewpoint may exhibit aliasing due to undersampling and/or point sampling. Pictorially, these artifacts appear as blocky shadow edges or distinctly jagged edges between shadowed and unshadowed regions. Aliasing can also result from other factors such as finite depth precision, but those topics will not be addressed here.

1.3.3 Depth Bias Issues

Figure 1.2 demonstrates one of the problems produced by discrete sampling. Because the depth values of the world are discretely sampled from the viewpoint of the light, a single lexel depth value is used for an entire slice of the world. That single value is unable to capture the possibly varying geometric depths across the rays spanned by the lexel. If a view camera pixel center intersects the world slice in a slightly different direction than that sampled by the light, the depth value compared against for shadow determination may be incorrect. This depth value offset can cause shadow "acne" because a surface that should be lit self-shadows itself. An additive depth bias is introduced to counter this problem. Determining the appropriate bias to use can be tricky and so a bias that is too large (causing the shadow to shift from its

[1] shorthand for light texture element

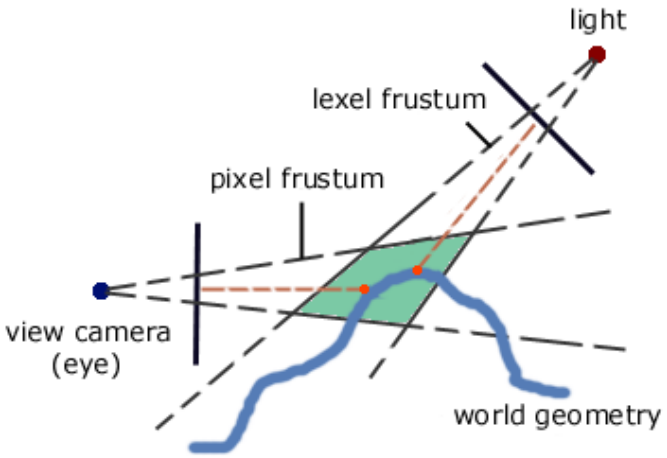


Figure 1.2: *Depth Bias*
Because a single depth is given to a whole lexel frustum, shadow acne can result from a mismatch in the point sampled in the lexel and pixel frustum intersection. Here, although the pixel should determine that the geometry is unshadowed, the point sampled is farther from the light than the lexel sample; therefore it will instead be reported as shadowed.

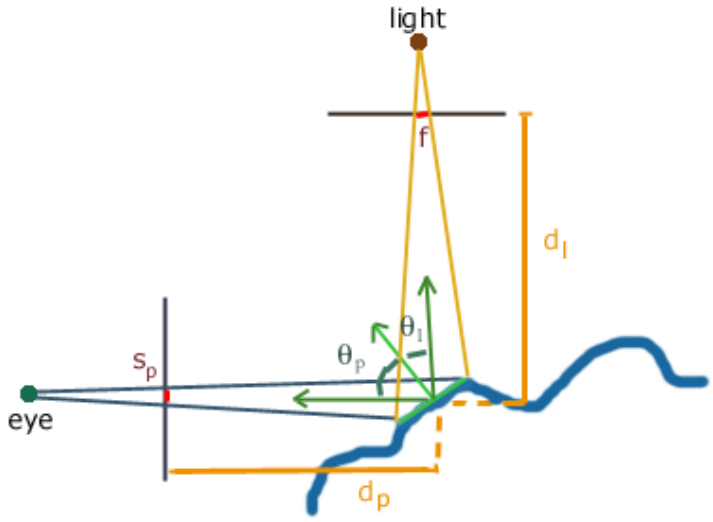


Figure 1.3: *Projecting pixels as lexel domain footprints*

real position) is usually preferred to one that is too small (which results in acne) [1].

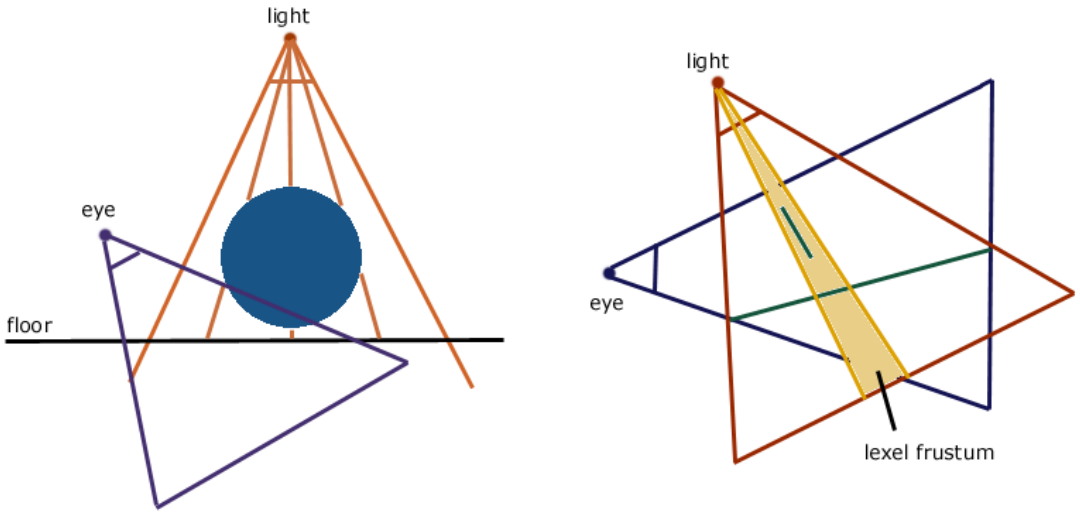
1.3.4 Sources of Aliasing

For simplicity, we will describe the aliasing issues in *flatland*, or the two-dimensional world. Given some pixel size and some lexel size (both measured with consistent units in world space), we would like to quantify the number of lexels covered by the image, or *footprint*, of the pixel when mapped to the lexel domain. The relation is approximately given by [9]:

$$f = s_p \frac{d_p}{d_l} \frac{\cos\theta_l}{\cos\theta_p}$$

where f is the size of the pixel footprint in the lexel domain, s_p is the size of a pixel, d_p is the distance from the near view plane to the world geometry, d_l is the distance from the geometry to the lexel plane, and θ_p and θ_l are the angles between the geometry normal and vector to the pixel and lexel planes respectively.

When $s_p d_p$ is much smaller than d_l, f is small, meaning that many neighboring pixel footprints will fit within one lexel. In that case, numerous pixels depend on the same lexel sample to make shadowing decisions. This situation leads to a blocky shadow. Intuitively, $s_p d_p$ is small relative to d_l when the view camera is moved in close to the geometry and therefore the viewer sees the finite resolution of the shadow map. This is called *perspective aliasing*.



(a) *(b)*

Figure 1.4: *Aliasing problems to address*
(a) *perspective aliasing – one lexel covers a large portion in the view camera's image space* (b) *projective aliasing – a large polygon is covered by a single lexel depth value*

If $s_p d_p$ is large compared to d_l, then f is large and covers multiple lexels in the shadow map. Supposing the closest lexel to the footprint center is used as the light depth sample, then slight changes in the view camera can lead to inversions of the binary shadow decision if the footprint covers lexel depths for which the shadow test both passes and fails. This leads to the viewer seeing jagged shadow edges which seem to "swim" with animated movement of the camera. The problem here is point sampling the lexel depth map undersamples the available information for shadow determination. Ideally, all the information from the entire footprint would be considered in determining the final pixel shadow properties.

If $\cos\theta_l$ is small relative to $\cos\theta_p$ then f is again small and one lexel sample covers a number of pixels. This scenario occurs when the light rays happen to be near parallel to the geometry surface and therefore stretch over a significant range of depths. This is referred to as *projection aliasing*. The case in which $\cos\theta_l$ is large relative to $\cos\theta_p$ requires consideration of the entire footprint to reduce aliasing.

1.3.5 Extensions of the Shadow Map Algorithm

Percentage Closer Fitting addresses the problem in which a pixel in the image maps to a region covering multiple lexels in the light depth buffer [7, 10].

In such cases, point sampling (e.g., simply choosing the closest point to the mapped image of the pixel center) produces unsatisfactory results because slight changes in the view camera position or orientation can switch the binary shadow decision to its inverse. Reeves, Salesin, and Cook, who presented the work noted that standard filtering of the depth map values will not solve the problem because interpolated depth values may yield a depth for which no actual geometry exists [7]. Instead, filtering must be performed on the results of the shadow tests in the footprint. Shadowing can then be determined to be proportional to the percentage of shadow tests that claim the point is in shadow. In some cases, the pixel footprint may cover such a large region in the lexel domain that Monte Carlo or other sampling methods must be employed to speed up the filtering process. Although percentage closer filtering remedies the problem of undersampling the lexel samples when deciding upon a pixel color, it does not solve aliasing due to undersampling of the world geometry from the light's point of view.

Adaptive Shadow Maps tackle the issue of undersampling from the perspective of the light. The approach creates a hierarchical depth map for the lexel domain [4]. It performs multiple passes to determine which portions require greater refinement and tunes the local resolution appropriately. Traversal of this hierarchical data for updates prevents this technique from being realized in current hardware. A dynamically changing scene would require the hierarchical updates each frame and therefore prevents this technique from being employed in real-time rendering.

Perspective Shadow Maps attack the undersampling problem by taking a snapshot of the scene in a different space [9]. Instead of creating a depth map for the scene in world space, it performs standard shadow mapping in the view camera's post-projective space (the world as the viewer sees it). The viewer's projective transform effectively maps the view frustum into the canonical cube $[-1,1]^3$ so that the viewer's image is created by a simple orthographic projection of the scene onto the near plane of the cube. This means that linearly spaced samples in the viewer's image (near) plane translate into samples that are linearly spaced on planes in the cube. The hope is that the viewer samples mapped onto the geometry in the cube are more uniform when seen from the viewpoint of the light post-projective transform. Therefore the uniform grid sampling at the light will better accommodate the requirements for reconstruction during image creation.

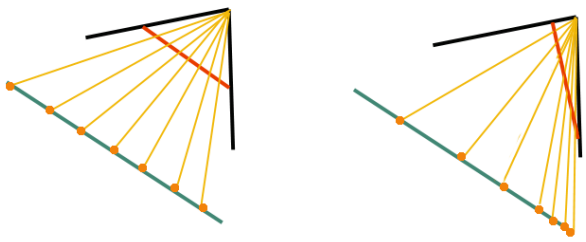


Figure 1.5: *Lexel plane angle affects sampling*
Uniform spacing on lexel plane corresponds to non-uniform depth sampling on plane in the world; the linear rational relationship is determined by the angle of the lexel plane

1.4 Our Approach

The approach we take builds off previous work done on standard and perspective shadow maps [1, 4, 7, 9, 10]. Although Stamminger and Drettakis make the interesting observation that shadow maps are more efficient in their use of lexel samples post-projective transformation, they do not explicitly identify the extra degree of freedom that allows their approach to mitigate shadow mapping's largest problem [9]. Consequently, they can only present a heuristic method for improving the quality of the generated shadows.

Note that (as shown by the decomposition of the projective mapping in section A.2) once the basis of the light's frame is chosen, all other projection mapping parameters are fixed. Therefore, the only remaining degree of freedom in the shadow mapping algorithm affecting sampling pattern is the angle of the light's film plane, represented as a rotation of the light's axes. It is this degree of freedom that is implicitly being exploited by the perspective shadow maps algorithm. In this presentation, we develop the framework for understanding shadow quality in terms of this degree of freedom and from there put forth quantitative arguments for an optimal orientation of the lexel plane. Recognizing the degree of freedom as lexel plane angle also has the additional benefit that sampling optimization need not take place in a different projective space, thereby eliminating the need to handle corner cases such as the light hull straddling the infinity plane.

Figure 1.5 illustrates how changing the lexel plane angle can affect the sampling of geometry depths within the world. Note that uniformly spaced intervals on the lexel plane correspond to non-uniform sampling in the world. Ideally, the sampling pattern in the world induced by the light's lexel plane will match the sampling pattern created if the viewer casts the pixel grid onto the world geometry. Intuitively, when choosing the lexel plane angle, it is advantageous to orient the lexel plane so that it points toward the densest region of pixel samples in the world.

We start our analysis by developing some metrics to quantify how closely lexel sampling maps to the desired sampling pattern in pixel space. Given such optimality criteria, we then delve into theoretical analyses of various cases, identifying in particular the setups for which ideal shadow quality is achievable.

Finally, we present an algorithm for the general flatland configuration. This algorithm proceeds in two parts:

1. **deciding the optimal angle for a single light frustum** – we seek an angle of the lexel plane that will minimize the developed metrics and thereby reduce perspective aliasing. Using an initial rendering pass, we can parameterize the error measures by angular offset. This is done by explicitly writing down a relation that re-maps lexels to rotated lexels. We can then solve for the optimal angle offset using standard calculus techniques.

2. **splitting the light frustum** – we greedily partition the light hull of interest into smaller light frusta, each with its own shadow map, and allocate resolutional resources so as to minimize the sum total of the error metric over the multiple frusta. The splitting of the light frustum allows the algorithm to isolate regions requiring local lexel plane or lexel resolution change. This facilitates further reductions in perspective aliasing but also addresses projection aliasing.

Chapter 2

Defining Optimality and Analyzing Cases

To establish the framework for quantifying the effectiveness of various shadow mapping algorithms, we now turn to formalizing the intuitions presented earlier. From this vantage point we will be able to deduce the optimality of certain shadow mapping setups and faultiness of others.

2.1 Metrics

Here we make concrete the idea of optimality. We start from the intuitive idea that what is really sought is that the sampling done in the lexel domain match the sampling rate required in pixel space. We desire this rate match over the entire pixel space of interest, Ω.

Ideally, for shadow mapping purposes, Ω would be the set of pixels corresponding (in world space) to the border between geometry mutually visible by the light and view camera (i.e., those portions of the world that see no shadow) and geometry seen only by the viewer (i.e. those in shadow). Given sufficient sampling over such a set, we are guaranteed that shadow determination can proceed without undersampling artifacts. However, knowledge of such a set would mean this shadowing problem is already solved. Therefore, Ω is instead chosen to be the set of pixels corresponding to world points visible to the view camera. These are the points for which we care (without further knowledge) about shadowing determination. Ω could be extended to include pixels for which no geometry is seen. However, limiting the size of Ω can only help reduce the aliasing problem because no lexels need be devoted to irrelevant regions of the scene.

The pixel space may be treated both as discrete or continuous. In the discrete setting, Ω is the set of pixels for which geometry is seen and hence for which lexel data is available. In the continuous context, Ω represents the union of pixel intervals for which geometry is seen.

2.1.1 Developing Appropriate Measures

Because our analysis will focus on the flatland implications of lexel angle choice, our metrics will be developed over one-dimensional lexel and pixel spaces. For each pixel, the quantity of interest will be $\frac{dp}{dl}$, where p is in units of pixels and l is in units of lexels. This derivative (evaluated at each pixel location) is a measure of the "badness" at that pixel. A large value means that moving one unit step over to the next lexel sample will incur a large jump in the pixel value. That translates to the lexel sample covering a large portion in the pixel space. A small value for $\frac{dp}{dl}$ implies that a couple lexel samples correspond to a single pixel and therefore supersampling of depth tests can be performed at the corresponding pixel. This measure of badness suggests a couple natural choices for metrics. A discrete and continuous version of each is presented here.

1. The L_1 metric[1].

$$\sum_{\mathbf{p}\in\Omega} \left|\frac{dp}{dl}\right| \quad \text{or} \quad \int_{\mathbf{p}\in\Omega} \left|\frac{dp}{dl}\right| dp$$

   Since changing the lexel plane orientation does not affect whether pixels see geometry, $|\Omega|$ is constant. This means that minimizing this measure of badness for the entire image is equivalent to finding the lexel plane orientation that minimizes the average pixel badness $\frac{dp}{dl}$.

2. The L_2 metric.

$$\sum_{\mathbf{p}\in\Omega} \left(\frac{dp}{dl}\right)^2 \quad \text{or} \quad \int_{\mathbf{p}\in\Omega} \left(\frac{dp}{dl}\right)^2 dp$$

   As before, $|\Omega|$ is constant with regard to changes in lexel plane orientation, so minimizing this measure can be interpreted as minimizing the second moment of $\frac{dp}{dl}$. Intuitively, this measure captures some notion of concentration around the mean. Therefore, under this metric shadow quality is less likely to deviate from the average quality. This average may or may not be worse than the average achieved by use of the L_1 metric.

[1] In the continuous version, we can meaningfully define an integral because we can simply remove from Ω the finite number of points $\mathbf{p}$ (measure zero set) such that the quantity is discontinuous. We know there are only a finite number because there are only a finite number of polygons and within each polygon $\frac{dp}{dl}$ is continuous and has the same sign

3. The L_∞ metric.

$$\max_{\mathbf{p}\in\Omega} \left| \frac{dp}{dl} \right|$$

Minimizing this measure means that we seek the lexel plane angle that ensures the worst pixel badness is as small as possible. This will have the tendency of pushing the $\frac{dp}{dl}$ toward being constant.

The choice of the metric may very well depend on the application and the desired shadow quality guarantee required. It may be that the worst shadow quality pixels are the most egregious and jarring. Therefore the L_∞ measure may be desirable. However, improving the worst pixels may come at the price of sacrificing the overall quality of the other pixels. Ensuring that most pixels have good shadow quality may prove more important than making the couple worst pixels better. Therefore the L_1 and L_2 measures may arguably be better choices in certain settings.

For the algorithm we present, the L_1 and L_2 measures will be used instead of the L_∞ option because they are differentiable with respect to lexel plane angle (well, the absolute value in the L_1 measure will be able to be removed without ill-effects for our purpose; it is the max operator that causes problems with differentiability in the L_∞ version).

2.1.2 Continuous versus Discrete Pixel Domain

We first turn to analyzing the behavior of the continuous metrics. The continuous variants are easier to work with because they are resolution independent whereas the discrete measures do depend on the pixel grid resolution.

Interestingly, the continuous metrics will all choose (if possible) the lexel plane angle such that $\frac{dp}{dl}$ is constant everywhere. For most scenes of interest this will not be possible however, so the continuous metrics will make tradeoffs according to the intuitions described above (choosing non-uniformity in the $\frac{dp}{dl}$ so as to minimize the mean or variance).

That a constant $\frac{dp}{dl}$ is in fact optimal for any given setup can be seen as follows. A basic result from variational calculus is that given:

$$\int f(x)dx = k$$

for some constant k, then:

$$\min_f \int f^n(x)dx$$

for integer $n \geq 2$ is given by choosing f such that it is constant everywhere. Let us now interpret this result in the context of our badness measures.

$$\int_{\mathbf{p}\in\Omega} dp = k$$

for some constant k which is just the size of Ω.

$$\begin{aligned}\int_{p\in\Omega} dp &= \int_{p\in\Omega} \frac{dp}{dl}\frac{dl}{dp}dp \\ &= \int_{l\in\Omega'} \frac{dp}{dl}dl \\ &= k\end{aligned}$$

where Ω' is the set of l corresponding to each $p \in \Omega$. Now we look at manipulating the L_1 metric:

$$\begin{aligned}\int_{p\in\Omega} \frac{dp}{dl}dp &= \int_{p\in\Omega} \frac{dp}{dl}\left(\frac{dp}{dl}\frac{dl}{dp}\right)dp \\ &= \int_{l\in\Omega'} \left(\frac{dp}{dl}\right)^2 dl\end{aligned}$$

Likewise, for the L_2 norm we have:

$$\begin{aligned}\int_{p\in\Omega} \left(\frac{dp}{dl}\right)^2 dp &= \int_{p\in\Omega} \left(\frac{dp}{dl}\right)^2 \left(\frac{dp}{dl}\frac{dl}{dp}\right)dp \\ &= \int_{l\in\Omega'} \left(\frac{dp}{dl}\right)^3 dl\end{aligned}$$

These imply that the optimal angle is that which makes $\frac{dp}{dl}$ constant.

In actual implementation, the pixel space is discrete and the measures are therefore resolution dependent. The continuous viewpoint is instructive nonetheless because high resolution renderings will tend to have extrema that behave similarly. In the limit that resolutions approach infinity the discrete linearizations look like their continuous analogues.

2.1.3 Computing the Measures Efficiently

Polygon rasterization is efficient because each pixel's relevant information can be computed through interpolation. Here, we likewise seek a method for computing the required $\frac{dp}{dl}$ quantities at each pixel through interpolation or through application of

a simple function of interpolants. This will allow us to store each derivative per pixel into a buffer that can be read back at some later time. Let us order the coordinates for 2D such that $[1,\ 0,\ 0]^t$ is the vector representing one unit step in pixel space. Then:

$$V_t P_t M_t^{-1} M_s P_s^{-1} V_s^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{d(lq)}{dp} \\ - \\ \frac{dq}{dp} \end{bmatrix}$$

$\frac{d(lq)}{dp}$ and $\frac{dq}{dp}$ are constant across a raster span. The product rule for derivatives gives us:

$$\frac{d(lq)}{dp} = \frac{dl}{dp} q + l \frac{dq}{dp}$$

$$\Rightarrow \frac{dl}{dp} = \frac{\frac{d(lq)}{dp} - l\frac{dq}{dp}}{q}$$

Note that l and q are both efficiently computable by linear rational or linear interpolation. Therefore, $\frac{dl}{dp}$ is efficiently computable at each pixel. $\frac{dp}{dl}$ is just the inverse.

2.2 Theoretical Analysis of Special Cases

Now that metrics have been put forth for quantitatively measuring the "badness" incurred by various shadow mapping setups, we can reason theoretically about the optimality of certain configurations or inevitable problems inherent in others.

2.2.1 Single Plane Geometry Case

Suppose now that our world geometry consists only of a single plane on which we really care about shadow quality (perhaps most shadows fall upon a flat floor). Then we can achieve an optimal configuration for the shadow map sampling. Figure 2.1 shows how the angle is chosen. In 2D, the intersection point between the world plane (line) and viewer plane is found. Then the lexel plane is chosen so as to be parallel to the line connecting this intersection point and the light.

Theorem 1 *Orienting the lexel plane in this way provides a constant* $\frac{dp}{dl}$.

Proof: Let us denote the intersection between the world plane and the view plane as **q**. Consistent with the notation presented in section A.2, we will denote the

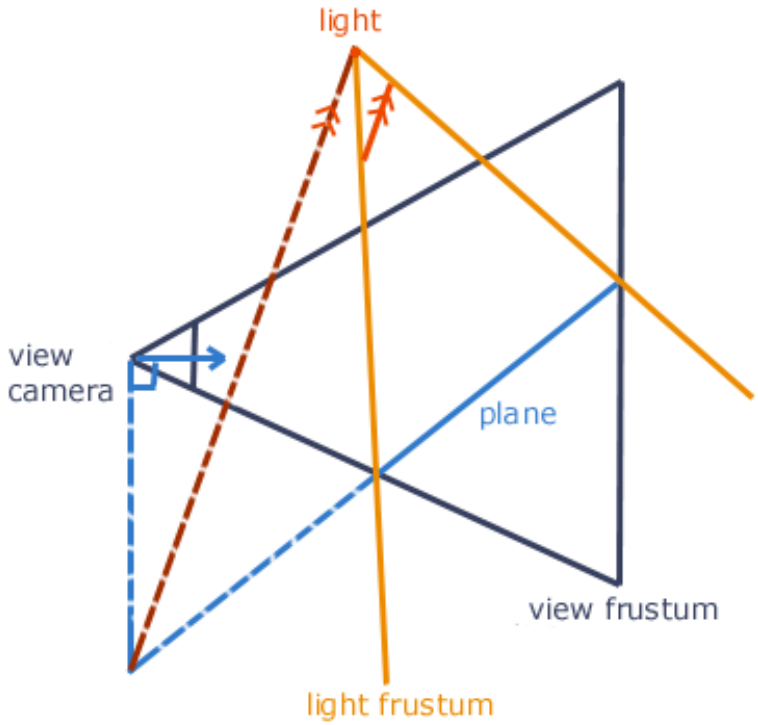


Figure 2.1: *Choosing the optimal angle for single plane geometry*

matrices associated with the projection of the world onto the shadow map texture with the subscript t. Those associated with the projection of the world onto the pixel screen will have the subscript s. $V_t P_t M_t^{-1}$ is the mapping such that after division by the *homogeneous coordinate* w_t, world space points are mapped into the light's lexel space (recall the decomposition of the projective mapping described in section A.2). Likewise, $V_s P_s M_s^{-1}$ followed by division with w_s maps world space points into the view camera's pixel space. Both these transformations, being projective transforms, map lines to lines [3]. Let U be the line that is the image of the world plane in the light's post-projective space. We seek to show: $P_s M_s^{-1}(P_t M_t^{-1})^{-1}$ restricted to U fixes the homogeneous coordinate and therefore represents an affine mapping from U in the light's post-projective space to the view camera's post-projective space. We have dropped the V_s and V_t because these represent affine transforms. Therefore, proving $P_s M_s^{-1}(P_t M_t^{-1})^{-1}$ affine implies movements in lexel space correspond linearly to movements in pixel space. Consequently, $\frac{dp}{dl}$ would be constant when we restrict our attention to the single world plane.

To show $P_s M_s^{-1}(P_t M_t^{-1})^{-1}$ fixes the homogeneous coordinate when restricted to U, we concentrate our attention on the line's point at infinity. Let $\mathbf{q}_t$ be the image of $\mathbf{q}$ under the transform $P_t M_t^{-1}$. Let $\mathbf{q}_s$ be the image of $\mathbf{q}$ under the transform $P_s M_s^{-1}$. Note that by construction, $\mathbf{q}$ lies on both the view plane of the light and

view plane of the camera. Therefore, since $P_t M_t^{-1}$ and $P_s M_s^{-1}$ pull the light and view planes respectively to infinity, the homogeneous coordinates of $\mathbf{q}_t$ and $\mathbf{q}_s$ must both be zero.

The mapping $P_s M_s^{-1}(P_t M_t^{-1})^{-1}$ restricted to U can be represented by a 3x3 real matrix because the unrestricted projective mapping can be. Let $[a, b, c]$ be the last row of this matrix. Since U is a line, we can write its point at infinity as $[\lambda, m\lambda, 0]^t$ or $[0, \lambda, 0]$ for nonzero λ where m is the slope of the line. The second case can be treated as the first in which m is zero by symmetry under exchange of the first two coordinates. Therefore, without loss of generality, we will consider only the first formulation of U as a line. To ensure that the transform does indeed map $\mathbf{q}_t$, a point at infinity, to a point at infinity, we require of [a,b,c]:

$$[a\ b\ c] \begin{bmatrix} \lambda \\ m\lambda \\ 0 \end{bmatrix} = 0$$

This implies

$$\lambda(a + bm) = 0$$

Therefore, any point $[\lambda, m\lambda + k, 1]$ on the line U (where m and k are constant) will map to a point with homogeneous coordinate given by:

$$\begin{aligned} [a\ b\ c] \begin{bmatrix} \lambda \\ m\lambda + k \\ 1 \end{bmatrix} &= \lambda(a + bm) + bk + c \\ &= bk + c \end{aligned}$$

The value $bk + c$ is constant, so the matrix can be normalized so that $bk + c$ is one and no division by the homogeneous coordinate is necessary to get post-projective space coordinates. ■

Corollary 2 *Choosing the lexel plane angle as such is optimal*[2]*.*

Proof: As demonstrated in section 2.1.2, the angle that makes $\frac{dp}{dl}$ constant everywhere minimizes the metric's integral. It is therefore the best choice according to the metric. ■

2.2.2 Miner's Headlamp

It has been commonly observed that for the case in which the light is positioned directly above the view camera (as a miner's lamp would be), close to optimal

[2]optimal with respect to the continuous metrics

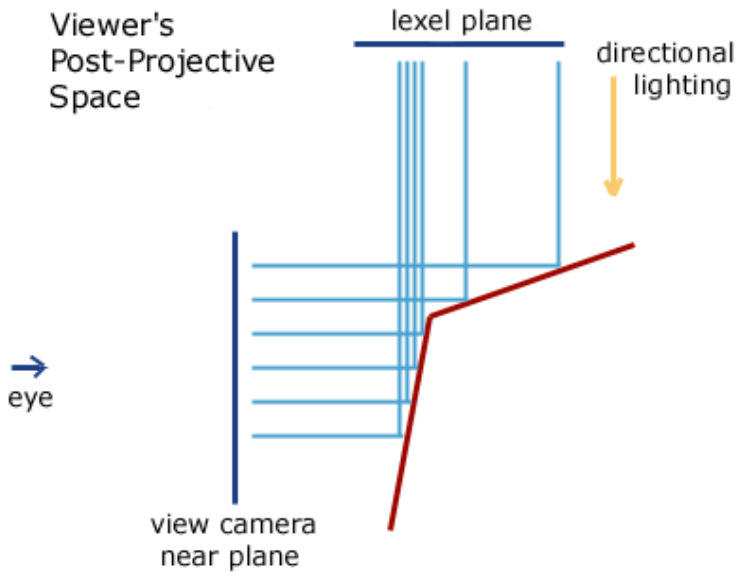


Figure 2.2: *Directional light and non-planar geometry in the viewer's post-projective space.*
No angling of the lexel plane in post-projective space can accommodate the discontinuous sampling density in the lexel domain induced by the uniform pixel samples.

sampling is achieved when the lexel plane is set parallel to the viewer's near plane. If scene geometry consists of only a single plane, then theorem 1 tells us a lexel plane parallel to the view plane is exactly optimal. In fact, this optimal angle holds for any position of the light on the view plane. At this setting the badness measure $\frac{dp}{dl}$ at each pixel is constant. Portions of the scene that are sampled sparsely from the viewer's perspective are likewise sparsely sampled from the light's perspective and regions densely sampled by the viewer are densely sampled by the light in exact proportion.

If the scene is more complex it turns out that $\frac{dp}{dl}$ will be constant across polygons. However, the value will differ for polygons of different orientations. Therefore, it is likely non-uniformity will allow for more optimal sampling by devoting more samples toward those planes that correspond to pixels that otherwise would have large $\frac{dp}{dl}$.

Figure 2.2 paints an intuitive picture of why the sampling differs for different planes. The example setup is drawn in the view camera's post-projective space. In this space, the view plane is pulled to infinity and both lighting and pixel sampling become directional. Although the pixel samples are taken at uniformly spaced intervals, they map to lexel samples that are not uniformly spaced by some constant factor. In this scenario it would be desirable to devote more lexels to the planes

more directly facing the viewer.

Proposition 3 *For arbitrary world geometry, constant $\frac{dp}{dl}$ on each polygon can be guaranteed if and only if the light is positioned in the view plane perpendicular to the view direction. Furthermore, the light direction must be oriented parallel to the view direction.*

The proof of this proposition is provided in the appendix B.1. It is interesting to note that for arbitrary scene geometry, positioning the light in the view plane and setting the film plane parallel is the only way to guarantee constant $\frac{dp}{dl}$ over each polygon and therefore if polygons of different orientations exist within the scene, it will be impossible to get a constant $\frac{dp}{dl}$ over the entire scene (unless the light position is the same as the view camera position).

Figure 2.2 paints a rather extreme scenario in which the directional light is perpendicular (or nearly perpendicular) to the view direction. This shows pictorially how large sampling discontinuities in the lexel domain for planes of different orientations may arise. As the directional light is changed so that it approaches being parallel with the view direction, the disparities between sampling rates among polygons shrinks to zero. This corresponds to moving the light located on the view plane (starting infinitely far from the view camera) closer and closer to the view camera. At the view camera location, sampling matches exactly. Therefore near the view camera, we can get close to optimal sampling for arbitrary geometry by setting the lexel plane parallel to the view plane.

That we can get close to optimal sampling in this miner's headlamp case for arbitrary geometry may be somewhat uninteresting because this is exactly the case in which the viewer sees very little shadow. A light positioned near the view camera casts shadows behind the visible objects.

2.2.3 Post-projective Rendering as Choosing Lexel Planes

To make meaningful comparisons with previous work, we now reformulate the perspective shadow mapping work into the lexel plane angle context. We then follow with some theoretical analysis and comparisons against the known ideal setups.

Perspective shadow maps differ from the normal approach to shadow mapping only in that it calls for setting up the shadow map rendering in the view camera's post-projective space, observing that sampling with respect to the view camera should likely be more uniform.

To get a sense of how close to optimal perspective shadow maps usually get, it would be helpful to convert setup of the light frustum in post-projective space into

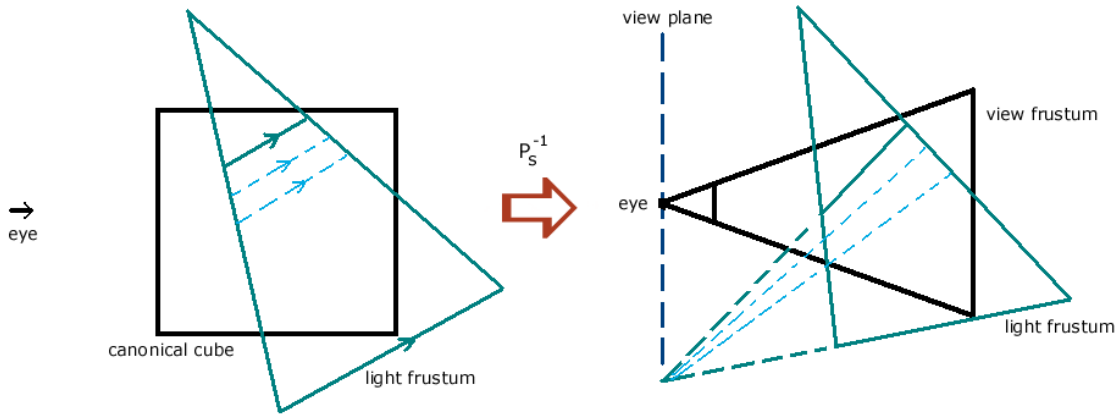


Figure 2.3: *Dependence of lexel angle on near plane in post-projective space*

a lexel plane angle in world space. Intuitively, once the light frustum is set up in post-projective space it can be mapped back to world space by an inverse-projective transform to yield some angle setting of the lexel plane. However, the heuristic of setting up a light frustum as "normal" in post-projective space is not completely well defined.

The translation to a lexel plane angle turns out to be somewhat ambiguous. For different distance offsets of the near plane in post-projective space, we get different lexel plane angles in world space. Figure 2.3 gives a flatland visualization of this phenomenon. While the near and far planes are parallel in post-projective space, they fail to remain so after the inverse-projective mapping back to world space. In post-projective space, the near and far planes, being parallel, intersect at the line at infinity. This line at infinity also happens to be the line on which the view camera is located since this is the *view camera's* post-projective space. Therefore, when mapping back to world space, this line is brought to the world's view camera plane. Consequently, the near and far light planes must also intersect on this view camera plane for any light near plane chosen.

The authors note that for perspective shadow maps, the best results are achieved when the light's near plane is pushed as far forward as possible [9]. While in standard shadow mapping, pushing up the near plane serves to increase depth precision, in the perspective setting it serves a dual purpose. As with normal shadow maps, decreasing the range of depths encountered aids precision computation. But, as noted above, choosing a different near plane also effectively changes the lexel plane

angle as seen from world space and hence changes the geometry depth sampling distributions. So the observed shadow quality improvements obtained by moving the light's near plane are a combination of these two effects. Since the range of angles spanned by the possible near planes can be quite large, the output quality from perspective shadow maps can depend greatly on how close relevant scene geometry gets to the light source in post-projective space. By recognizing the angle of the lexel plane as the adjustable degree of freedom in shadow mapping, we can avoid such dependencies and choose the angle based on the desired sampling distribution.

An additional advantage of taking the lexel plane orientation interpretation is that handling post-projective space rendering involves dealing with some special cases. For instance, when the light frustum includes objects located behind the view plane, the projective mapping yields a light frustum that straddles both sides of the infinity plane. Such complexities make for tricky implementations, so the authors suggest simply pulling the view plane back for shadow map generation until the light frustum of interest is completely in front of the view plane [9]. In the extreme when the view plane must be pulled back to infinity, this action causes perspective shadow maps to converge to the behavior of normal shadow maps. Any such required pulling back of the view plane, however, does introduce some amount of perspective aliasing. In contrast, choosing a lexel plane angle is something that can be done once and for all in world space without regard for potential wrapping around infinity due to projective mappings.

Ideal Performance

When scene geometry consists of just a single plane and the light is positioned on the view plane, perspective shadow maps get the lexel plane settings right. These are the cases in which after projective transform, the lighting is directional. This corresponds in world space to point lights on the view plane or directional light parallel to the view plane.

Because in post-projective space such lighting is directional, it does not matter what angle the lexel plane is set at. Rotation of the plane does not affect the sampling density because uniformly spaced rays stay uniformly spaced. Directional lighting in which the light rays are uniformly spaced will hit a line within the canonical cube with linearly separated samples. Likewise, because geometry is then orthographically projected onto the image plane, those linear samples stay linearly separated in pixel space. Consequently, constant sampling density is achievable.

Failure Cases

Consider now the case in which lighting is again directional in post-projective space but this time the scene possesses planes with different orientations. Figure 2.2 shows that although sampling for each plane individually is constant, between planes sampling can be quite uneven. However, no matter what angle of the film plane is chosen in post-projective space, all lexel samples will fall such that each polygon individually gets constant sampling density. However, setting the lexel plane parallel to the view plane in world space is often sub-optimal for arbitrary geometry because we really want $\frac{dp}{dl}$ to be as constant as possible over the entire scene, not each polygon separately. Intuitively, it would be advantageous to angle the lexel plane such that the $\frac{dp}{dl}$ differed minimally between world geometry with different orietations.

For point light sources that remain point light sources in post-projective space, the perspective shadow map method can give rather arbitrary results as well. Consider a scenario as in figure 2.2 (shown in post-projective space) in which the directional light is replaced with a point light above the view frustum. Perspective shadow maps generate a lexel plane in post-projective space that is close to horizontal. This translates to a lexel plane in world space that is likewise near horizontal. But here it is desirable to tilt the lexel plane such that more samples are devoted to the polygons more directly facing the viewer. This shows that while perspective shadow maps may empirically perform better than standard shadow maps in a number of test cases, the heuristic does not necessarily tend to an optimal improvement.

Chapter 3

Revising the Shadow Map Algorithm

We divide the presentation of our algorithm into two parts. The first part addresses the issue of choosing an optimal lexel plane angle for a particular given light frustum. To do so, we rewrite the metrics to minimize as a function of the lexel plane angle and solve for the minimum. Choosing a perspective optimal lexel plane angle tackles the perspective aliasing problem because it devotes more lexels to those areas more densely sampled by the view camera.

While determining an angle that best matches the sampling distribution required provides beneficial results, it is not a complete solution to the aliasing problem. There are setups for which no setting of the lexel plane angle will yield satisfactory results. In choosing an optimal lexel plane angle, an algorithm must contend with a number of competing effects. Both lost lexels and projection aliasing can greatly bias the choice of angle settled upon. *Lost lexels* refers to the situation in which part of the light frustum has no geometry of interest and therefore all lexels devoted to covering it do not contribute to the final image rendering step. An algorithm seeking to minimize the loss of lexel samples will skew the angle of the lexel plane away from such loss. *Projection aliasing* occurs when a polygon decently sampled by the view camera is oriented so that it points toward the light. In this setting, the pixel samples along that piece of geometry fit within a very small footprint in the lexel domain. To fit a sufficient number of lexel samples within that footprint, the only resort may be to greatly alter the lexel plane angle so that adequate sampling occurs on that plane to the detriment of other portions of the scene.

Lost lexels and projective aliasing are tricky to handle by lexel plane angle changes because they really require abrupt local resolutional changes to accommodate the sampling needs of the scene. The second part of our proposed algorithm

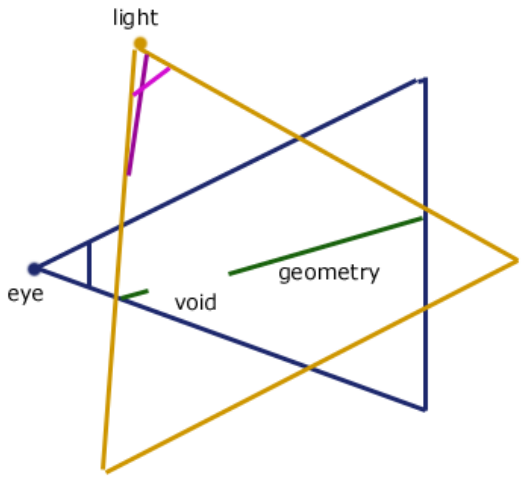


Figure 3.1: *Lost lexels – missing geometry causes lexel plane to no longer focus samples near view camera*

which partitions the light frustum into multiple smaller frusta is designed to remedy this situation. The design is meant to allow segregating lost lexel and projection aliasing regions so that such frusta may be treated separately. This gives our algorithm the mobility of making a prescribed number of local sampling changes so as to maximize the shadow quality in the end.

In both portions of the algorithm, a single initial low resolution rendering of the geometry from the perspective of the view camera is used to provide the necessary information for optimal angle and light frusta determination. Instead of storing the normal color information in the image created, the color channels are used to store lexel locations l and derivatives $\frac{dp}{dl}$ for each pixel location. These values, of course, must be relative to some initial lexel plane angle. This initial angle can be chosen arbitrarily as long as it is valid (for instance, the naive half-angle approach used in normal shadow maps that sets the light direction to bisect the angular sweep of the light frustum).

3.1 Determining the Optimal Angle

Our ultimate aim here is to re-write the metrics decided upon in section 2.1 in terms of a single angle parameter so that we can use the usual rules of calculus to find the

angle that gives the minimum sampling error. We would like to reformulate:

$$\sum_{\mathbf{p}\in\Omega}\left|\frac{dp}{dl}\right| \quad \text{or} \quad \sum_{\mathbf{p}\in\Omega}\left(\frac{dp}{dl}\right)^2$$

Suppose we have the per pixel $\frac{dp}{dl}$ generated for a particular initial setting of the lexel plane. We would like to determine the $\frac{dp}{dl'}$ that would be generated if rotation of the light had been changed to yield new lexel coordinates l' instead of l associated with each pixel. If we can determine explicitly the mapping from l to l' in terms of the rotation applied and compute $\frac{dl}{dl'}$ for each lexel l, then the chain rule for derivatives gives us:

$$\frac{dp}{dl'} = \frac{dp}{dl}\frac{dl}{dl'}$$

This is re-mapping of lexels to rotated lexels is essential to our algorithm because $\frac{dp}{dl}$ is some given value for each pixel that is constant with respect to angle changes; $\frac{dl}{dl'}$ for each pixel is a quantity depending only on the angular change. Therefore, if the re-mapping function could be written out explicitly, this would enable formulating the error metrics in terms of the single angular parameter. Fortunately in 2D, this re-mapping is easy enough to describe. The re-mapping transformation is given by:

$$P_{l'}R_x^{-1}P_l^{-1}$$

where P_l is the original projective transform that yielded the l coordinates, R_x is the rotation of the light's axes by amount x, and $P_{l'}$ is the projection matrix with the new top and bottom extents giving us the l' coordinates. Notice that once R_x is chosen, $P_{l'}$ is completely determined by the extents of the samples the light frustum needs to cover. Let θ_t and θ_b represent the angles (with respect to the original light's axes) corresponding to the top and bottom extents of the near plane. Then for some change in the lexel plane angle x, the new angles of extents are given by $\theta_t - x$ and $\theta_b - x$.

This allows us to write for some fixed $p_i \in \Omega$ and hence fixed l_i:

$$\left(\frac{dl}{dl'}\right)_i(x) = \frac{\cos\theta_t\cos\theta_b\sec(\theta_t - x)\sec(\theta_b - x)(2\cos x + \sin x\,(\tan\theta_t + l_i\tan\theta_t + \tan\theta_b - l_i\tan\theta_b))^2}{4}$$

For the L_1 norm we then seek to solve[1]

$$\frac{d}{dx}\sum_{i=1}^{|\Omega|}\left(\left.\frac{dp}{dl}\right|_{p_i}\left(\frac{dl}{dl'}\right)_i(x)\right) = 0$$

[1]We have dropped the absolute value sign since we can choose to perform rasterization such that all the derivatives point in the same direction (say positive for convenience). Therefore, since we only care about the extrema points, these solutions are left unchanged

The key observation to make here is that only the $\left(\frac{dl}{dl'}\right)_i(x)$ depend on the angular change x. The roots to this equation can then be solved for and written in closed form. Mathematica gives eight critical points, corresponding to the possible rotations yielding the desired overall minimum of the error metric. Of these eight choices, we can rule out those that yield invalid angles and compute the errors on the rest in constant time to find the real minimum.

It could be argued that perhaps all the extrema fall outside the range of valid angles of the lexel plane. In practice, we have never observed a case in which this happens. Furthermore, there is good reason to believe that for nondegenerate cases we will always find at least a valid local minimum. As the angle of the lexel plane approaches the invalid angles (i.e., as it is slanted to shoot off to infinity and therefore have infinite length), the badness measures in turn tend to infinity. Yet for some valid angle on scenes of interest (say, the angle determined by normal shadow maps), the error metric will take on a finite value. Therefore, since the error metric function is continuous, there must exist a local minimum. If no finite measure exists within the range of valid angles, then the minimum might as well be taken to be the values given at the boundary.

For the L_2 norm, we seek a solution x to:

$$\frac{d}{dx}\sum_{i=1}^{|\Omega|}\left(\left.\frac{dp}{dl}\right|_{p_i}\left(\frac{dl}{dl'}\right)_i(x)\right)^2 = 0$$

The solution to this expression is too unwieldly to write in closed form. However, we can turn to Newton's Method for finding a root. Falling back to Newton's Method may be slightly disturbing because only quadratic equations are guaranteed to not cause chaotic behavior [6]. However, the function here has experimentally proved itself relatively well behaved. Over all the experiments and various initial guesses, Newton's Method tended to converge to within one hundredth of a degree within only five or six iterations. A set of randomly generated initial guesses provides a decent set of values over which to check for valid minima. Even simply restarting only after getting convergence to an invalid angle proved to provide the desired minima in most cases[2]. For a discussion on quantifying the size of a local region in which superconvergence is guaranteed, the reader is referred to [6].

For a single lexel plane, we can find the optimal angle with the approach described in algorithm listing 1. For this algorithm, we first choose some initial setting of the lexel plane (can be any heuristic that gives a valid angle (an angle for which the projection matrix makes sense). We then render from the view camera's perspective in low resolution and save the data *depth*, l, and $\frac{dp}{dl}$.

[2]an initial guess of zero was used with some additive constant applied each successive try

Algorithm 1 angle GetOptimalAngle(*initial_angle*, { (*depth*, l, $\frac{dp}{dl}$) })

{k is an adjustable parameter for number of potentially different solutions to look for}
for $i = 0$ to k **do**
  $guess \leftarrow i \cdot$ RandomOffset()
  $result[i] \leftarrow$ invalid angle
  while $result[i]$ is invalid **do**
    $result[i] \leftarrow initial\_angle +$ NewtonSolution(guess, { (*depth*, l, $\frac{dp}{dl}$) })
    $guess \leftarrow guess +$ RandomOffset()
  end while
end for
return $result[i]$ that gives the least error

3.2 Dividing Up the Light Frustum

In this part of the algorithm, we generalize the problem setup. Now we are given a light frustum budget (a maximum number of light frusta we may use) and a lexel budget (the total number of lexels allowed for all light frusta). The question is now how to divide up the lit scene among the smaller light frusta and allocate the lexels so that we achieve the best shadow quality in the finally rendered image.

We render the scene from the perspective of the view camera in low resolution as before to get initial *depth*, l, and $\frac{dp}{dl}$ data from which to make informed decisions. We then split the light coverage into a set of smaller frusta, starting with some predefined maximum MAX_LIGHT_FRUSTA. Until we are within our light frusta budget, we greedily merge adjacent frusta such that the merging process and reallocation of lexels to each frustum yields the smallest badness increase according to our chosen metric. Additional frusta can only help so once the frustum budget is reached it is not necessary to perform any further merges. Further merges may take place only if it is judged that the marginal gain from having one more frustum is too small to justify the computation cost of generating another shadow map. The optimal angle for each individual frustum is determined as before using the lexel remapping method. It is important to note that the inputs l and $\frac{dp}{dl}$ must undergo a linear transformation to make sense in the context of the smaller frusta because the l value changes meaning. The l coordinate must be rewritten to be a coordinate with respect to the smaller frustum instead of the original single light frustum used for the initial pass. The $\frac{dp}{dl}$ value must be multiplied by the size ratio of this smaller frustum near plane to the initial single frustum near plane.

Allocation and reallocation of lexels to each frusta is decided by use of lagrange multipliers. In order to meaningfully compare the errors between frusta, we need

Algorithm 2 void SetOptimalAnglesAndLexels(void)

```
1: num_frusta = MAX_LIGHT_FRUSTA
2: initial_angle ← SetInitialAngle()
3: texture_map ← low-resolution rendering saving depth, l, dp/dl
4: for i = 0 to MAX_LIGHT_FRUSTA-1 do
5:    {evenly partition by angle the light view into MAX_LIGHT_FRUSTA frusta}
6:    tbextents[i] ← bottom extent of ith light frustum
7:    tbextents[i + 1] ← top extent of ith light frustum
8:    frustum_data ← ∅
9:    for all texture_map entries with l within the extents of light frustum i do
10:      frustum_data ← frustum_data ⋃ {depth, rescaled l, rescaled dp/dl}
11:   end for
12:   angle[i] ← RemapZero(frustum_data)
13:   error[i] ← error incurred by using inital_angle + angle[i] as lexel plane angle
14: end for
15: {Perform merges on adjacent light frusta greedily until we get within budget}
16: while num_frusta >LIGHT_FRUSTA_BUDGET do
17:   for i = 0 to num_frusta − 2 do
18:      for all texture_map entries with l within the extents of light frustum i and
         i + 1 do
19:         frustum_data ← frustum_data ⋃ {depth, rescaled l, rescaled dp/dl}
20:      end for
21:      temp_angle[i] ← GetOptimalAngle(initial_angle, frustum_data)
22:      temp_error[i] ← error incurred by using inital_angle + angle[i] as lexel
         plane angle
23:   end for
24:   for i = 0 to num_frusta − 2 do
25:      res_set[i] = ComputeFrustaResolutions(i, error, temp_error[i])
26:      res_err[i] = error from frusta resolution settings according to res_set[i]
27:   end for
28:   ind ← argmin_i res_err[i]
29:   shift tbextents, error, and angle down by one after element ind
30:   error[ind] ← temp_error[ind]
31:   angle[ind] ← temp_angle[ind]
32:   lexel frusta resolutions ← res_set[ind]
33:   num_frusta ← num_frusta − 1
34: end while
```

our metric values to have shared units. Therefore we implicitly multiply each $\frac{dp}{dl}$ by a 1 lexel sized neighborhood. This converts $\frac{dp}{dl}$ to a measure of footprint size in terms of pixels. The decision to use size 1 neighborhoods is arbitrary and any constant size will do (the minimum is unchanged by constant scales of each value). Let $e_1, \ldots, e_n$ be the errors from each frustum where each e_i has been scaled such that the lexel range for that frustum is unit length. This means that for the L_1 metric we seek to minimize:

$$\frac{e_1}{x_1} + \cdots + \frac{e_n}{x_n}$$

where x_i is the lexel resolution for frustum i. In the L_2 formulation, we have:

$$\frac{e_1}{x_1^2} + \cdots + \frac{e_n}{x_n^2}$$

This minimization is subject to the constraint that:

$$x_1 + \cdots + x_n = L$$

where L is our lexel budget. Error values are computed with real number lexel allocations. Only at the end is rounding performed. If the lexel budget is a strict upper limit, each allocation can be rounded down to an integer value. Then any remaining lexels can be allocated according to a water-filling bit allocation algorithm [5].

Chapter 4

Results and Discussion

We have presented a framework for quantitatively understanding the relation between shadow mapping's remaining degree of freedom and its implications for mitigating the undersampling problem encountered during image rendering. We have also introduced a method for getting optimal sampling according to developed metrics.

4.1 Implementation Results

In this section we present details on running the revised shadow mapping algorithms and put forth some experimental results.

4.1.1 The Initial Rendering Pass

We first consider the initial rendering pass used by the algorithm. Throughout it has been noted that a low resolution rendering will be used (perhaps feeding in coarse geometry as well for a fast approximation). However, we have yet to specify an exact number of pixels. The resolution used determines how closely the discrete metrics used in the algorithm approximate the continuous ones (or the high resolution discrete metric implied by the resolution of the final image). The required number of samples depends largely on the scene. Scenes with many polygons at various spatial and angular settings may require more samples than scenes with few abrupt changes. To determine a suitable number of pixels, a variety of scenes were rendered at high resolution (1000 pixels). Each scene was subsequently rendered at lower and lower resolutions (100, 50, 25, 10, 5) and an optimal angle was determined for each using both the L_1 and L_2 versions of our algorithm. For each low resolution

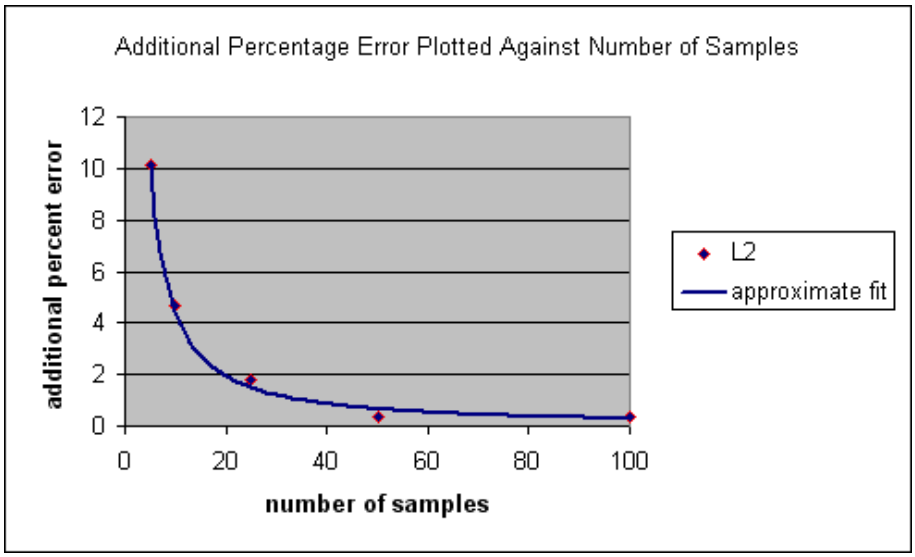


Figure 4.1: *Error added by low resolution approximation to metrics*

rendering's optimal angle, we found the percent increase in error that would result had that angle been used in the high resolution context. In this case, for the L_1 metric, error refers to the average size of the lexel footprint in the pixel domain. For the L_2 metric, error is taken to be the mean footprint size plus the standard deviation (a reasonable quantity to look at because the L_2 metric is meant to provide concentration around the mean so that few pixels will have poor shadowing). The experiments show that for many scene setups, taking only ten samples does a decent job. Most of the scene configurations gain less than five percent additional error when using angles determined from ten samples. That means a footprint would have to cover twenty pixels before a full additional pixel were covered due to this low resolution approximation. Figure 4.1 shows the numbers from one of the typical scenes.

4.1.2 Splitting the Light Frustum

For the scenes tested, the results of splitting the light frustum into smaller hulls, each with its own shadow map, were mixed. For a range of scenes, the light frustum partitioning process provided about 1.3 times improvement. As discussed later, this may or may not be significant depending on the cost of looping through the geometry. However, for a certain set of scenes with lots of projective aliasing (because polygons point toward the light), the benefits are much larger. A number of cases showed

2-3 times improvement after performing the light hull partitioning. In each scenario the benefit derived from successive splits after the first diminished quite rapidly. Undoubtedly, scenes can be constructed to greatly benefit from numerous splits; however, our experiments suggest that two or three partitions usually achieve most of the noticeable gain. In the examples run below, we settled upon a low resolution pass of 100 pixels so that the initial individual frusta in the light hull partitioning algorithm would typically have enough lexel samples to make reasonable decisions about the optimal angle for their slice of the hull.

4.1.3 Case by Case Examination

We now present some experimental results from each of the shadow mapping variants on specific cases. These examples have been constructed in flatland and serve to highlight some of the behavioral similarities and differences between the approaches. The angles for Perspective Shadow Maps are determined by the angle of the near planes when pushed as far forward as possible. Table values:

Angle is measured in degrees. The angle refers to counterclockwise rotation with 0 meaning a vertical lexel plane.

Worst is measured in pixels. The value corresponds to the size of the largest footprint of a lexel when mapped into pixel space (maximum size is taken over lexels which are visited in constructing the pixel image). The smaller the value, the better according to the L_∞ metric.

Average is measured in pixels. This represents the average lexel footprint size in pixel space.

Mean $+\sigma$ is measured in pixels. This is the size of the lexel footprint one standard deviation from the mean (in the worse direction).

Single Plane Geometry

In the first case, we have a scene consisting only of a single plane of interest. Such cases may arise in architectural walkthroughs or games in which shadows on a floor are the most important. In this particular setup we have tried to present a "typical" example of a floor with a light shining from above. Even when the world geometry is simply a single plane, perspective shadow maps gives a range of lexel plane angles that may not include the optimal. Irrespective of whether the point light is positioned in the front or back of the view frustum, perspective shadow mapping chooses a near plane corresponding a lexel plane oriented at about 91 degrees. This

is partly a consequence of the decision to push the near plane in post-projective space as close to the geometry as possible. The angle chosen in post-projective space is about 90 degrees and therefore intersects the plane at infinity close to the viewer's viewpoint pulled out to infinity. Therefore, when mapped back to world space, the lexel plane will extend to intersect the view plane around the viewer's viewpoint. Since the near plane was pushed as far forward as possible, this translates to a choice of about 90 degrees in world space as well. If instead some constant near value in post-projective space were chosen, as the light is pulled upward, the lexel angle in world space would tend toward zero, which is the optimal for the point light (because it gives more samples to the geometry closer to the viewer). Only at infinity, when the light becomes directional does it make sense to orient the lexel plane at 90 degrees (of course, at that point, the angle doesn't really matter as long as it is valid).

Normal shadow maps use the naive half-angle approach which happens to be better than the perspective shadow mapping value for this particular light position but performs poorer as the light is pushed to the back of the view frustum.

	Angle	Worst	Average	Mean+σ
Our Method (both)	9	0.975	0.960	0.968
Perspective SM	91	4.33	1.88	3.10
Normal SM	51	2.08	1.17	1.65

Simple Geometry

In this scene the view camera looks at a simple setup with lighting from above the view camera. This arrangement demonstrates that although perspective shadow maps may give some noticeable improvement overall, there are certainly a number of cases where it does poorer than the naive approach. In this case, the naive half-angle approach of just making the light direction bisect the light frustum gives an angle in the optimal range.

	Angle	Worst	Average	Mean+σ
Our Method (L1)	24.8	1.54	1.09	1.42
Our Method (L2)	29.2	1.54	1.10	1.41
Perspective SM	86.8	4.74	1.95	3.20
Normal SM	25.3	1.54	1.09	1.42

Light from Top

This is an example modeled after the second scenario presented in section 2.2.3. A point light shines from on top of the scene's geometry which includes one plane facing the view camera. Both the perspective shadow maps and the normal version do not devote enough lexel samples to this plane. Here we see that an arbitrary amount of improvement is possible over the previous approaches. This particular setup yields an approximately fifteen times improvement.

	Angle	Worst	Average	Mean+σ
Our Method (L1)	5.11	6.42	3.38	4.39
Our Method (L2)	5.13	6.42	3.38	4.36
Perspective SM	77.3	73.7	44.4	67.4
Normal SM	83.1	76.7	46.2	70.2

Projection Aliasing

This case exhibits the benefits of splitting the light frustum and performing lexel allocation when projection aliasing exists. A piece of geometry is oriented so that it points toward the light and faces the viewer flat on. In this case, over a two times improvement is obtained with the partition.

	Angle	Worst	Average	Mean+σ	Mean (two frusta)
Our Method (L1)	33.9	60.0	17.7	41.4	7.75
Perspective SM	67.7	72.7	22.1	50.5	–
Normal SM	83.7	82.5	25.3	57.5	–

4.1.4 Weighing Advantages and Disadvantages

Ultimately, with real-time rendering as our aim, the relative advantage or disadvantage of our approach with respect to previous methods must be judged in terms of the time required to obtain a certain level of shadow quality. Given some factor N improvement provided by our algorithm (either in terms of average footprint size or mean plus standard deviation), the previous methods can employ N times as many resources to achieve the same resulting shadow quality.

The extra cost incurred by our method comes from performing an initial rendering pass, running an optimal angle solver over the samples, and making multiple passes for each shadow map frustum determined by light hull partitioning. This must all be weighed against the cost suffered by previous algorithms rendering with N times higher resolution shadow maps.

Unfortunately, ultimate decision as to whether the proposed algorithm produces worthwhile gain must be deferred for later testing once versions operating in three-dimensional scenes of interest are constructed. As of now, the inability to theoretically determine a probability distribution over all possibly encountered scenes, nuances of hardware and driver timing, and complications of extrapolating to a higher dimension render speculation about final performance very imprecise. The system will simply have to be timed in a set of typical test cases once implemented in full to give a real sense for relative performance.

4.2 Future Work

4.2.1 Generalizing to 3D

The natural next step is to generalize the algorithm to three dimensions. While much of the framework and analysis follows through, the extra dimension adds a couple complications.

Metrics

For 2D image and lexel planes it is less straightforward to settle upon the "right" metric. It would be convenient simply to extend the metrics such that the derivatives turn into directional derivatives for movements both horizontally and vertically in the lexel space. For instance, the L_2 metric might become:

$$\sum_{i,j}\left[\left(\frac{\partial x}{\partial u}\right)^2_{i,j}+\left(\frac{\partial y}{\partial u}\right)^2_{i,j}+\left(\frac{\partial x}{\partial v}\right)^2_{i,j}+\left(\frac{\partial y}{\partial v}\right)^2_{i,j}\right]$$

where (u, v) are the texture coordinates and (x, y) are the pixel coordinates. The i, j subscripts loop over the lexels in Ω'.

While this is a simple extension, the "geometric stretch" measure proposed by Sander *et. al.* may be a more desirable measure [8]. Not only does it try to minimize the area covered by a footprint in the pixel domain, it reduces the amount the footprint stretches so that lexels will have more local effects on the pixels. Geometric stretch is defined to be the trace of the metric tensor. The metric tensor is the product of the Jacobian transpose and the Jacobian for our mapping from lexel to pixel space. Equivalently, the geometric stretch is also the sum of the squares of the Jacobian's eigenvalues.

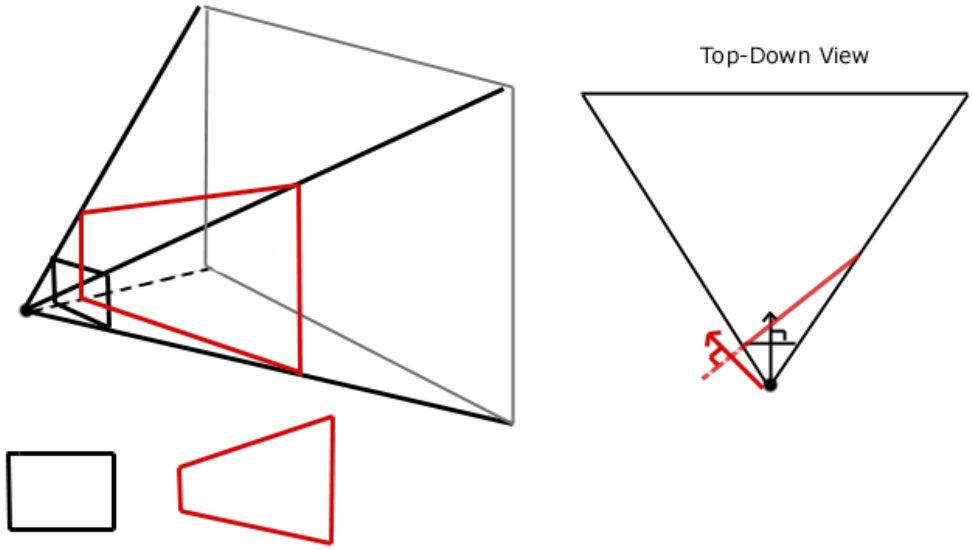


Figure 4.2: *Change of frustum extents with change of angle*

Lexel Re-mapping

While in 2D the metrics could be parameterized in terms of angle with relative ease, in 3D it becomes more tricky to analytically classify the extents of the frustum based upon rotations in 3D. There are now three rotations to consider – pitch, yaw, and roll. There exist algorithms for determining the minimal rectangle enclosing points in a 2D plane, which implies a method for finding an optimal roll and light frustum extents given a fixed pitch and yaw. However, categorizing how the results of such an algorithm change based on changes in pitch and yaw is more complex.

The difficulty in determining the light frustum extents is that after a rotation of yaw or pitch, the top, bottom, left, and right extents move under the rotation to give a non-rectangular near plane (imagine the intersection of the rotated near plane with the original convex hull defined by the old near plane extents). In fact the dimensions of the new near plane can be quite skewed. Figure 4.2 shows that for a set of points in the world that define a particular light hull of interest (chosen to have a rectangular near plane in the initial setting for simplicity), rotation of the light will lead the near plane's intersection with that hull to be non-rectangular. The projection matrices supported by graphics hardware presuppose a near plane that is always rectangular (allowing a straightforward affine map between it and the lexel or pixel grid). Therefore, to continue allowing an affine map to the grid, it seems that having some lost lexels is inevitable.

Writing down a good approximation to the optimal light hull extents after rotation of the light's axes such that the near plane is rectangular is essential to the algorithm as developed in 2D. A poor approximation may find false minima that can have little correlation with the right answer or provide minimal improvements relative to the computational cost.

Light Hull Partitioning

The frusta merging process during lexel allocation also becomes more constrained. Merges of frusta can no longer be done simply by taking adjacent frusta. Collapses must follow some type of quad-tree rule so that in the end the remaining frusta are rectangular. The lexel allocations to each individual shadow map also cannot be added on the lexel level anymore either. Since shadow map textures must be rectangular, adding more lexels to a shadow map means adding entire rows or columns of samples.

Newton's Method

In a higher dimension, there is always a concern that Newton's Method may not converge to a solution nicely. The functions whose roots are solved for must be chosen such that Newton's Method will give reasonable results. Perhaps other approximations can be made to allow for a closed form approximate solution.

4.2.2 Running the Algorithm in a Different Projective Space

Bringing the algorithm into a different projective space may allow it to settle upon angles that were invalid in the other space. Although from an implementation standpoint the move to a different projective space may not be worth the effort, it would be interesting to classify how much improvement might be extracted.

Acknowledgements

My first round of thanks goes out to Shlomo for making my undergraduate experience here so rewarding. He has provided me with many opportunities I would most likely have gone without elsewhere.

I would also like to give a huge thanks to my parents and siblings for their unwavering support throughout the years. I appreciate the many sacrifices you all have made to help me get this far.

Thanks as well to my readers for getting through this thesis and giving these thoughts your attention.

Appendix A

Background Material

In this chapter, we shall first motivate our discussion of shadowing algorithms with a brief overview of the *rendering*, or image creation, process. We will then turn our attention to understanding the mathematics behind the texture mapping process.

A.1 Image Rendering

Computer graphics in essence is a study of light simulation. Given some representation of a three-dimensional world (perhaps as a soup of triangles or a set of algebraic functions defining implicit surfaces), we want to render a flat image consistent with what a person living in that world would see. Our model for a viewer's eye is the pinhole camera. Light from the world passes through the camera's pinhole, and forms an inverted image on the film plane located behind the hole. Because in a virtual setup, locating the film plane behind the hole is an arbitrary decision (required in real life only by physical limitations), in computer graphics it is customary to move the film plane in front of pinhole to deal with non-inverted images. This film plane, henceforth referred to as the *near plane*, can be overlayed with some finite and discrete grid, often chosen so as to be in one-to-one correspondence with a grid imposed by the output display (e.g., a computer monitor or LCD screen). Each partition of the grid is referred to as a picture element, or *pixel*. Image rendering then becomes the determination of what color to light each pixel.

A.1.1 Ray-Tracing

A natural approach to determining the color of a pixel is to pursue further the analysis of what the physical behavior of light mandates. In the natural world, light

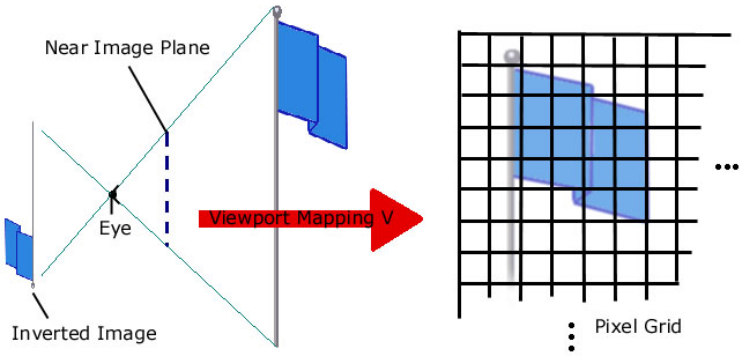
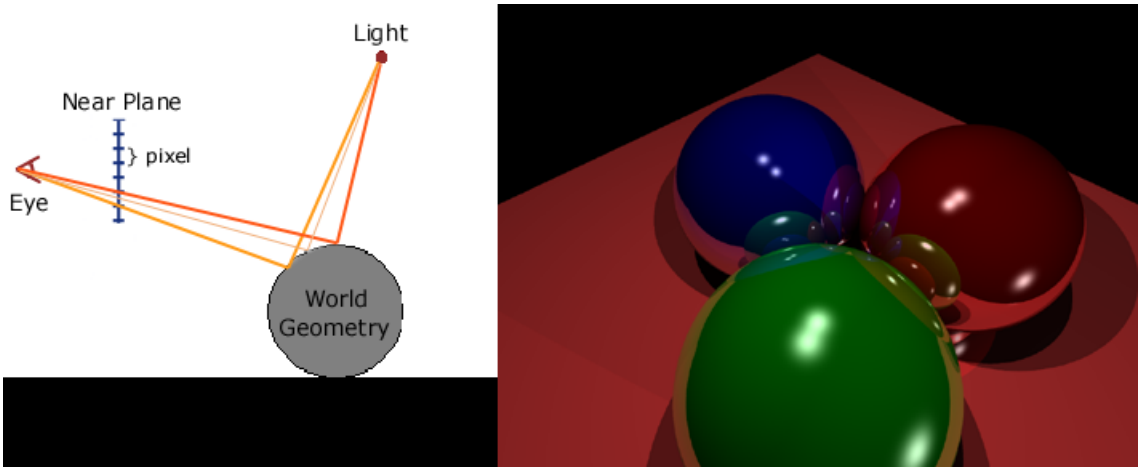


Figure A.1: *View camera setup*



(a) *(b)*

Figure A.2: *Ray-Tracing*
(a) *light rays contributing to the lighting seen at one pixel* (b) *a ray-traced image*

is emitted from a source such as the sun, reflects and refracts off objects within the world, and finally enters an eye's pupil to help constitute the photons that form the eye's sensory image of the world. Ray-tracing is based on the observation that these physical laws governing light's transport are consistent if applied backward. Instead of starting from the light source and traveling to the eye, the eye sends out "feeler rays" in the reverse direction to determine what objects are encountered.

Algorithm 3 color RayTrace(point $\mathbf{x}$, direction $\vec{r}$, int *depth*)

```
1: if depth ≥ MAXIMUM_DEPTH then
2:   return BLACK
3: else
4:   ipoint ← ∅
5:   for all objects Γ in world do
6:     list ← set of intersection points of x + λr⃗ with Γ for λ ∈ (0, +∞)
7:     if list contains a point closer to x than ipoint then
8:       ipoint ← closest point
9:     end if
10:  end for
11:  if ipoint == ∅ then
12:    return BLACK
13:  else
14:    d⃗_r ← direction of reflected ray
15:    d⃗_t ← direction of refracted (transmitted) ray
16:    local_color ← radiance determined by local material and light interactions
17:    if Γ is not a light source then
18:      reflection_color ← RayTrace(ipoint, d⃗_r, depth+1)
19:      transmission_color ← RayTrace(ipoint, d⃗_t, depth+1)
20:    else
21:      reflection_color ← local_color
22:      transmission_color ← local_color
23:    end if
24:    return Mix(local_color, reflection_color, transmission_color)
25:  end if
26: end if
```

The process of creating a ray-traced image proceeds as shown in algorithm listing 4. Notice that the ray tracing subroutine terminates only upon reaching some maximum depth of recursion, when intersecting the ray with a light source, or upon shooting the ray off into the void. Ideally, there would not exist a limit to the recursive depth; however, such a restriction is enforced to keep the computations running within a prescribed time bound. Proper shadowing can be fit into this ray-

Algorithm 4 void CreateRayTracedImage(void)

```
1: for all pixels p in the near plane do
2:     r⃗ ← the ray direction passing from the pinhole through pixel p
3:     p's color ← RayTrace(pinhole, r⃗, 0)
4: end for
```

tracing framework on line 16 of algorithm 3, in which the local color is computed given some model of the local material and light interactions. At this stage, it is straightforward to send *shadow feeler* rays from the intersection point *ipoint* to each of the light sources in the scene (assumed here to be point sources for simplicity) and check whether that light source is the closest object to *ipoint*. For each light source that has no other objects occluding it, we can add it's contribution to the local color. We need only worry about direct illumination because any indirect lighting will presumably be handled by the recursive calls to RayTrace().

Benefits of Ray-Tracing

Ray tracing produces convincing results because it models how light really behaves. A number of refinements exist for allowing the ray-tracing technique to capture advanced lighting effects such as caustics and volumetric fog. Especially pertinent to this exposition is the fact that shadowing can be handled in a straightforward manner.

Drawbacks of Ray-Tracing

Ray-tracing, although producing very high quality outputs, necessitates a degree of computation that is currently infeasible for real-time rendering. For every pixel in the image drawn, passes over the geometry must be made to perform intersection tests. These tests generally prove to be quite costly and, even after optimizations, prohibit full-fledged ray-tracing to be done at interactive frame rates.

A.1.2 Polygon Rasterization

Polygon rasterization refers to the process of converting polygons into a series of parallel horizontal scanlines, or *rasters*, on a cathode-ray tube (CRT) or analogous display. Each polygon is then rendered scanline by scanline. Rasterization takes advantage of the fact that polygons are flat and that under linear and projective transformations lines map to lines to speed up raster rendering. Given two points of a polygon that transform to the endpoints of a raster, all the relevant information

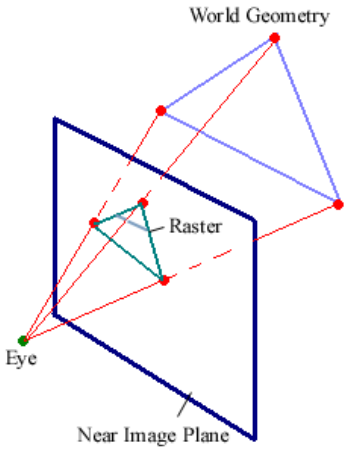


Figure A.3: *Triangle Rasterization*

about raster points in between can be computed by a series of linear and linear rational interpolation steps, the details of which will be visited in section A.2. Although the same basic properties will hold for any polygon, for simplicity and convenience triangles are often chosen as the basic rendering primitive.

Algorithm 5 void RasterizeScene(void)

```
1:  for all triangles Γ in the world do
2:    V1, V2, V3 ← vertices of Γ transformed to near plane coordinates
      {the rasters are determined by V1, V2, V3}
3:    for all raster scanlines do
4:      for all pixel locations p in raster do
5:        interpolate to get color and depth for Γ at location p
6:        if depth is closer to the near plane than previously seen depths at p then
7:          color pixel p according to color
8:        end if
9:      end for
10:   end for
11: end for
```

Algorithm listing 5 presents the core methodology for the rendering process according to triangle rasterization. The *depth* value computed at each pixel location is the perpendicular distance to the plane parallel to the near plane containing the point on Γ seen through p. Determination of whether *depth* is the closest yet seen

value at p can be achieved by use of a *depth buffer* in which a buffer the size of the pixel grid is created to store the closest encountered *depth* values for each pixel location. This buffer is often also referred to as a *z-buffer* because view cameras are usually set up to look down the negative direction of their $\hat{z}$-axis.

Benefits of Polygon Rasterization

Polygon rasterization is efficient because once the polygon's vertices are mapped to the near plane, interpolation can be used to fill in the convex hull – a process much faster than performing intersection tests per pixel.

Drawbacks of Polygon Rasterization

Polygon rasterization, while fast because it can render a primitive with knowledge of only the transformed vertices, treats each polygon independently and therefore loses some of the more global information. In particular, a triangle rendered by interpolation has no way of knowing which pixels if any correspond to portions of the triangle occluded from the light by some other triangle. Rasterization therefore requires a separate algorithm for determining shadowing information.

A.2 Texture Mapping Background

Texture mapping is the process of applying a two-dimensional image, or texture map, to scene geometry. Texture maps get their name from the fact that images are often mapped onto geometry to add greater detail to an object without increasing geometric complexity. Often, such texture maps are mapped linearly onto a single triangle or geometric primitive. However, texture maps can also be projected onto world geometry as a slide projector would cast a slide's picture onto the world. This is termed *projective texturing*. Current graphics hardware has efficient methods for handling both linear and projective applications of textures to world geometry.

Understanding the texturing process is important because shadow maps take advantage of the same mechanism used to perform projective texturing for determining what lexel sample to reference at each pixel of the view image. The idea is that taking a snapshot of geometry depth from the viewpoint of the light is analogous to projecting that depth map onto the world's geometry and claiming each sample value as truly representative of the depth of the geometry hit. This feature of shadow mapping allows them to be implemented efficiently on current graphics hardware and therefore allows for real-time rendering.

To analyze the action of capturing a depth map from the perspective of the light, we decompose the mapping process into two parts – a projective mapping P and an orientation-preserving rigid motion M. M describes the position and orientation (axes) of the light's view. P maps a frustum aligned with the light's axes into a cube such that geometry can be orthographically projected to the near plane to form an image. The motion M is further decomposed as the composition of a translation T with a rotation R. At this point it is useful to differentiate between the representation of points and of vectors in our system. Because we are interested in affine and not simply linear motions [1], we append a fourth coordinate to the usual $(x,\ y,\ z)$ representation in 3D that is 0 for vectors and 1 for points. It is easily verified that the usual intuitions concerning points and vectors are preserved (e.g., the sum of two vectors is a vector, the sum of a point and vector is a point, etc.). This convention allows us to write T and R as 4x4 real matrices. In a 2D world, points and vectors are given three coordinates and the transformations are represented with 3x3 matrices.

T is chosen to represent the translation from the origin to the light camera with respect to world coordinates. R is the rotation with respect to world space necessary to line up with the orientation of the light camera. Often in linear algebra it is convenient to leave the basis of the space implicit and work only with coordinates. However, when dealing with change of basis it is useful to utilize the "concrete to abstract" map[2]. Let $\mathbf{L}$ be the matrix whose column vectors are the basis elements of the light camera's basis. Let $\mathbf{W}$ be the matrix whose columns represent the world's axes. Then, we can relate the two with:

$$\mathbf{L} = \mathbf{W}M = \mathbf{W}TR$$

The transformation $M = TR$ describes the mapping from world aligned axes to the light camera's axes. Then any point $\mathbf{p}$ described in coordinates with respect to world space can be re-represented as coordinates in light space by observing:

$$\mathbf{W}\mathbf{p} = (\mathbf{L}M^{-1})\mathbf{p} = \mathbf{L}(M^{-1}\mathbf{p})$$

Therefore the "abstract to concrete" map[3] gives us $M^{-1}\mathbf{p}$ as the coordinates of $\mathbf{p}$ with respect to the light domain.

The projective transform takes a view frustum and maps it to the canonical cube with coordinates ranging from -1 to +1. Since the projective transform is

[1] linear mappings do not describe all orientation-preserving Euclidean motions because it requires the origin be mapped to the origin. Therefore it does not accommodate translations in particular.

[2] writing the concrete coordinates as a linear combination of chosen basis vectors from an abstract vector space

[3] writing a point in the abstract vector space as the coordinates, or coefficients, of the chosen basis vectors

meant to capture perspective, rays from the viewpoint's origin diverge as they move outward so that objects pulled farther away appear smaller on the near image plane. The size of an object is inversely proportional to its distance from the view plane.

Although this inverse relation between near plane coordinates and object depth exists, we nonetheless attempt to model the transformation with a matrix. A view frustum can be defined in terms of six parameters: the near plane depth n, the far plane depth f, the top and bottom extents of the near plane t and b, and the left and right extents of the near plane l and r. The projection matrix that transforms this frustum into the canonical cube is given by:

$$P = \begin{bmatrix} -\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & -\frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

It is conventional to set up the camera such that in the camera's frame, the direction to the right corresponds to the $\hat{x}$ axis, the upward direction corresponds to the $\hat{y}$ axis, and the viewer looks down the $-\hat{z}$ axis. Therefore, n and f take on negative values.

Notice that this matrix assumes the light's near plane will always be perpendicular to the light's $\hat{z}$-axis. Therefore the matrix is completely determined by the extents required to project light onto the entire scene of interest. So although we conceptually understand shadow mapping's degree of freedom as choosing an angle for the light's near (film) plane, algebraically we represent this as a choice of rotation of the light's basis. The light's rotation will determine its $\hat{z}$-axis and therefore fix the matrix P by the extents required to cover the light hull. This understanding puts limits on allowable rotations of the light (or angles of the lexel plane). Any rotation such that the light hull does not live entirely in the $-z$ subspace is "invalid" because P cannot be defined. Physically, this corresponds to requiring a finitely sized lexel plane. Rotations such that the lexel plane shoots off to infinity are unrealizable.

Now we return to addressing the fact that projection is not an affine mapping and therefore a 4x4 matrix multiplication cannot describe it fully. This projection matrix maps a point $[x, y, z, 1]^t$ represented in coordinates with respect to the viewpoint's axes to some point $\mathbf{p}'$:

$$P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \mathbf{p}'$$

Because the w coordinate has been setup by matrix P to store the depth of the geometry as seen from the viewpoint, division of $\mathbf{p}'$ by w yields the desired viewer's

post-projection coordinates $[x_v, y_v, z_v, 1]^t$ (the division gives us the inverse distance relationship required for perspective). By convention then, we will write $\mathbf{p}'$ as:

$$\mathbf{p}' = \begin{bmatrix} (x_v w) \\ (y_v w) \\ (z_v w) \\ w \end{bmatrix}$$

We now denote all points and matrices related to the view camera screen (pixel) space with the subscript s and the points and matrices pertaining to the light's texture map space with the subscript t. We can then relate the transformation from pixels to texture coordinates with the following mathematical statement:

$$\frac{V_t P_t M_t^{-1} M_s P_s^{-1} V_s^{-1} \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix}}{\frac{w_t}{w_s}} = \begin{bmatrix} x_t \\ y_t \\ - \\ 1 \end{bmatrix}$$

V represents the bijective mapping from a canonical cube's near plane to the coordinates of the grid overlayed on top of it. Here we have omitted a z_t because we are performing lookups on a 2D texture. The depth value of interest is obtained by the actual lookup.

Note that the product $V_t P_t M_t^{-1} M_s P_s^{-1} V_s^{-1}$ is a 4x4 matrix and therefore represents a linear transformation. Therefore linear movements in screen space coordinates correspond to linear movements in

$$\begin{bmatrix} (x_t q) \\ (y_t q) \\ - \\ q \end{bmatrix}$$

where $q = \frac{w_t}{w_s}$. This means that given these values at a raster's endpoints, all the intermediate pixels texture coordinates can be computed by linear interpolation followed by a division per pixel (to divide the interpolated $x_t q$ and $y_t q$ by the interpolated q value). This division of linear interpolants is referred to as *linear rational interpolation.*

In the reduced dimension of flatland, these relations still hold; we simply drop a coordinate.

Appendix B

Proofs

B.1 Proving Constant $\frac{dp}{dl}$ Over Each Polygon

Proposition 3

For arbitrary world geometry, constant $\frac{dp}{dl}$ on each polygon can be guaranteed if and only if the light is positioned in the view plane perpendicular to the view direction. Furthermore, the light direction must be oriented parallel to the view direction.

Proof: We seek to show that setting the light plane to be parallel to the viewer's near plane in the miner's headlamp case leads to constant sampling density on each polygon individually. To show this, we show that movements in screen space (over a given polygon) correspond linearly to moving in texture space. As presented in section A.2, texture coordinates for pixels in a raster are computed by noting that the values:

$$\begin{bmatrix} (x_t q) \\ (y_t q) \\ - \\ q \end{bmatrix}$$

where $q = \frac{w_t}{w_s}$ are affine in the screen coordinates. This allows us to perform linear rational interpolation to get the desired texture coordinates x_t and y_t. However, notice that if q is a constant shared by each vertex, then linear interpolation yields the same constant q everywhere within the polygon's convex hull. Furthermore, division by q at each pixel is just division by the same constant each time so we can factor it out and notice that the texture coordinates are merely linearly interpolated within a polygon. For constant q, linear rational interpolation is reduced to linear interpolation. Under arbitrary world geometry assumptions, constant q is the only case in which we can guarantee linear interpolation is sufficient to generate the

correct texture coordinates because there clearly exists choices of world coordinates in which linear interpolation will not yield the right values (linear interpolation is not as expressive as linear rational interpolation). Having q constant for each transformed world coordinate means $w_t = cw_s$ for some constant c.

To understand the behavior of w_s and w_t we can examine the following relations:

$$P_l M_l^{-1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_t w_t \\ y_t w_t \\ - \\ w_t \end{bmatrix}$$

$$P_s M_s^{-1} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_s w_s \\ y_s w_s \\ - \\ w_s \end{bmatrix}$$

Notice that the last row of any projection matrix is given by [0, 0, -1, 0]. Therefore, when fed the same world points $[x, y, z, 1]^t$, the transformations will yield w_t and w_s in constant ratio if the third rows of M_l^{-1} and M_s^{-1} are constant multiples of each other. Each rigid motion M_i^{-1} can be written as the composition of a rotation and translation, $R_i^{-1} T_i^{-1}$.

$$T_i^{-1} = \begin{bmatrix} 1 & 0 & 0 & -t_{i_x} \\ 0 & 1 & 0 & -t_{i_y} \\ 0 & 0 & 1 & -t_{i_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\vec{t_i}$ is the position of camera i in world coordinates.

$$R_i^{-1} = \begin{bmatrix} & \vec{r_{i_x}}^{\,t} & & 0 \\ & \vec{r_{i_y}}^{\,t} & & 0 \\ & \vec{r_{i_z}}^{\,t} & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\vec{r_{i_k}}^{\,t}$ is the row vector representing the k-axis of camera i after rotation. The third row of M_i^{-1} is therefore given by $[\ -\ \vec{r_{i_z}}^{\,t}\ -\ \vec{r_{i_z}} \cdot (-\vec{t_i})\ ]$. The third rows of M_i^{-1} must be the same element wise up to the constant c. However, because M_i^{-1} is a rigid motion, $\vec{r_{i_z}}^{\,t}$ is unit length for both cameras and therefore $c = 1$. Consequently, $\vec{r_{i_z}}^{\,t}$ is fixed and $\vec{t_i}$ must be such that $\vec{r_{i_z}} \cdot (-\vec{t_i})$ is a shared constant value. Let us denote that value k. Then for some $\vec{t}$ satisfying $\vec{r_{i_z}} \cdot \vec{t} = k$, we have:

$$\begin{aligned} \vec{r_{i_z}}^{\,t} \cdot (\vec{t} + \triangle\vec{t}) &= k \\ \vec{r_{i_z}} \cdot \vec{t} + \vec{r_{i_z}} \cdot \triangle\vec{t} &= k \\ \vec{r_{i_z}} \cdot \triangle\vec{t} &= 0 \end{aligned}$$

So translations (and only such translations) perpendicular to $\vec{r_{i_z}}$ are allowable because they preserve this property. $\vec{r_{l_z}}$ and $\vec{r_{s_z}}$ being fixed means the light whose position and rotation is given by M_l must look down the same z-axis as the view camera. ■

Bibliography

[1] C. Everitt. Shadow Mapping. nVIDIA Corporation, 2001. from http://developer.nvidia.com/docs/IO/1252/ATT/ShadowMapping.pdf

[2] C. Everitt, M. Kilgard. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. nVIDIA Corporation, 2002. from http://developer.nvidia.com/docs/IO/2585/ATT/RobustShadowVolumes.pdf

[3] O. Faugeras. **Three-Dimensional Computer Vision: A Geometric Viewpoint** The MIT Press, Massachusetts 1993.

[4] R. Fernando, S. Fernandez, K. Bala, and D.P. Greenberg. Adaptive Shadow Maps. *Proceedings of SIGGRAPH 2001*, pp. 387-390, 2001.

[5] A. Gersho, R. Gray. **Vector Quantization and Signal Compression** Kluwer Academic Publishers, 1991.

[6] J. Hubbard, B. Hubbard. **Vector Calculus, Linear Algebra, and Differnetial Forms** Prentice-Hall, Inc., New Jersey 1999.

[7] W. Reeves, D. Salesin, and R. Cook. Rendering Antialiased Shadows with Depth Maps. *Computer Graphics (Proceedings of SIGGRAPH '87)*, 21(4):283-291, 1987.

[8] P. Sander, S. Gortler, J. Snyder, H. Hoppe. Signal-Specialized Parametrization. Thirteenth Eurographics Workshop on Rendering, 2002.

[9] M. Stamminger, and G. Drettakis. Perspective Shadow Maps. *Proceedings of SIGGRAPH 2002*, pp. 557-562, 2002.

[10] A. Watt, M. Watt. **Advanced Animation and Rendering Techniques: Theory and Practice**. Addison-Wesley, New York 1992.