Scene Optimized Shadow Mapping

Hamilton Y. Chong and Steven J. Gortler

TR-07-07

Computer Science Group
Harvard University
Cambridge, Massachusetts

Harvard SEAS Technical Report (2007)

Scene Optimized Shadow Mapping

Hamilton. Y. Chong[†] and Steven J. Gortler[‡]

Harvard Univeristy

Abstract

Shadow mapping possesses an aliasing issue that is difficult to control. Recently a number of techniques, beginning with Perspective Shadow Maps *have been proposed to choose the best possible 4 by 4 matrix for the light's projection matrix. These new methods offer no guarantees of optimality, and can perform very poorly for some light configurations. Here we describe a method that chooses the 4 by 4 matrix using an optimization framework. We then describe numerical experimental results comparing our method to some of the previously suggested techniques.*

1. Introduction

Although shadow mapping has found widespread adoption among those who seek realistic and believable lighting, the algorithm possesses an aliasing issue that is difficult to control. Recent insights have spurred a flurry of activity and bred a number of novel heuristic approaches to such control [SD02, MT04, WSP04]. Unfortunately, many of the proffered heuristics, while improving shadow quality for certain scenes, offer little guarantee outside these specific domains.

In this work, we generalize the optimization framework in [Cho03] to handling all 3D scenes. In particular, we present a metric formulation of shadow quality that consequently permits us to solve for approximately optimal parameters (under a chosen metric) for mitigating the aliasing issue. We do a low resolution readback of the scene to get an estimate of the geometry and run an inexpensive optimizer at each frame. At the expense of the readback and added CPU computation we are able to obtain better shadows using the same shadow resolution.

Posing shadow map setup as an optimization problem guarantees a type of robustness to any configuration of lights and camera. This is in stark contrast to heuristic approaches that (while perhaps doing no worse than normal shadow mapping) can at times be arbitrarily far from optimal. The presence of a metric further allows us to quantify the degredation in shadow quality for various shadow map and resource choices.

The optimization framework can also notably benefit both offline and real-time rendering. In offline production settings, shadow maps are often manually tuned with meticulous care. Such work is laborious and not scalable. This research provides a framework for automating this process by determining optimal shadow map allocations and parameters that guarantee shadow quality within a specified epsilon tolerance. Real-time applications with constrained configurations can also make use of this as precomputation. Then during runtime, the metric-based optimization ascertains the best usage of the allocated resources.

Previous Work. The literature on shadow mapping is extensive. Since its initial exposition in [Wil78] it has gained wide adoption and stimulated substantial research. Variants aimed at addressing the aliasing issue may be divded into 3 broad categories: (1) those that use information beyond a single float for shadow determination [RSC87, LV00, CD03, CD04, SCH03], (2) those that use a hierarchy and split the light frustum into sub-frusta [FFBG01, Cho03, Arv04, LTYM06], and (3) those that utilize freedom in shadow map setup to affect sampling distribution [SD02, Koz04, WSP04, MT04, CG04, LTYM06]. Algorithms in each category work largely independently, so many hybrid approaches are possible.

In this work, we focus on the third category. Following the initial observation that shadow mapping possesses ex-

[†] e-mail: hchong@fas.harvard.edu

[‡] e-mail:sjg@cs.harvard.edu

tra degrees of freedom that can be used to fight aliasing [SD02, Koz04], a number of heuristic warping algorithms have been proposed to get shadow map sampling to better match the view camera's sampling [WSP04, MT04]. In [Cho03], the rudiments of an optimization framework is presented, but the work analyzes only the flatland case. The plane optimal algorithm of [CG04] is the only attempt at provable shadow quality in 3D, but only provides guarantees for a few selected planes of interest. Furthermore, it is shown there exist configurations for which a single shadow map cannot both obtain perfect sampling on the specified plane and simultaneously capture the entire shadow frustum of interest, thus requiring division into smaller sub-frusta. Our work here is closest to [Cho03] in taking the optimization route. We generalize the setup and provide solution to this more difficult 3D problem.

2. Method

2.1. Notation

We describe a point in R^3 using a homogeneous coordinate vector $[X^1, X^2, X^3, 1]^t$ in "light space". Light space uses some orthonormal coordinate frame whose origin coincides with the light's position [†]. A point can be mapped to its screen space coorditates (position and z-buffer value) $[x^1, x^2, x^3, 1]^t$ as

$$\begin{bmatrix} x^1 w_q \\ x^2 w_q \\ x^3 w_q \\ w_q \end{bmatrix} = Q \begin{bmatrix} X^1 \\ X^2 \\ X^3 \\ 1 \end{bmatrix} \tag{1}$$

where Q is the appropriate 4 by 4 matrix that combines viewport, projection, and modelview. Q is given as part of the scene description. The w_q must be divided out to obtain the screen space coordindates.

A point can also be mapped to its shadow map coordinates (position and z-buffer value) $[u^1, u^2, u^3, 1]^t$ as

$$\begin{bmatrix} u^1 w_p \\ u^2 w_p \\ u^3 w_p \\ w_p \end{bmatrix} = P \begin{bmatrix} X^1 \\ X^2 \\ X^3 \\ 1 \end{bmatrix} \tag{2}$$

Where P is the "light matrix".

As first pointed out in [SD02], the matrix P is not completely determined by the scene input, and thus there is some freedom in chosing the entries in P. We are of course not allowed to choose any 4x4 matrix for P (see figure 1). Because we started with a light space frame, P takes the form of a *camera matrix*, and must map the origin $[0,0,0,1]^t$ (the light's position) to a point at inifinity in the z-direction, with

$$P = \begin{bmatrix} \diamondsuit & \diamondsuit & \diamondsuit & 0 \\ \diamondsuit & \diamondsuit & \diamondsuit & 0 \\ - & - & - & - \\ \clubsuit & \clubsuit & \clubsuit & 0 \end{bmatrix}$$

Figure 1: *The elements marked ♢ represent the affine freedoms. The elements marked ♣ control the projective freedoms and, as a unit vector, exactly represent the optical axis. The 3rd row affects only z precision, not sampling. The rest are zero.*

coordinates $[0,0,k,0]^t$ for some constant k. Thus, there must be zeros in the three matrix entries shown in figure (1).

We also have the following inequality constraints:

$$0 \le u^1 \le \text{width}_u, \quad \forall \text{relevant geometric points}$$
$$0 \le u^2 \le \text{height}_u, \quad \forall \text{relevant geometric points}$$

The relevant geometric points of interest are any scene points that could be in shadow and appear in the final image. We must ensure that these points fall into the shadow map's bounds.

Subject to these constraints, we want to choose P to optimize the quality of the shadow sampling.

We first note that only the first, second, and fourth rows of P (determining shadow map image coordinates u^1 and u^2) affect sampling quality (discounting z-buffer precision issues). Therefore, we ignore the third row of P altogether as something to be chosen using traditional methods. This leaves us with only nine entries of P for optimization. There are in fact only eight true degrees of freedom since the action of P is not affected by scaling the matrix. We choose to disambiguate this scale factor by enforcing that the fourth row has unit length.

We refer to the six degrees of freedom in the first 2 rows of P as *affine* degrees of freedom. This is because they can be completely controled by left multiplying our current P by a viewport-like matrix of the form

$$\begin{bmatrix} \heartsuit & \heartsuit & 0 & \heartsuit \\ \heartsuit & \heartsuit & 0 & \heartsuit \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

where the entries marked ♡ effectively perform a 2d affine transform in the shadow map domain.

We refer to the unit 3-vector in the fourth row of P as our projective degrees of freedom. We represent it using the symbol $\tilde{p}'_4$. It can be thought of as the direction of the optical axis of the light's camera. (This is because all points that are on the linear subspace orthogonal to this axis will map to infinity in screen coordinates).

[†] Our method can also be easily applied to directional lights, but we will omit the technical details for brevity.

2.2. Optimization Framework

During shadow mapping we map a screen point to a shadow map point using:

$$\begin{bmatrix} u^1 \frac{w_p}{w_q} \\ u^2 \frac{w_p}{w_q} \\ u^3 \frac{w_p}{w_q} \\ \frac{w_p}{w_q} \end{bmatrix} = PQ^{-1} \begin{bmatrix} x^1 \\ x^2 \\ x^3 \\ 1 \end{bmatrix} \quad (4)$$

Away from depth discontinuities in the screen image, this defines a continuous and locally invertable map from $[x^1, x^2]^t$ to $[u^1, u^2]^t$, and thus has a well defined and invertable Jacobian. To compute this Jacobian at a point in the screen image, one must know the depth of the observed point, as well as its surface normal. For a general scene, this will necessitate some form of readback from the framebuffer.

We want to choose our degrees in freedom of P such that each pixel on the screen gets as large a footprint in the shadow map as possible under this map. Expressed as a minimization problem, we want to ensure that each lixel gets a small footprint on the screen under the inverse map. This lixel footprint is described differentially by the inverse Jacobian. Thus the entries of this inverse Jacobian, $\left\{ \frac{dx^j}{du^i} \right\}$, provide natural "badness" measures. Large values mean that for unit steps in u^i, we move a lot in x^j in screen space. This implies one shadow map sample is referenced by many pixels.

We thus seek to minimize an integral of the form:

$$h(P) \equiv \int g\left(\frac{dx^1}{du^1}, \frac{dx^1}{du^2}, \frac{dx^2}{du^1}, \frac{dx^2}{du^2} \right) I(x^1, x^2) dx^1 dx^2 \quad (5)$$

where g is some (yet to be determined) function, and I is a weighting function that is zero outside the screen extents. Assuming each screen pixel's shadow quality near shadow boundaries is equally important, we make I an indicator function (sum of dirac deltas) for the pixel center samples near shadow boundaries. This samples the shadow boundaries according to the screen induced distribution. In our implementation we choose g to correspond to the L2 norm, so our energy (5) becomes:

$$h(P) = \sum_{\text{pixel samples}} \left[\sum_{i,j=1,2} \left(\frac{dx^i}{du^j} \right)^2 \right] \quad (6)$$

2.3. Iterative Solution

The basic algorithm is summarized in Algorithm 1. Each step is described in further detail following the pseudocode.

In step 1, the algorithm computes the geometric information required for optimization. This is accomplished with an initial rendering of the scene from the viewpoint of the camera in which the color buffer is used to store a per light bit-mask that marks which screen pixels are shadowed (and would otherwise get a nonzero shading contribution) for each light in question. The usual four byte color buffer supports up to 32 such light bit masks. Optimized shadow maps from the previous frame are used for shadow determination in this pass. While creating and reading back full sized z-buffers and bit buffers guarantees our shadow maps (post optimization) are tuned to the needs of each frame, we can do just as well in most cases by subsampling. We accomplish this by rendering step 1 in low resolution.

Algorithm 1 OptimizeShadowMap

1. Render scene (shadow receivers) from camera's viewpoint in low resolution → z and bit buffer
2. Read back z-buffer and bit buffer
3. Compute light space pts $\vec{X}$ corresponding to camera samples near shadow boundary
4. Repeat "k" times
   a. Compute gradient direction $\frac{\partial}{\partial \vec{p}_4^t}(h)$
   b. For each candidate θ
      i. Rotate $\tilde{p}_4^t$ in negative gradiant direction by θ
      ii. Approximate optimal affine parameters given $\tilde{p}_4^t$
      iii. update "optimal P" if best seen so far
5. output the "optimal P" if is a "sufficient improvment" over that from the previous frame.

Figure 2: *The samples used in optimization.*

Step 2 is trivial for offline software renderers; details for amortizing costs in real-time renderers are presented in section 2.4.

In step 3, we use the bit masks to find pixels near shadow boundaries for each light's optimization. This improves quality since we optimize only over samples we really care about (i.e., shadow boundaries on receivers that front-face the light). See for example Figure 2 to see the optimization samples used for one scene. The z-buffer allows us to compute the light space coordinates of each sample.

Step 4 defines our optimization procedure. The vector $\frac{\partial}{\partial \vec{p}_4^t}(h)$ denotes the gradient of h with respect to the three components of $\vec{p}_4^t$. Its computation is outlined in the appendix. Since we enforce $\vec{p}_4^t$ to be unit length, we want to update $\vec{p}_4^t$ with a rotation instead of the more usual addition. By

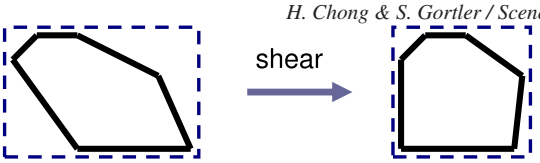

Figure 3: *The convex hull (black) is surrounded by the minimum area rectangle algined with the bottom edge. Since both left and right extents of non-bottom edge points are to the left of their respective bottom edge extents, we can apply an affine shear to shift the points right so that the leftmost non-bottom edge point aligns with the leftmost edge point. After shearing, fewer smaples are wasted.*

keeping only the part of the gradient vector oriented perpendicular to $\tilde{p}_4^t$, we get a differential direction in which to rotate the optical axis. The vector $\tilde{p}_4^t \times (-\frac{\partial}{\partial \tilde{p}_4^t}(h))$ defines this axis of rotation. Our problem is now reduced to a 1D search. We rotate $\tilde{p}_4^t$ by various angles ranging between three and fifty degrees (with one negative angle for forced exploration) and choose the rotation that gives us the smallest objective $h(P)$ using the optimal affine parameters.

To compute the affine parameters in step 4bii we observe that our shadow map is rectangular and therefore compute the minimum area rectangle enclosing the 2d convex hull of points in the shadow map. The minimum area rectangle then defines the near image plane's extents of the light's frustum. It is well known that the minimum area rectangle will have an edge line up with an edge of the convex hull, so we need only loop over all convex hull edges to find the best fit [FS75]. Our method for computing affine skew is heuristic (see Figure 3): When looping over the convex hull edges for the minimum area rectangle algorithm, we check to see if the minimum and maximum extents in the current edge direction can be shifted to better line up with the edge's endpoints. Accounting for affine skew seems to provide minimal benefit in practice when compared to the other freedoms, so ignoring this adjustment can be done without much ill effect. Finally, to handle shadow maps that do not have aspect ratios of one, we make sure to map the longer side of the min-area rectangle to the dimension of the shadow map with more samples.

In step 5, to avoid temporal artifacts, we do not want to drastically change P from the previous frame's if it does not substantially improve the sampling quality. We consider a substantial improvement to be when the median value of $\sum_{i,j=1,2} \left(\frac{dx^i}{du^j}\right)^2$ taken over the screen is less than 80% from that measured using the previous frame's P. Of course the previous frame's P must include the necessary new scale and shift to ensure that the shadow map incudes all of the currently relevant samples.

2.4. Implementation Details

The Scene Optimized Shadow Mapping system was implemented on a computer equipped with a NVIDIA GeForce 7800 GT graphics card, PCI-Express bus, and AMD Athlon 64 X2 Dual Core 4400+ 2.21 GHz CPU. We made use of OpenGL extensions for framebuffer objects (for render to texture), pixel buffer objects (for asynchronous readback), and texture rectangles (for textures of various aspect ratios). Shaders were written in GLSL.

Though readbacks are typically considered too costly for real-time rendering, a couple key observations make the method viable. First, temporal coherency between frames suggests that we do not need to perform the full readback-and-optimize process each frame. Secondly modern hardware allows for asynchronous readbacks over the PCI-Express bus. Latency and transfer costs are measured to be on the order of 16-18ms. This means that depth and bit mask data can be updated at a peak rate of up to 55-62 fps. To take advantage of this, we only update P every three frames. In this case much of the readback time can occur concurrently with our frame rendering.

Another important note is that we only readback one depth buffer for the screen's view, independent of the number of lights. For the shadow bits, we only need one additional bit per pixel per light. In fact, due to hardware implementations, readbacks of up to 31 additional lights adds no extra cost. Hence the algorithm's additional costs (geometry pass in step 1, readback in step 2) are amortized over the number of lights.

Due to texture format readback and precision issues, step 1 actually uses multiple render targets to output both the bit mask and a depth buffers. Floating point depth values (transformed to [0,1] range) are packed into RGBA unsigned bytes and read back in BGRA format. For simplicity, the proof-of-concept code utilized only two ongoing asynchronous readbacks at any time. Further speedups can of course be gained by increasing the pipelining to match the fps needs of the application.

In step 1, the number of samples required depends on scene depth slope variation. For the scenes we tested (rendered at 1150x1150 resolution), a 180x180 sample resolution was sufficient. Every couple frames we generate standard shadow maps for step 1 instead of using the previous frame's. This ensures we capture all disjoint shadow components that may have moved into the scene after animation.

In step 4a we only use shadow boundary points where we can compute reliable partial derivatives (i.e. we must avoid screen samples near depth discontinuities). After reading back the z-buffer in step 2, we use this data to compute appromixate partial derivatives of z. Derivative continuity is checked by thresholding the acceleration in z. The presented framework is quite robust to various methods for performing this classification since discontinous samples are dominated by the more plentiful smooth ones.

In step 4a, we found three iterations of gradient descent were sufficient to reach "convergence." For each iteration

we test -10, 3, 5, 9, 16, 23, 30, 40, and 50 degree rotations. The algorithm is quite robust to choices here too as long as a range of values is covered (otherwise more iterations are required). The negative rotation amount is present only to force possible exploration away from local minima.

In step 4b, we must fit a bounding rectangle in the shadow map to the convex hull of the used samples. Due to the subsampling in step 1, this bounding rectangle may be too agressive, so we increase the rectangle size slightly.

3. Results

We compared our algorithm against a Normal shadow mapper, a Focused variant that adds a frustum limiting step, a Plane Optimal shadow mapper, and a LiSPSM shadow mapper. The frustum limiting step shrinks the field of view by projecting the view frustum onto the light frustum's near plane, intersecting the convex hull of these points against the light frustum's near plane extents, and finding a minimum rectangle about the intersection. Of the variants of PSM, we chose LiSPSM as it appeared to represent one of the more recent and well behaved representatives of this family.

We created a scene make up of a few dozen objects on or near a floor (1,137,772 triangles in total). We fixed the viewer to be standing slightly above the floor, as would be typical in a walkthrough. We placed the light at 500 randomly chosen positions uniformly on a distant hemisphere, as might be typical for a sunlit scene. See Figure(4) for an example configuration. The screen was rendered at 750^2 resolution. The shadow maps were rendered at 400^2 resolution. The shadow data in the output screen was compared to a "ground truth" rendering that used a focused shadow map with resolution 4000^2. For each rendering we computed the number of misclassified shadow pixels, as compared to the ground truth rendering.

We report the number of misclassifed pixels in table 1. In the first row we report the average misclassification number for each of the methods, using all 500 inputs.

We also wanted to better understand how each algorithm degrades. To do this, *for each method*, we ranked the scenes by the number of misclassified shadow pixels it produced. We then chose the worst 10% of the scenes according to this rank. For these 50 scenes we report the average number of misclassifed pixels found by all of the methods. This ranking and averaging was repeated for each of the five methods. These numbers are reported on the five remaining rows of the table.

For a visual sense of how these methods compare, refer to the video.

The following table gives the running time for the three algorithms for different resolution shadow maps and different number of lights. The screen resolution was 1150x1150.

Running Time (fps)				
# lights	Res.	Optimal	Focused	LiSPSM
1	500x500	56.7	67.3	69.3
1	700x700	55.1	65.3	66.7
1	1000x1000	51.9	60.7	62.4
2	500x500	39.3	44.4	45.7
2	700x700	36.7	41.7	43.0
2	1000x1000	33.8	37.9	39.4
3	500x500	29.2	32.5	33.3
3	700x700	27.1	30.2	31.5
3	1000x1000	24.8	27.1	28.3
4	500x500	22.8	25.2	25.9
4	700x700	21.1	23.2	24.2
4	1000x1000	19.4	21.2	22.0

Table 2: *Running time versus number of lights and shadow map resolution.*

We see from these tables that as the number of lights is increased, our algorithm becomes increasingly attractive. Not only do we get per light memory savings for the same shadow quality, but the discrepancy in frame rates also shrinks. For 4 lights, an optimized shadow map with resolution 1250x1250 gives similar quality to a normal shadow map of resolution 1750x1750 while sacrificing about 1 frame per second (7% of framerate) and saving 6 million shadow samples.

As an example of the timing profile for scenes in the experiments, a 4 lights and 750x750 shadow map resolution case is shown in the table below. The optimization timings include convex hull timings as well. Therefore, the optimization time for normal shadow mapping is simply the time it takes to do convex hull and intersection for limiting the frustum. In optimal shadow maps, the optimization time additionally includes CPU time spent running the actual optimization procedure. Render & wait includes time for issuing render calls and waiting for readback or waiting due to throttling from limited GPU pipeline depth.

Time Profile for a 4 light scene using 750x750 shadow maps				
	Optimal		Normal	
	Time (ms)	% total	Time (ms)	% total
Total scene	56.0	100	41.4	100
Optimization	6.96	12.43	0.16	0.40
Convex Hull	1.10	1.96	0.16	0.40
Render&Wait	49.03	87.57	41.23	99.60

Table 3: *Time costs for various stages of frame processing.*

4. Conclusion & Future Work

We have presented an optimization framework for addressing shadow map aliasing along with a gradient descent algorithm for finding solutions. This provides for robust

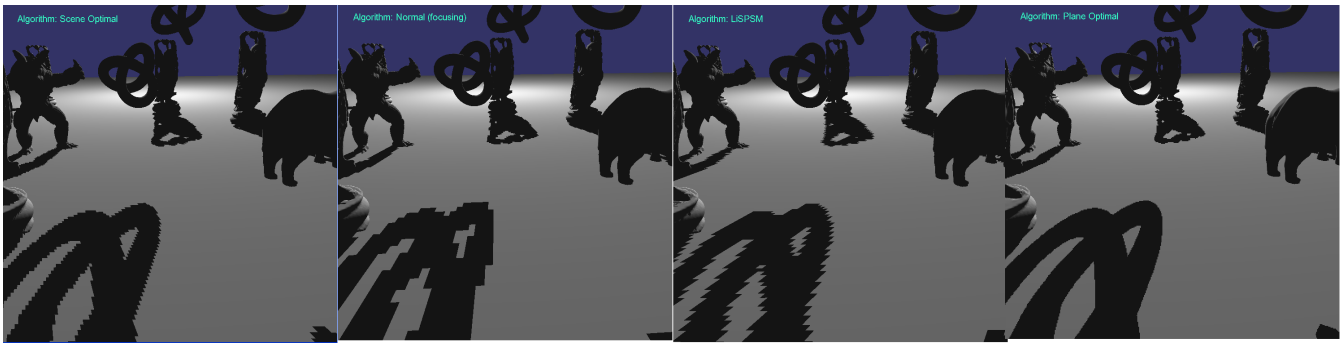

Figure 4: *From left to right: Optimal, Focused, LiSPSM, Plane.*

Number of Misclassified Pixels					
	Optimal	Focused (%More)	LiSPSM (%More)	Plane Opt. (%More)	Normal(%More)
Using all data	1886057 (0)	2407686 (27.65 %)	2929613 (55.33 %)	7828974 (315.09 %)	5405579 (186.60 %)
Worst 10%					
Opt	304839 (0)	403529 (32.37%)	324559 (6.47%)	500413 (64.16%)	1146830 (276.21%)
Foc	263638 (0)	485088 (84.00%)	354289 (34.38%)	373324 (41.60%)	1203575 (356.53%)
LiSPSM	253331 (0)	422387 (66.73%)	377762 (49.12%)	408887 (61.40%)	1068285 (321.70%)
Plane	170056 (0)	217869 (28.12%)	197196 (15.96%)	1242144 (630.43%)	692796 (307.39%)
Normal	272953 (0)	448302 (64.24%)	338566 (24.04%)	521885 (91.20%)	1308651 (379.44%)

Table 1: *Number of misclassified shadow pixels. Percentages are percent more error than optimized shadows.*

and quantifiable improvements in shadow quality. Real-time applications with free texture memory and only a single light need not use the method since the cost of read-back makes any benefit moot. However, as the number of lights is increased or as texture memory becomes more tightly constrained, optimal shadow mapping provides increasingly large payoffs. For offline rendering, the cost structure changes as manual labor must also be factored in as a resource. More experiments and user studies must be carried out to evaluate the method's precise utility in this domain.

References

[Arv04] ARVO J.: Tiled shadow maps. In *Proceedings of Computer Graphics International 2004* (2004), IEEE Computer Society, pp. 240–247.

[CD03] CHAN E., DURAND F.: Rendering fake soft shadows with smoothies. In *Proceedings of the Eurographics Symposium on Rendering* (2003), pp. 208–218.

[CD04] CHAN E., DURAND F.: An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering* (2004), pp. 185–195.

[CG04] CHONG H. Y., GORTLER S. J.: A lixel for every pixel. In *Proceedings of the Eurographics Symposium on Rendering* (2004).

[Cho03] CHONG H.: Real-time perspective optimal shadow maps. *Harvard University Senior Thesis* (April 2003).

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *SIGGRAPH '01* (2001), pp. 387–390.

[FS75] FREEMAN H., SHAPIRA R.: Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM 18*, 7 (1975), 409–413.

[Koz04] KOZLOV S.: Perspective shadow maps: Care and feeding. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics* (2004), Addison-Wesley Professional.

[LTYM06] LLOYD B., TUFT D., YOON S., MANOCHA D.: Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (2006), pp. 215–226.

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *SIGGRAPH '00* (2000), pp. 385–392.

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (2004), pp. 153–160.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *SIGGRAPH '87* (1987), pp. 283–291.

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. *ACM Trans. Graph. 22*, 3 (2003), 521–526.

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. In *SIGGRAPH '02* (2002), pp. 557–562.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *SIGGRAPH '78* (1978), pp. 270–274.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (2004).

Appendix A: Computing Required Quantities

We seek to solve our optimization problem (6) via Algorithm 1. Although we chose the L2 metric for implementation, what follows is completely general and can be applied to other metrics. In all cases, we must compute h and its gradient with respect to $\tilde{p}_4^t$. To compute h, we require the derivatives $\frac{dx^j}{du^i}$ and the rest is straightforward. To compute h's gradient, we need these same derivatives plus their gradients with respect to $\tilde{p}_4^t$.

To describe the computation of these terms, we will use the additional notion shown in the following table:

Notation	
$\vec{X} = [X^1, X^2, X^3]^t$	Light space coordinates
$w_f = \frac{w_p}{w_q}$	Homogeneous coordinate of shadow mapping PQ^{-1}
$\vec{q}_j$	First 3 entries in j-th column of matrix Q^{-1}, (j=1,2,3,4)

Table 4: *Additional notation for computations.*

While our interest is in computing the entries $\frac{dx^j}{du^i}$ of the inverse Jacobian, it is easier to first compute the entries $\frac{du^j}{dx^i}$ of the Jacobian and then invert this two by two matrix.

To perform the computation, we think of x^3 as being defined locally as a function of x^1 and x^2. We can safely ignore regions of non-smoothness in x^3 as a measure zero set in the $[x^1, x^2]^t$ domain. By definition we have (for $i, j = 1, 2$):

$$\frac{du^i}{dx^j} \equiv \left(\frac{\partial u^i}{\partial x^j} + \frac{\partial u^i}{\partial x^3} \frac{\partial x^3}{\partial x^j} \right) \tag{7}$$

The $\frac{\partial x^3}{\partial x^j}$ term is computable from the read back z-buffer values for each pixel and its neighbors. The product rule, re-arrangement of terms, and factorization into the notated quantities gives us (for $i = 1, 2; j = 1, 2, 3$):

$$\begin{aligned} \frac{\partial u^i}{\partial x^j} &= \frac{1}{w_f} \left[\frac{\partial (u^i w_f)}{\partial x^j} - u^i \frac{\partial w_f}{\partial x^j} \right] \\ &= \frac{w_q}{\tilde{p}_4^t \vec{X}} \left[\tilde{p}_i^t \vec{q}_j - \frac{\tilde{p}_i^t \vec{X}}{\tilde{p}_4^t \vec{X}} \tilde{p}_4^t \vec{q}_j \right] \end{aligned} \tag{8}$$

These relations allow us to compute (7).

The inverse function theorem then relates these quantities to the derivatives we really care about.

$$\begin{bmatrix} \frac{dx^1}{du^1} & \frac{dx^1}{du^2} \\ \frac{dx^2}{du^1} & \frac{dx^2}{du^2} \end{bmatrix} = \begin{bmatrix} \frac{du^1}{dx^1} & \frac{du^1}{dx^2} \\ \frac{du^2}{dx^1} & \frac{du^2}{dx^2} \end{bmatrix}^{-1}$$

We then make use of the explicit formula for the inverse of a 2x2 matrix to obtain:

$$h\left(\frac{dx^1}{du^1}, \frac{dx^1}{du^2}, \frac{dx^2}{du^1}, \frac{dx^2}{du^2} \right) = \sum_{\text{samples}} \frac{1}{(det\mathbf{J})^2} \left[\sum_{i,j=1,2} \left(\frac{du^i}{dx^j} \right)^2 \right] \tag{9}$$

$$\text{where} \quad \mathbf{J} = \begin{bmatrix} \frac{du^1}{dx^1} & \frac{du^1}{dx^2} \\ \frac{du^2}{dx^1} & \frac{du^2}{dx^2} \end{bmatrix}$$

This makes h fully computable from known quantities. To compute the gradient of h, we apply product rule to Equation 9. The result is an expression involving derivatives of the form in Equation 7 and their gradients. So the remaining ingredient for computing the gradient of h is:

$$\frac{\partial}{\partial \tilde{p}_4^t} \left(\frac{\partial u^i}{\partial x^j} \right) = \frac{w_q}{(\tilde{p}_4^t \vec{X})^2} \left[\left(\frac{2(\tilde{p}_i^t \vec{X})(\tilde{p}_4^t \vec{q}_j)}{\tilde{p}_4^t \vec{X}} - (\tilde{p}_i^t \vec{q}_j) \right) \vec{X} - (\tilde{p}_i^t \vec{X}) \vec{q}_j \right]$$