Eurographics Symposium on Rendering (2004)
A. Keller, H. W. Jensen (Editors)

A Lixel for every Pixel

Hamilton Y. Chong Steven J. Gortler

Harvard University

Abstract

Shadow mapping is a very useful tool for generating shadows in many real-time rendering settings and is even used in some off-line renderers. One of the difficulties when using a shadow map is obtaining a sufficiently dense sampling on shadowed surfaces to minimize shadow aliasing. Endlessly upping the light-image resolution is not always a viable option. In this paper we describe a shadow mapping technique that guarantees, that over a small number of chosen planes of interest *(such as a floor and a wall), the shadow map is, in fact, perfectly sampled, ie. for each pixel in the viewer camera, there will be exactly one lixel in the shadow map that samples the exact same geometric point.*

1. Introduction

Shadow mapping[Wil78] is a very useful tool for generating shadows in many real-time rendering settings and is even used in some off-line renderers. One of the difficulties when using a shadow map is obtaining a sufficiently dense sampling on shadowed surfaces to minimize *shadow aliasing*. Endlessly upping the light-image resolution is not always a viable option. There are interesting methods that minimize the effects of aliasing [RSC87] as well as more complicated shadowing algorithms that employ some aspects of the original shadow mapping algorithm (see [GLY*03] and references therein). This paper focuses on the sampling patterns of simple shadow mapping and can be considered orthogonally to these advances.

To use a shadow map, the point-light position in 3d, l, is specified. Depending on the light and its position in the scene, we may be interested in all of the light rays emanating from l, or we may know that we only need to account for some smaller solid angle (wedge) of rays. In either case the input does not specify the 4 by 4 matrix M_l that is used for mapping points from world coordinates to light-image coordinates.

In their very insightful paper, “Perspective Shadow Maps,” Stamminger and Drettakis [SD02] pointed out this inherent freedom in shadow mapping, and described a recipe for choosing a matrix that made more efficient use of the pixels in the light-image (henceforth *lixels*). The existence of this inherent freedom, an almost obvious fact in retrospect, seems to have been completely overlooked in the prior art. In the perspective shadow mapping method, the viewer camera information, specified by a 4 by 4 matrix M_v that maps points from world coordinates to viewer-image coordinates, is used as part of the process for determining M_l. In this setting, the shadow map is re-rendered whenever the view changes; this imposes no extra cost whenever one is rendering a scene with dynamically changing geometry.

Perspective shadow mapping tends to allocate lixels more densely in directions pointed toward the viewer’s position. As such, it can minimize the blockiness of shadows on objects seen up-close. However, except in very specific camera/light configurations, it is hard to predict the quality of the shadow sampling that results from their algorithm or say that it has any guaranteed properties. In fact, in our experience, it often performs significantly worse than conventional shadow mapping.

The question of how to *optimally* determine M_l given a particular viewer camera, a light position, and scene geometry is investigated in [Cho03]. Their solution involves first performing a rendering pass that computes not colors, but the derivative of the "viewer to world to light" mapping at each visible fragment. This data is then read from the frame buffer. From this data, a numerical optimization step can be run to find the optimal M_l. The expense of the readback suggests that optimal shadow mapping is not currently practical for real time rendering.

In this paper we describe a shadow mapping technique that guarantees that over a small number of chosen *planes of interest* (such as a floor and a wall) the shadow map is, in

fact, perfectly sampled, i.e. for each pixel in the viewer camera, there will be exactly one lixel in the shadow map that samples the exact same geometric point. Our basic construction is borrowed from the "plane-stabilization" technique used in computer vision to stabilize the view of a plane that is imaged in a moving camera (e.g. [IA98]). On other parts of the geometry, we have observed that our shadow sampling is typically not worse than that obtained by normal shadow mapping or perspective shadow mapping.

Computing our light matrices is done by solving a very small linear system and is essentially free. But we do incur some additional costs. First, the environment's author must choose the desired planes of interest. Second, the method must compute one shadow map (requiring a geometry pass) for each plane of interest. This is a significant expense when vertex processing is the bottleneck. Thirdly, we are constrained by hardware to use rendering-frusta and shadow-maps of rectangular shape. As a result, there is typically some overlap in the shadow maps used for the multiple planes, and we must allocate "wasted" lixels that are not used in the rendering. Fourthly, even when there is only one plane of interest, our method may require two shadow maps in order to cover the whole scene. Finally, because our method is indeed not optimal over the whole scene, there may still be shadows that are significantly aliased.

2. Method

Here we describe the basic matrix construction employed by our shadow mapping technique. We build up the construction incrementally.

Basic Idea For each frame we are given the 4 by 4 view camera matrix M_v that transforms points from world to viewer-image coordinates

$$[x_v q, y_v q, z_v q, q]^t = M_v [x_w, y_w, z_w, 1]^t$$

where we divide by q to obtain the viewer-image coordinate $[x_v, y_v, z_v, 1]^t$. We are also given the light position l, described in world coordinates by $[x_w^l, y_w^l, z_w^l, 1]^t$, and a geometric plane of interest P.

Our job is to come up with the matrix M_l that transforms points from world to light-image coordinates.

$$[x_l q, y_l q, z_l q, q]^t = M_l \ [x_w, y_w, z_w, 1]^t$$

It is well known [Fau93] that a 4 by 4 projective matrix can be uniquely specified (up to scale) by constraining the pre and post transform coordinates of 5 geometric points, (with no 4 co-planar). Given these points, the matrix can be determined by solving two appropriate 4 by 4 linear systems (see [Fau93] for details). Because M_l images from the point l, we immediately have the constraint

$$[0,0,1,0]^t = M_l \ [x_w^l, y_w^l, z_w^l, 1]^t \qquad (1)$$

To specify the rest of the matrix, We first pick 4 pixels given

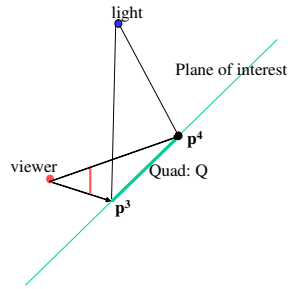

Figure 1: *Rays are shot through 4 pixels of the viewer-image. These rays are intersected with the plane of interest P obtaining the quad Q. The light matrix is constrained to sample the quad identically as the viewer-image did.*

in viewer-image coordinates as

$$\begin{bmatrix} 0 \\ 0 \\ * \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ h-1 \\ * \\ 1 \end{bmatrix}, \begin{bmatrix} g-1 \\ h-1 \\ * \\ 1 \end{bmatrix}, \begin{bmatrix} g-1 \\ 0 \\ * \\ 1 \end{bmatrix} \qquad (2)$$

where g (resp. h) is the width (resp. height) of the viewer image measured in pixels. The * in the third slot denotes that we do not care about the z value.

We then shoot rays from the viewer position through these pixels, hitting the plane P in four points p^i (with $i = [1..4]$) denoted in world coordinates by

$$[x_w^i, y_w^i, z_w^i, 1]^t$$

These 4 points bound a quad Q on P that exactly fills the viewer-image's sampling rectangle (see Figure 1). We now add the following four more constraints on our M_l:

$$[0,0,1q,q]^t = M_l \ [x_w^1, y_w^1, z_w^1, 1]^t \qquad (3)$$

$$[0,(h-1)q,1q,q]^t = M_l \ [x_w^2, y_w^2, z_w^2, 1]^t \qquad (4)$$

$$[(g-1)q,(h-1)q,1q,q]^t = M_l \ [x_w^3, y_w^3, z_w^3, 1]^t \qquad (5)$$

$$[(g-1)q,0,1q,q]^t = M_l \ [x_w^4, y_w^4, z_w^4, 1]^t \qquad (6)$$

Which states that the four p^i should have the same viewer-image and light-image x,y coordinates. The '1' in the third slot of the light-image coordinates has been chosen arbitrarily for now. Equations (1) and (3-6) give us 5 point constraints on M_l. (Note that these 4 points on P are obviously co-planar; we will fix this up before we are done).

We have four points on a plane (no three collinear) that have the same image (x,y) coordinates in both the viewer and light images, therefore *every* point on the plane must have the same (x,y) coordinates in both images. This is because the mapping between two pin-hole images of a plane must be a 2d projective transformation (3 by 3 matrix), which is fully determined by the mapping of four points [Fau93]. (The identity map is the only 2d projective map that leaves 4 generic points unchanged).

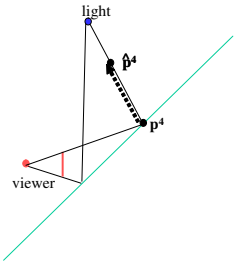

Figure 2: *To obtain non-coplanar points, one of p^4 is raised up to $\hat{p}^4$.*

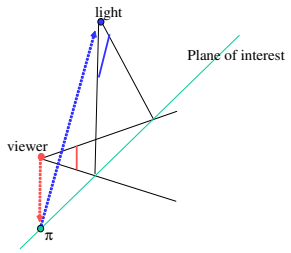

Figure 3: *Geometric intuition behind the light camera determined by our method. From the viewer's position, draw a line (dotted red) parallel to the viewer's image plane (solid red) until it hits the line of interest at some point π. Now draw a line (dotted blue) from π to the light position. The light-image plane orientation (solid blue) is chosen to be parallel to the dotted blue line.*

Note that to achieve proper clipping in OpenGL, M_l, must be factored into $'M_l = GLV \;\; GLP'$ an appropriate GL viewport and canonical projection matrix.

Finally, the quad on the plane may actually straddle across infinity (see Figure 5). This does not affect the construction of M_l. But it is clear from this case, that one must choose the "sign" of $' \pm GLP'$, determining the "direction" of the light frustum, such that geometric points in the desired direction have positive q values and are not clipped during rendering.

Co-planarity and light z-values As mentioned above the 4 p^i are co-planar and hence cannot be used to fully constrain M_l. This can be easily fixed up as follows (see Figure 2). We compute a ray r^4 emanating from the light position l through the geometric point p^4. We then chose any generic point $\hat{p}^4$ along the r^4. $\hat{p}^4$ is described in world coordinates as $[\hat{x}^4_w, \hat{y}^4_w, \hat{z}^4_w, 1]^t$. Finally equations(3-6) are replaced by

$$[0,0,1q,q]^t = M_l \;\; [x^1_w, y^1_w, z^1_w, 1]^t \qquad (7)$$

$$[0,(h-1)q,1q,q]^t = M_l \;\; [x^2_w, y^2_w, z^2_w, 1]^t \qquad (8)$$

$$[(g-1)q,(h-1)q,1q,q]^t = M_l \;\; [x^3_w, y^3_w, z^3_w, 1]^t \qquad (9)$$

$$[(g-1)q,0,-1q,q]^t = M_l \;\; [\hat{x}^4_w, \hat{y}^4_w, \hat{z}^4_w, 1]^t \qquad (10)$$

Where the -1 (resp. $+1$) in the third slot of the light-image coordinates represents the near (resp. far) plane†. We can now use Equations (1) and (7-10) to fully constrain the matrix M_l. It is important to note that with this construction, we still will exactly satisfy Equations (3-6). In particular, p^4 will still have the same viewer and light image (x,y) coordinates.

Geometric Intuition Matrices aside, it is useful to have a geometric interpretation of our construction. We will start with a description in flatland (see Figure 3). We are given the viewer camera, a line of interest and light position. We are also given the desired fov of the light-camera. We want to solve for the light-camera, which is specified by a 3 by 3 matrix up to an arbitrary scale factor (8 dof). If we ignore the depth values in the light-image for now, we are left with a 2 by 3 matrix (5 dof remaining). The position of the light in flatland locks down 2 more dof (3 dof remaining). The (affine) viewport in the eventual light-image specifies a scale and translation. The one remaining degree of freedom, which is our only non-affine remaining dof, can be simply thought of as the orientation (angle) chosen for the light-image plane.

Let us choose the light-image plane angle using the geometric construction of Figure 3 (see electronic version for colors). From the viewer's position, draw a line (dotted red) parallel to the viewer's image plane (solid red) until it hits the line of interest at some point π. Now draw a line (dotted blue) from π to the light position. Choose the light-image plane orientation (solid blue) to be parallel to the dotted blue line.

It is easy to see that with this configuration, the 1D mapping between the two cameras (ignoring depth values), applied only to points on the line of interest must be affine. Formally, if a point on the line of interest has coordinates $[x_w, z_w, 1]^t$, and we have the light-image and viewer image coordinates given by

$$[x_l q, z_l q, 1q]^t = M_l [x_w, z_w, 1]^t$$
$$[x_v q, z_v q, 1q]^t = M_v [x_w, z_w, 1]^t$$

then the transformation of the image coordinate of these points, induced by

$$[x_l q, z_l(x_l) q, q]^t = M_l M_v^{-1} [x_v, z_v(x_v), 1]^t$$

(where the z's are the appropriate depths of points on the plane) can be expressed as $[x_l, 1]^t = A[x_v, 1]^t$,where A is a matrix of the form $\begin{bmatrix} s_x & t_x \\ 0 & 1 \end{bmatrix}$

This can be proven from the following three facts: 1) the point π has infinite coordinates in both light-image and viewer-image coordinates. 2) the mapping between two flat-

† The choices of ± 1 for the light z-coordinates is arbitrary. Other choices will result in some affine transform of the z-values in post projective light coordinates. We do not explore optimization of these dofs in this work.

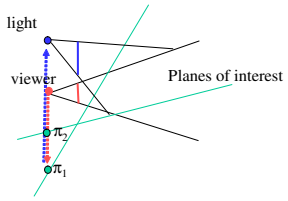

Figure 4: *In the miner's lamp configuration, the same light-image plane orientation is chosen independently of the plane of interest.*

land cameras of points constrained to lie on a line is a 1D projective transform. 3) any projective transform that does not map infinite points to finite points must be affine [Fau93].

Therefore, by an appropriate choice of the affine coordinates for the light-image (scale and translation) we can force the mapping A to be the identity. This (unique) combination of light image orientation and the appropriate affine coordinates in the light-image results in the M_l obtained by the constraints of Equations (1) and (7-10).

It should be clear (and it is easily proven formally [Cho03]) that for an arbitrary view-camera and light position, two arbitrary planes of interest will require different light-image orientations; otherwise the affinity of A will not be maintained. There is one special exception, the miner's lamp case (see Figure 4). In this case, regardless of how we choose the plane of interest, the geometric construction results in a light-image orientation parallel to that of the viewer. (Note that this kind of pair of cameras with parallel image planes is equivalent to the rectified camera configuration typically used in stereo vision [Fau93]). In this case, an appropriately chosen for M_l will give us an affine mapping A_i for each plane of interest P_i. Of course at most one of the A_i can be the identity map, other planes will be over/under-sampled.

It can also be shown[Cho03], that in the miner's lamp case, the perspective shadow mapping method of [SD02], implicitly chooses the orientation of rectified stereo images. When one is not in the miner configuration, it is harder to give any concrete statements about the behavior of [SD02]. In contrast, our algorithm results in an affine (in fact the identity) mapping on the specific plane of interest for any camera and light configuration.

In the 3d case, we are instead solving for the 4 by 4 matrix (up to scale) of the light camera (15 dof). If we ignore the depth coordinates, we are left determining a 3 by 4 matrix (11 dof). The position of the light fixes 3 more dofs (8 dofs remaining). In this case, there are 6 affine degrees of freedom on the image plane: scale, aspect ratio, 2 translations, one shear, and one image twist. The remaining two degrees of freedom can be thought of as two angles determining the orientation of the image plane in space. The proper choice

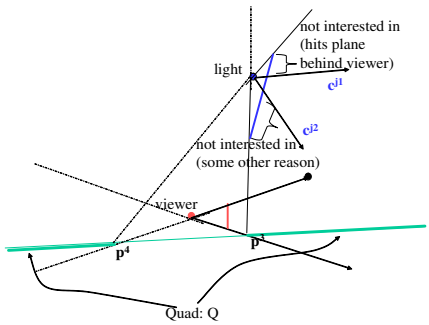

Figure 5: *The construction works even in Q spans infinity. We must be careful to choose the sign of the light matrix to specify the "forward" direction. The c^j describe what the range of rays we wish to have in our shadow map*

.

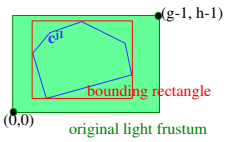

Figure 6: *As seen from the light's image. The image of C may include less of Q. We form the bounding box and change the matrix to only include the needed rays.*

for this orientation is again completely determined π, which in this case is a line and not a point.

Less of the plane Thus far, we have assumed that we want Q to exactly fill the light-image's sampling rectangle. But it may be the case that the viewer is not interested in all of Q (see figure 5) . For example part of Q, may be known to be occluded. Part may be closer to the viewer than its near plane. If the vanishing line of the plane is visible in the viewer camera, then part of Q will be beyond infinity (ie. behind the viewer). If our light is a "spot-light," then some part of Q may be known to be un-illuminated. In all of these cases, we may want to compute shadows only on a subset of Q, and as a result, we need fewer lixels than pixels.

So besides M_v, l and P, we also assume that we are given C, a set of directions for which we want light rays. C is given as a spherical polygon, with vertices c^j. We require that all of the rays of C lie within the light frustum of M_l as originally computed above (see Figure 5 and 6).

We then alter the computed M_l as follows. We compute the light-image coordinates for each of the vertices c^j, denoted by $[x_l^j, y_l^j, z_l^j, 1]^t$. We then compute the axis aligned rectangle in the light-image that has integer (pixel sample position) coordinate corners and encloses the $[x_l^j, y_l^j, *, 1]^t$ (see Figure 6). We can then create the 2d translation matrix A_l that renumbers the lower-leftmost pixel as $(0,0)$ and, from now on, use the matrix $M_l := A_l M_l$ instead. Now we

only need to allocate a sufficient number of lixels to cover the bounding box, and clip accordingly (using the appropriate viewport/projection factorization).

More generally, we may have a situation where we have decided not to allocate one lixel per pixel. In this case, one needs to include the appropriate 2d scale factor in the matrix A_l . This will make the lixels hitting P appear as axis aligned rectangles of finite size in the viewer-image. In addition, we may wish to include a 2d rotation (twist) in A_l, so that the light-image bounding rectangle more tightly bounds C. This can give us more efficient use of the samples; but in this case the lixels hitting P will appear as *rotated* rectangles in the viewer-image [‡].

3. Implementation

We built an implementation of our shadow mapper to handle a few planes of interest in some specific configurations. Each shadow map was allocated a fixed number of lixels (reallocation is expensive or complicated). A pixel shader was used to determine which shadow map to use for each fragment. Here we describe some of the details.

The extent of the shadow-casting scene geometry was bounded by a box. We assumed (for ease of implementation) that the light had an inherently limited fov of less than a hemisphere, specified as a bounding rectangle with respect to some appropriate geometric plane G. We implemented the "extent" computation described in [SD02], which defines the 3D hull H bounding the part of the scene that light-image needs to cover. The rays from l bounding H define C^{tot}. With our "less than hemisphere" assumption, C^{tot}, can be represented as a convex polygon on G. In addition, all of the code to compute the 3d extent-hull can be implemented using only 2D convex-hull and convex set intersection on G.

In one of our environments we chose the "wall" and "floor" as planes of interest. We defined C^{floor} as the set of rays with the following properties: they are a subset of C^{tot}, they hit the floor plane "below" the floor/wall intesection line, and they hit the floor plane on a point within the viewer's frustum. C^{floor} is computed using 2d calculations on the floor and is projected onto G. C^{wall} is defined similarly, but restricted to rays hitting the wall "above" the floor/wall intersection line. A third conventional shadow map was used to cover $C^{\text{fallback}} = HULL(C^{\text{tot}} - C^{\text{wall}} - C^{\text{floor}})$.

We included a 2d rotation (twist) in the A_{floor} matrix so that floor/wall intersection line would project as a horizontal line in the floor's shadow map. And any point "above" this line would project out of bounds in the floor's shadow map.

In the pixel shader, we always first attempted to use the floor shadow map. If the pixel was out of its bounds, we would then attempt to use the wall shadow map. If the pixel was out of these bounds, we would use the conventional fallback shadow map. The fallback shadow map was rarely hit, and so we allocated to it a smaller number of lixels.

We also experimented with a table/floor configuration. In this case C^{floor} was not convex since it did not include, rays that hit the table top. For this configuration, we would first attempt to use the table's shadow map. We used four explicit point line tests in the pixel shader to determine if a ray missed the interior of the tabletop polygon. If the ray missed the table top, we would then attempt to use the floor's shadow map.

4. Results

We implemented our algorithm, as well as the perspective shadow map algorithm as described in [SD02] and the paper's web page. We also implemented a conventional shadow map. We ran all of the algorithms on a scene with a floor, wall and a number of bouncing bunnies. We used the same computation (described in [SD02]), to determine C^{tot} for all of the algorithms. Our algorithm used 3 shadow maps, while the conventional and perspective shadow maps used 1. This required us to do an extra two geometry passes. Our scenes had fewer than $100,000$ polygons. As we were fill-rate limited, these passes did not significantly impact performance. Our shadow map was implemented in a pixel shader. During the rendering of each view-image pixel, the shader evaluates a set of conditionals to determine which shadow map to use. We were easily able to run these pixel shaders using 1024 by 1024-pixel view and light images at 30fps.

In our experiments, we allocated the same total number of lixels for each of the three algorithms. For the screen-shots we gave the conventional and perspective shadow maps a buffer of 512 by 512 lixels. We gave our algorithm two buffers of 350 by 350 and one buffer of 128 by 128. The view image was given 1024 by 1024 pixels.

To compare of the results, (see Figure 7) we did not use any shadow antialiasing methods so that the actual sampling behavior was not obscured. For many views, all three performed comparably. When the light was moved to make the shadows elongated, the conventional shadow map became quite pixelated. The perspective shadow map occasionally did better than the conventional algorithm. When the light moves in front of the view camera, the perspective shadow maps often deteriorated badly[§]. Note, that the conventional

[‡] We have also experimented with allowing the use of a set of rays C that includes more of the plane P than the quad Q. One must be very careful in this case. If the set C includes rays that approach the line π (see figure 3), then size of the projected bounding box in the light's view approaches infinity, and almost all of the lixels will be allocated to regions outside of Q.

[§] This seems to be because C contains rays that cover much of the view-camera's near plane. In this case, C expressed in the view-cameras post perspective space, has a huge field of view. When the perspective shadow map is created with such a large field of view, most of the lixels are dedicated to its extremes directions where they are needed least.

Figure 7: *Left: conventional (aka normal) shadow map. Middle: Perspective shadow map. Right our (aka plane optimal) shadow map. Zoom up in viewer to observe details.*

shadow map algorithm used here includes the hull computation described in [SD02], which may account for its competitiveness with the perspective shadow map algorithm.

Our method reliably gave very high resolution shadows for most configurations, even on objects with geometry different from the planes of interest (e.g., the sphere seen in the figures). But there were some cases where we did perform worse than conventional shadow mapping. One such case was when the view camera was very close to the (optimized) floor and then looked at shadows on some (not optimized) plane perpendicular to the floor.

5. Conclusion

Inspired by the insights described in [SD02], we have presented a method to obtain high resolution shadows on a small number of "planes of interest." This is a very frequent case encountered in real-time 3d applications. Our algorithm would not be optimal in a much more "free-form" environment, where the important shadow receivers cannot be described in this simple way. In this case, one could certainly imagine various hybrid approaches.

Acknowledgments We would like to thank Leonard McMillan, Peter-Pike Sloan, and Danil Kirsanov, who contributed many central ideas to this work.

References

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

[Fau93] FAUGERAS O.: *Three-dimensional computer vision: a geometric viewpoint*. MIT Press, 1993.

[GLY*03] GOVINDARAJU N., LLOYD B., YOON S., SUD A., MANOCHA D.: Interactive shadow generation in complex environments. *SIGGRAPH* (2003).

[IA98] IRANI M., ANANDAN P.: A unified approach to moving object detection in 2d and 3d scenes. *IEEE PAMI* (1998).

[RSC87] REEVES W., SALESIN D., COOK R.: Rendering antialiased shadows with depth maps. *SIGGRAPH* (1987).

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. *SIGGRAPH* (2002).

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH* (1978).