Interpolation-Friendly Soft Shadow Maps

Orion Sky Lawlor*
Department of Computer Science, University of Alaska at Fairbanks

Abstract

We present Penumbra Limit Maps, a technique for extracting soft shadows from an modified shadow map. The shadow representation used by our method has excellent interpolation properties, allowing the shadow boundary to be rendered with sub shadowmap-pixel accuracy, which partially mitigates the resolution problems common to shadow map methods. Unlike similar shadow map methods, our method includes both inner and outer penumbrae, and is in fact physically correct for the simplest case of a straight object edge and infinitely distant extended light source. At object corners or where multiple object edges overlap, our method is no longer physically exact, but still gives plausible results. Finally, we show the method can be implemented naturally and efficiently on programmable graphics hardware.

Keywords: soft shadows, shadow map, GPU.

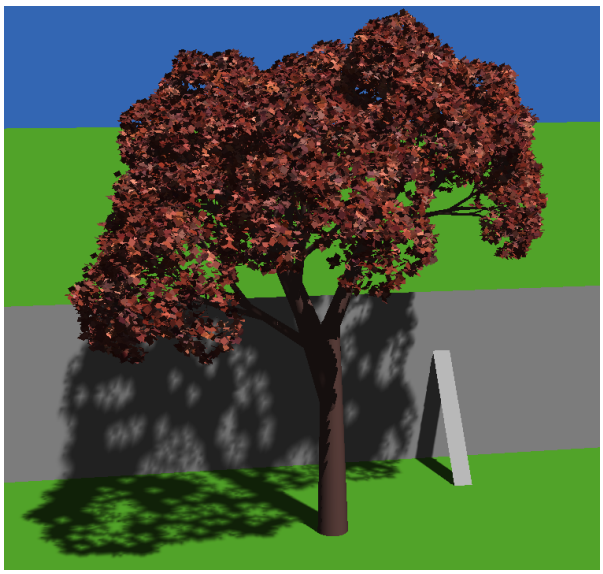

Figure 1. A 59K-polygon scene with soft shadows rendered via a 256x256 penumbra limit map on graphics hardware.

1 Introduction

Shadows are a key visual cue that allows viewers to better understand the geometry of a rendered scene. Direct light, such as sunlight, is often approximated by a point light source, which casts a hard-edged shadow. But in reality an area light source casts a soft shadow, with a fully black umbra surrounded by the partially bright penumbra. This penumbra region, which expands and softens with increasing distance from the object, provides an important distance cue. Even for a point light source, soft shadows can be used to inexpensively antialias hard shadow boundaries.

A simple scene consisting of 59,000 polygons rendered in realtime using penumbra limit shadows is shown in Figure 1. For this image, the shadow map was prepared in software at 256x256 resolution, which took 77ms using a depth image propagation technique. The scene can then be rendered on graphics hardware at 23fps. Without shadows the same scene renders at 26fps.

1.1 Related Shadow Work

There are three main classes of techniques to compute shadows in computer graphics. Raytracing can trivially determine if a light source is visible from a point and can hence determine extremely accurate shadow boundaries. *Shadow volumes* are a screen-space technique which extrudes, rasterizes, and counts signed object silhouette crossings for each pixel—visible surfaces that lie within a nonzero number of silhouettes are in shadow. Finally, *shadow maps* [17] rasterize the scene from the light source's point of view, and store the depth to the first occluder—objects beneath the first occluder are in shadow. Broadly, raytracing is difficult to implement efficiently, shadow volumes become fillrate-bound for objects with complicated silhouettes such as trees, and shadow maps display aliasing and resolution problems.

*Presenting author. Author email: olawlor@acm.org. Author postal address: 2380 Steese Hwy, Fairbanks, Alaska 99712-1701. Phone 1 907 474-7678 or fax 1 907 474-5030. Submitted to the 2006 International Conference on Computer Graphics & Virtual Reality (CGVR'06).

Figure 2 shows a cross section of an occluder edge casting a soft shadow. The "inner penumbra" is the portion of shadow where part but less than half of the light source is visible, while the "outer penumbra" is the portion of shadow where more than half of the light source is visible, but not all.

The literature abounds with interesting methods for computing soft shadows, the oldest of which are methods derived for global illumination [13]. Raster-type soft shadow methods range from Reeves' venerable shadowmap filtering [15] to recent work on GPU micro-occluders [3]. Other approaches include lighting convolutions evaluated analytically [12] or in the frequency domain [16]. Screen space soft shadow approaches include the Arvo et al flood fill [1].

There exist a variety of methods, including as the "penumbra wedge" soft shadow volume technique of Assarsson [2], that can accurately compute physical penumbrae. However, few of these techniques are affordable at interactive rates, especially for objects with high silhouette complexity such as foliage, or for multiple light sources.

A good survey of existing soft shadow modifications to shadow map techniques is presented by Hasenfratz et al. [8]. Many methods exist which compute soft shadows as a combination of several hard shadows, which is equivalent to the raytracing technique of approximating an extended light source as a collection of point light sources. The aliasing caused by point sampling results in quantized or noisy output, which requires as many as thousands of samples to average away.

A widely cited technical report by Parker et al [14] introduced an inexpensive and now-common technique for creating soft shadows, by pasting on fictitious geometry to make the shadow boundary of all objects fuzzy. Haines' Shadow Plateaus [7] use a similar silhouette meshing technique to generate physically correct shadows along a fixed planar receiver.

Brabec and Siedel [4] show a CPU-based "nearest occluder" search process based on a standard depth map for computing soft shadows. Recently de Boer [6] extended this approach to the GPU, including both inner and outer penumbra. Much shadow map work can be seen as a way to precompute the result of this slow nearest occluder search.

Our work is most similar to the inner penumbra "single-sample shadow map" technique of Kirsch and Döllner [9], who like us precompute a modified shadow map and evaluate the shadow using a fragment program. Chan and Durand's outer penumbra "smoothies" technique [5] also begins with the usual shadow depth map, but like our technique adds additional buffers to store the distance to the soft shadow. Wyman and Hansen's (outer) penumbra maps [18] are similar. None of these techniques handle both inner and outer penumbrae, and as we show in Section 4 Penumbra Limit Maps are capable of much smoother results at low shadowmap resolutions.

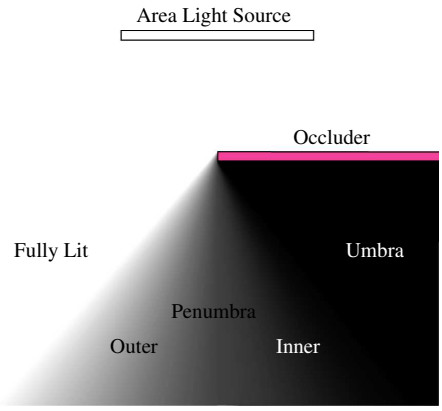

Figure 2. Cross-section of a simple soft shadow scene that can be exactly represented by one Penumbra Limit Map.

2 Penumbra Limit Shadows

In this paper, we present a soft shadow representation in the style of shadow maps. For each light source, the shadow map is filled with the shadow casting geometry of the scene, rendered with the camera at the center of each light source. During rendering, each light source's shadow map is queried to determine the illumination at a pixel.

We first derive the illumination for the simplest 2D case: an infinitely distant light source with a given angular extent, and a half-infinite occluder with edge lying perpendicular to the incoming light. Our shadowmap parameterization runs along parallel lines pointing directly toward the light source, with increasing depth measured downward and away from the light.

As shown in Figure 3 (A), we compute and store two depths for each shadow map pixel. The first depth, c, is the depth of the occluder corner that casts the shadow. This value only changes when switching between soft shadows cast by different objects or different parts of the same object. The second depth we store, p, is the penumbra limit depth. In the outer penumbra, which we consider first, p is the actual depth to the start of the penumbra, and is hence always less than c and decreases as we move away from the occluder.

When rendering, we want to find the fraction of the

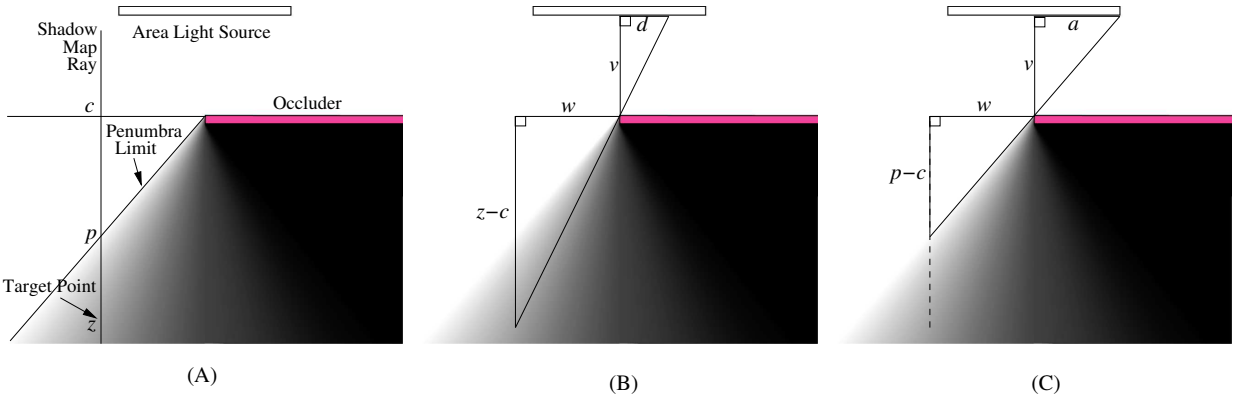

Figure 3. Derivation of penumbra limit equations for an infinitely distant area light source. Similar triangles relate $\frac{d}{a}$ to $\frac{z-c}{p-c}$

light source visible from some target point. Call the depth of that point z. As shown in Figure 3 (B), two similar right triangles are formed by the target point and occluder corner, and the occluder corner and light source hit point. The light source is infinitely distant, but we can imagine a scaled-down version hovering a distance v directly above our edge with visible length d. For the moment consider only the right half of the light source, and take the horizontal target/occluder distance as w. Then by similar triangles

$$\frac{w}{z-c} = \frac{d}{v}$$

As shown in Figure 3 (C), another two similar right triangles are formed by the penumbra limit and occluder corner, and the occluder corner and light source corner. Taking the right-half light source length as a, again by similar triangles

$$\frac{p-c}{w} = \frac{v}{a}$$

Now multiplying the two above equations,

$$\frac{p-c}{z-c} = \frac{d}{a}$$

Note that w and v cancel, leaving $\frac{d}{a}$ on the right hand side. This is the fraction of the right half of the light source that is visible. Thus for the outer penumbra, the total fraction l of the entire light source that is visible from the target point is just

$$l = \frac{1}{2} + \frac{1}{2}\frac{d}{a} = \frac{1}{2} + \frac{1}{2}\frac{p-c}{z-c}$$

Now consider the inner penumbra. By mirror symmetry, now $\frac{d}{a}$ gives the fraction of the left half of the light source that is not visible. This means in the inner penumbra, there's just a sign change to

$$l = \frac{1}{2} - \frac{1}{2}\frac{p-c}{z-c}$$

To uniformly support inner and outer penumbrae, we can simply change our definition of p in the inner penumbra. We choose to define a signed distance $p_i = -(p-c)$ in the inner penumbra, and $p_i = +(p-c)$ in the outer penumbra. Then the fraction of the light source visible in both the inner and outer penumbrae is exactly represented by

$$l = \frac{1}{2} + \frac{1}{2}\frac{p_i}{z-c} \tag{1}$$

Hence overall we store and interpolate p_i and c in the shadow map, and when rendering geometry evaluate Equation 1 to find the amount of light reaching a rendered pixel at depth z.

Our rather strange definition for the penumbra limit surface p_i has a number of benefits. First, it means we need not store or test whether each rendered pixel is is in the inner or outer penumbra. Second, because the surface p around one occluder edge forms an inverted "V" centered on c, it has a sharp crease which is difficult to sample and interpolate properly. However, the surface p_i around a straight occluder edge is simply a plane. This makes bilinear interpolation in p_i exact, and hence works well even at very low shadowmap resolutions, as we quantitatively evaluate in Section 4.

The largest benefit of this definition of p_i is that the midpoint of the shadow, $l = \frac{1}{2}$, lies along the zero-crossing line of the plane p_i. This means unlike with other shadow representations, with penumbra limit shadows the shadow cast by a smooth surface can be located very precisely–much more precisely than a shadowmap pixel!

2.1 Upper and Lower Limits

The light source visibility can only be computed via the soft shadow equations above when the target point is ac-

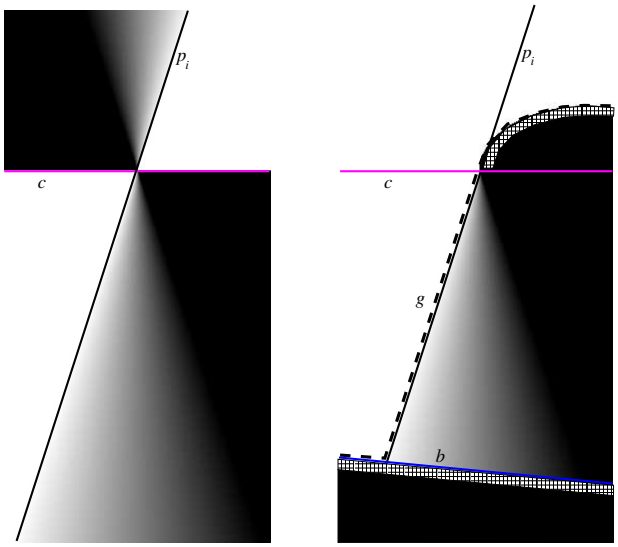

Figure 4. Light source visibility l at left (A) and geometry-clamped visibility l_g at right (B) for occluder depth c (magenta line), geometry upper limit g (dashed line), lower limit b (blue line), and penumbra limit depth p_i (black line). Light coming from above creates a soft shadow from a cylindrical occluder on the right.

```
TEMP shadowmap, L, cmp, Lg, illum;
ATTRIB shadowcoord=fragment.texcoord[1];
TEX shadowmap, shadowcoord, texture[1], 2D;
MAD L.b, shadowmap.a, 2.0, -1.0; # Unpack (1/2)p_i
SUB L.a, shadowcoord.z, shadowmap.r; # z - c
RCP L.a, L.a; # L.a = 1.0/(z - c)
MAD_SAT L, L.b, L.a, 0.5; # l from Equation 1
SLT cmp, shadowcoord.z, shadowmap; # Compare z
DPH cmp, cmp, {-2,4,1,-1}; # Three "if" statements
ADD_SAT Lg, L, cmp; # l_g from Equation 2
TEX illum, Lg, texture[2], 1D; # Illumination table
```

Figure 5. An OpenGL `ARB_fragment_program` to compute illumination using a Penumbra Limit Map. texture[1] is the penumbra limit shadow map as generated in Section 3; texture[2] is the visibility-to-illumination table as computed in Appendix A.

tually in soft shadow. Luckily, any time the target point is below the occluder depth c, Equation 1 will give the correct sign for the light visibility l, so graphics hardware can compute l correctly via the graphics card's cheap saturating arithmetic—that is, we compute l normally, then clamp to the range $[0, 1]$.

But for target points above the occluder depth c, Equation 1 gives nonsensical results—note the top of Figure 4 (A). Hence above c, we need to clamp the visibility to 0 (black) if shadowed by the geometry or to 1 (white) when above the geometry; similarly, when below deeper geometry, we must clamp to black. To do this, we store the uppermost geometry (or soft shadow) depth g and lowermost soft-shadow depth b in the shadow map, then compare the target depth z with both values and set the light visibility appropriately, which results in Figure 4 (B). So overall the fraction of the light source we can see is:

$$l_g = \begin{cases} 1 & \text{if } z < g, \\ 0 & \text{else if } z < c, \\ 0 & \text{else if } z > b, \\ \text{sat}(\frac{1}{2} + \frac{1}{2}\frac{p_i}{z-c}) & \text{otherwise.} \end{cases}$$

Since graphics hardware has poor branching performance, we implement this function using one set-if-less-than instruction to compare z with c, g, and b, then scale the compared values using a single dot product. Because saturating limits the input and output range to $[0, 1]$, the following is always equivalent to the l_g above:

$$l_g = \text{sat}(\text{sat}(\frac{1}{2} + \frac{1}{2}\frac{p_i}{z-c}) + 4(z < g) + \\ - 2(z < c) + (z < b) - 1) \quad (2)$$

The visible fraction of the light source l_g is converted to an actual illumination value using a visibility-to-illumination lookup table as described in Appendix A.

2.2 Graphics Hardware Implementation

We store the occluder depth c in the red channel, geometry upper limit g in the green channel, geometry lower limit b in the blue channel, and $\frac{1}{4}p_i + \frac{1}{2}$ in the alpha channel of the penumbra limit shadow map. Since all four quantities are just unsigned depths, we can afford to use even standard 8-bit per component RGBA textures, although 16-bit integer or float textures have better range and precision. Since all three quantities vary smoothly over the shadow map, we get excellent results by sampling the shadow map texture using ordinary bilinear interpolation.

To render geometry using penumbra limit illumination, we begin by computing the shadow map coordinates and target depth z in a vertex shader. Then in a pixel shader we evaluate Equation 2 to compute l_g, the fraction of the light source that is visible. Once the shadow map is created, like most shadow map techniques penumbra limit shadows create zero additional geometry—the rendering cost is purely arithmetic and texturing in the pixel shader. Our implementation in Figure 5 samples the shadow map

Figure 6. Scene lit from above by a striped colored light source.

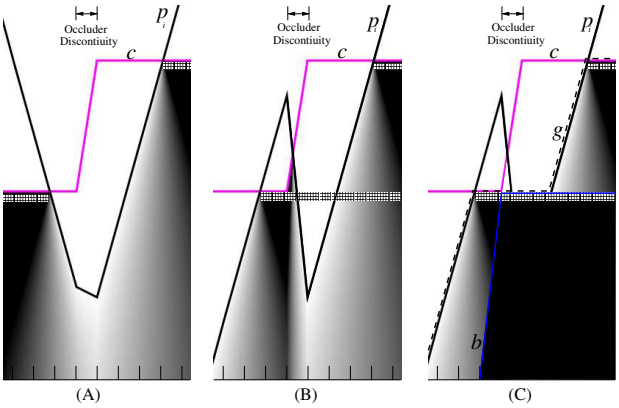

Figure 7. When occluders face each other (A), interpolating c and p_i in the shadow map works quite well. But when occluders are on top of each other (B), interpolation can cause ugly artifacts when switching between occluders. The same occluders have no artifacts (C) when the occluder discontinuity is clamped between the upper and lower limits g and b. Only ten shadowmap pixels are visible; the pixel centers are indicated with tick marks at the figure bottom.

and computes a target pixel's penumbra limit illumination using just nine pixel shader instructions.

3 Penumbra Limit Generation

The preceding section describes how to evaluate penumbra limit shadows at a particular point given the occluder and penumbra limit depths. This section describes how we compute those depths. Along a single silhouette edge at a depth e, the occluder depth is exactly $c = e$. At a signed horizontal distance w from the edge, the penumbra limit depth can be computed as $p_i = wsr$, where s is a shadow map pixel's horizontal size and r is the "depth rate" associated with the angular size of the light source, $r = 1.0/\tan(\text{lightAngle}/2)$. Hence the penumbra limit map is easy to compute exactly along a single occluder edge.

Since multiple-occluder shadows like those in Figure 6 are too complicated to represent exactly with any single shadow map, we must determine some approximation for the shadows cast by multiple occluders. That is, we must throw away some shadow detail in order to project the true shadow onto the space of shadows we can represent with Equation 1. This shadow complexity problem is the main reason why our method works best for small light sources—light sources that are too large (for example, an overcast sky) create many fuzzy overlapping penumbrae that are difficult to combine along in a single shadowmap ray.

The most general and reliable approach to the shadow complexity problem is to first calculate the true shadow profile (e.g., by dense sampling along each shadow ray) and then choose shadowing-function parameters to best fit this known shadow profile. Lokovic and Veach describe such an approach in detail in Deep Shadow Maps [11]. This approach is ideal for offline computation, but is too slow to perform at interactive rates.

For scenes consisting mostly of smooth objects, shadow-casting object silhouettes can be explicitly enumerated and rendered entirely on the graphics card as described in the next section. But for scenes consisting of rough objects like foliage silhouette enumeration can be expensive, and for objects like particle systems the silhouette is not even well defined. Hence sometimes better results can be obtained by expanding a rasterized depth image into a Penumbra Limit Map, as we describe in another work [10].

3.1 GPU Penumbra Limit Generation

The most GPU-friendly way to approximate multiple occluders is to extract each occluder's shadow silhouette, and expand and rasterize the soft shadow cast by that silhouette into the Penumbra Limit Map. This is exactly the same approach taken by Chan and Durand's Smoothies [5], but for Penumbra Limit Maps includes both the inner (inside the silhouette) and outer (beyond the silhouette) penumbrae.

The simplest heuristic for combining the occluder and penumbra limit depths from two different silhouettes is "highest penumbra limit wins". This means that moving down a shadowmap ray, we only consider the first penumbra we hit—penumbrae from deeper occluders are ignored. This can be implemented on graphics hardware by rasterizing the silhouettes with the z-buffer depth equal to the depth of the penumbra limit surface, and keep-

Figure 8. Stanford Bunny rendered using a 256x256 RGBA8 Penumbra Limit Map generated entirely on graphics hardware. Note self-shadowing errors at shadow of tip of right ear and base, and silhouette leakage at shadow of base of left ear.

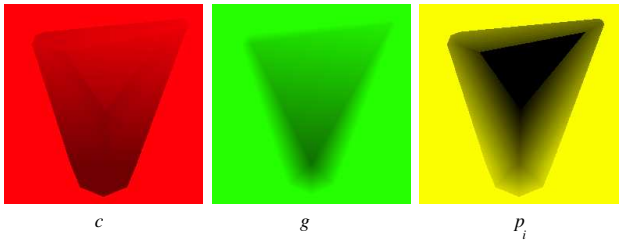

Figure 9. Penumbra Limit Map for triangle scene in Figure 11.

ing the highest z-buffer value as usual. We find it helps penumbra limit interpolation to bias up the z-buffer depth to expand the penumbra limit over a few extra pixels.

Where the soft shadows of two occluders with different depths overlap, interpolation can cause artifacts. Consider the case where a small object lies entirely above a large object. The large object's soft shadow causes no problems, but if the small object casts a soft shadow, it will cut down through the large object's center, casting light through the center of the object, as shown in Figure 7 (B). Worse, at the transition between the small and large objects' penumbrae, the penumbra limit depth p_i crosses over the occluder depth c, which results in a highly visible erroneous shadow transition. The imperfect solution we currently use is to clamp away the soft shadows at each occluder transition, by bringing together the geometry upper and lower limits g and b as shown in Figure 7 (C).

Hence overall we generate each penumbra limit map on the GPU in three steps:

1. Render all front-facing geometry to the geometry upper limit g channel, using the depth buffer to keep the topmost geometry. So far, this is exactly how normal shadow maps work.

2. Extrude all silhouette edges into small soft shadow ribbons, and use the ribbons to rasterize the occluder depth c and penumbra limit p_i along each soft shadow. We also raise the geometry upper limit g to match the outer penumbra limit so g does not clip away the outer penumbra.

3. Render all back-facing geometry to the geometry lower limit b channel, but throw away geometry above or near the occluder depth c to prevent a surface from occluding its own soft shadow.

The penumbra limit map for Figure 8 is computed on the graphics hardware using this silhouette enumeration technique. The model consists of 69,451 triangles, and the silhouette averages about 5,000 edges. We can prepare a new penumbra limit map using the full geometry and render the scene with it at 35fps. For a 2,000 triangle version of the same model, we get 100fps; for a 871K triangle model, we get 5fps.

4 Shadow Comparison

Figure 11 compares shadow images computed using a variety of techniques. In the top row, we examine the shadow cast on the ground plane; the bottom row shows the difference between this image and the exact soft shadow. The model is a single triangle skewed up from the ground, illuminated by an infinitely distant circular light source directly overhead. The top right triangle corner touches the ground plane, the top left corner is raised slightly, and the bottom corner is raised far off the surface. The actual penumbra limit map used is shown in Figure 9

Figure 10 lists the root-mean-square pixel error (-255 to $+255$) for each soft shadow algorithm for a variety of shadow map resolutions. We can see penumbra limit shadows have excellent performance compared to similar techniques, especially for low shadowmap resolutions. Smoothies, and Kirsch and Döllner's work perform acceptably at higher resolutions when taking into account that they only compute one half of the penumbra. Surprisingly, Reeves filtering actually gets less accurate as the shadowmap resolution increases–this is because Reeves filtering always creates one-shadowmap-pixel-wide soft

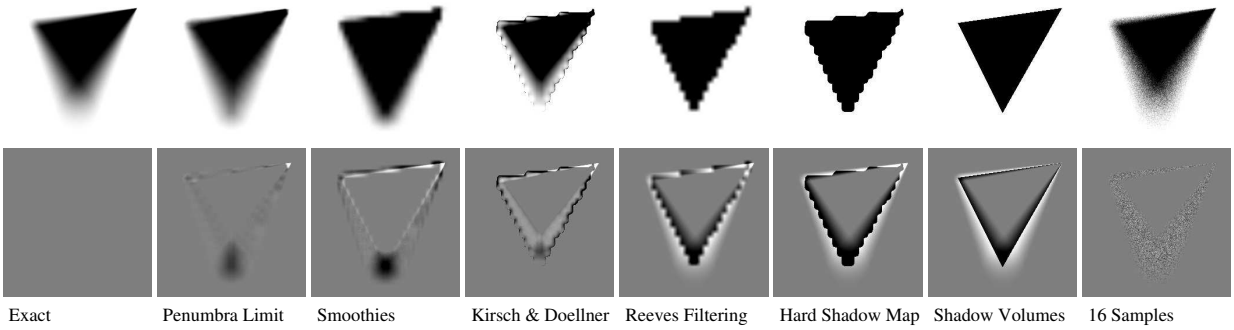

Figure 11. Comparison of soft shadows computed using a variety of techniques. Top row is the shadow, bottom row is the signed distance to the exact shadow. Smoothies and Kirsch and Döllner are compared with the outer and inner penumbra respectively. The shadow map resolution is just 32 x 32.

Resolution:	**16**	**32**	**64**	**128**
Penumbra Limit Maps	11.3	9.0	9.2	8.8
Smoothies	24.7	17.8	16.4	15.3
Kirsch & Doellner	37.2	23.0	15.6	10.9
Reeves Filtering	25.7	26.8	28.5	30.3
Hard Shadow Map	45.5	39.3	35.1	34.2
Shadow Volumes	33.5	33.5	33.5	33.5
16 Samples	12.0	12.0	12.0	12.0

Figure 10. Shadow algorithm RMS pixel error for various shadow map resolutions (in pixels on a side). Images for 32×32 resolution are shown in Figure 11.

shadows, which happen to better match the true soft shadow at low resolutions! The performance of all five shadowmap techniques was approximately equal ignoring the shadowmap setup time, but penumbra limit shadows do have a slightly more complicated shadowmap setup.

Shadow volumes and stochastic sampling do not use a shadow map, and hence always show the same error. 16 stochastic sample soft shadows were approximately 10 times slower than any other method. The "exact" image was computed via Monte Carlo raytracing taking 4,096 shadow rays per pixel, which is hundreds of times too slow for interactive use.

Penumbra limit shadows display a few minor problems. At the large blurry model corner, penumbra limit shadows are a bit darker than the exact image. Because of the low shadowmap resolution, penumbra limit shadows cannot capture the sharp corner at the top left. For a more complicated model, there might be a few artifacts where soft shadows overlap. But as claimed, up to depth quantization error penumbra limit shadows are exact along straight occluder edges. Unlike with the other shadow map methods, with Penumbra Limit Maps the impact of the low shadowmap resolution is difficult to even discern.

5 Conclusions and Future Work

We have presented Penumbra Limit Maps, an extension of the shadow map to generate soft shadows on graphics hardware. This technique produces good results even with bilinear interpolation, can be physically correct along object edges even for arbitrary shadow receivers, and yet is about as fast as competing techniques.

We have discussed the difficulties with representing multiple simultaneous soft shadows along a single shadowmap ray. A logical extension of this technique would be to store a series of occluder and penumbra limit depths, and then combine or choose between them at runtime. It is also possible a slightly different formulation would more easily support simultaneous soft shadows.

As presented, our method assumes a purely-radius-dependent infinitely distant light source assuming a constant surface BRDF. It should be possible to approximate local light sources using a perspective transform centered on the light source, which pushes the light source out to shadow-coordinates infinity. To support a nonsymmetric light source, we would need to store the occluder's orientation as well as location—storing this orientation as an angle, θ, would be straightforward, but would interpolate poorly along the line of $0 - 2\pi$ wraparound. A linearly-varying surface BRDF could be handled by adding a BRDF-rate axis to the (currently 1D) visibility-illumination table. Similarly, it may be possible to extend this technique to exactly capture the shadow from multiple occluders, or occluders with curves or corners, by adding more axes to the visibility-illumination table.

References

[1] Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. Approximate soft shadows with an image-space flood-fill

algorithm. *Computer Graphics Forum*, 23(3):271–280, 2004.

[2] Ulf Assarson. *A Real-Time Soft Shadow Volume Algorithm*. PhD thesis, Chalmers University of Technology, 2003.

[3] Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Chuck Hansen, and Franois Sillion. Soft shadow maps: Efficient sampling of light source visibility. Technical Report RR-5750, INRIA, November 2005.

[4] Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. *Graphics Interface*, pages 219–228, 2002.

[5] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. *Eurographics Symposium on Rendering*, pages 208–218, 2003.

[6] William de Boer. Smooth penumbra transitions with shadow maps. *ACM Journal of Graphics Tools*, 2006. To appear.

[7] Eric Haines. Soft planar shadows using plateaus. *J. Graph. Tools*, 6(1):19–27, 2001.

[8] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*, pages 1–20, 2003.

[9] Florian Kirsch and Jürgen Döllner. Real-time soft shadows using a single light sample. *Winter School on Computer Graphics*, 11(1), 2003.

[10] Orion Sky Lawlor. *Impostors for Parallel Interactive Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, December 2004.

[11] Tom Lokovic and Eric Veach. Deep shadow maps. In *SIGGRAPH Proceedings*, pages 385–392, August 2000.

[12] Michael D. McCool. Analytic antialiasing with prism splines. In *SIGGRAPH Proceedings*, pages 429–436. ACM Press, 1995.

[13] T. Nishita and E. Nakamae. Half-tone representation of 3-d objects illuminated by area sources or polyhedron sources. In *IEEE 7th Intl. Computer Sw. & App. Conf. (COMPSAC)*, pages 237–242, 1983.

[14] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, University of Utah, October 1998.

[15] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH Proceedings*, pages 283–291. ACM Press, 1987.

[16] Cyril Soler and François Sillion. Fast calculation of soft shadow textures using convolution. In *SIGGRAPH Proceedings*, pages 321–332, Jul 1998.

[17] Lance Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH Proceedings*, volume 12, pages 270–274, August 1978.

[18] Chris Wyman and Charles Hansen. Penumbra maps: Approximate soft shadows in real-time. *Eurographics Symposium on Rendering*, pages 202–207, 2003.

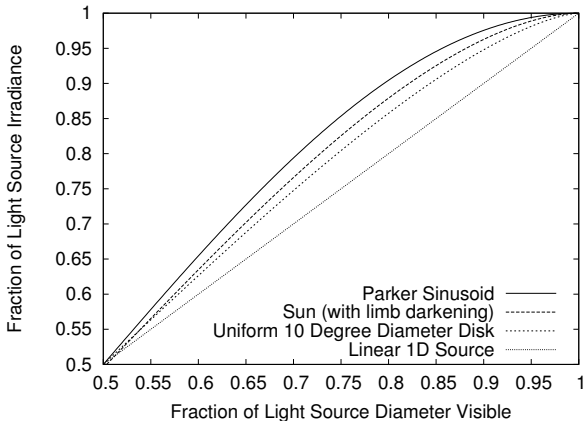

Figure 12. Normalized illumination from various light sources occluded by a straight edge. The graph is odd-symmetric about (0.5,0.5), since by complementarity $f(l) = 1 - f(1 - l)$.

A Visibility to Illumination

Equation 1 gives us the fraction l of the diameter of the light source that is visible at a point. For an infinitesimal, uniform-radiance, 1D line source directly overhead, this is equal to the fraction of the light that arrives at the point. But for any other light source, visibility isn't quite the same as illumination. That is, covering up the first 10% of the diameter of a disk light source decreases the delivered illumination by less than 6%, because the portion of the disk we begin covering is fairly narrow.

The fact that for a disk, visibility and illumination have a nonlinear relationship is well-known, but a large number of authors [5] [18] [4] [7] have erroneously described the visibility-illumination relationship as "sinusoidal". Parker even gives an analytic expression for this purported sinusoid [14]. However, the actual visibility-illumination relationship is not a sinusoid. Though it is easy to derive the exact relationship for simple shapes [10], we follow Haines [7] and precompute a visibility-illumination table, indexed by the linear visibility fraction l.

A visibility-illumination table allows us to exactly represent the irradiance from any purely radius-dependent light source (assuming a constant surface BRDF). In particular, the sun's colder outer layers cause an intensity drop of about 50% near the edge of the solar disk. This "limb darkening" effect flattens the visibility-illumination curve, as shown in Figure 12. In the images presented here, we use a table derived from a linear-light image of the sun, integrated and stored into a 512-entry lookup table.