Penumbra Maps

Chris Wyman   Charles Hansen

University of Utah, School of Computing
Technical Report UUCS-03-008

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

April 23, 2003

Abstract

Generating soft shadows quickly is difficult. Few techniques have enough flexibility to interactively render soft shadows in scenes with arbitrarily complex occluders and receivers. This paper introduces the *penumbra map*, which extends current shadow map techniques to interactively approximate soft shadows. Using object silhouette edges, as seen from the center of an area light, a map is generated containing approximate penumbral regions. Rendering requires two lookups, one into each the penumbra and shadow maps. Penumbra maps allow arbitrary dynamic models to easily shadow themselves and other nearby complex objects with plausible penumbrae.

1 Introduction

Shadows provide cues to important spatial relationships. By changing shadow size, position, or orientation in an image, an object can appear to change size or location[1]. Similarly, soft shadows give contact cues. As an occluder approaches a shadowed object, its soft shadow becomes sharper. When objects touch the shadow is completely hard.

Many recent interactive applications have incorporated real-time shadows. Generally, these applications use shadow volumes[2], shadow maps[3], or related techniques. These methods use point light sources which only cast hard shadows. Since real world lights occupy not a point but some finite area, realistic images require soft shadows. Thus, as interactive graphics systems become more realistic, methods for quickly rendering soft shadows are needed.

Shadows consist of two parts, an *umbra* and a *penumbra*. Umbral regions occur where a light is completely occluded from view and penumbrae occur when a light is partially visible. Until very recently the only techniques to compute these regions involved either evaluating complex visibility functions[4] or merging hard shadows rendered from various points on the light[5]. Evaluating visibility is slow, and sampling techniques produce banding artifacts unless many samples are used. Other approximations have emerged, but most do not allow dynamically moving objects to shadow arbitrary receivers.

We introduce the *penumbra map*, which allows arbitrary polygonal objects to dynamically cast approximate soft shadows onto themselves and other arbitrary objects. A penumbra map augments a standard shadow map with penumbral intensity information. Our shadows (see Figure 1) harden when objects touch, avoid banding artifacts inherent in sampling schemes, and are generated interactively with commodity graphics hardware. Additionally, penumbra maps can leverage existing research on shadow maps (e.g. perspective shadow maps[6] to reduce foreground shadow aliasing). On the other hand, our approach breaks down when the umbra region significantly decreases or disappears. This happens for very large area light sources or as an occluder moves away from the objects it shadows.

The next section describes related work followed by a discussion of our algorithm in section 3. Section 4 discusses some implementation specifics and outlines the limitations of our technique. Section 5 presents our results, after which we conclude.

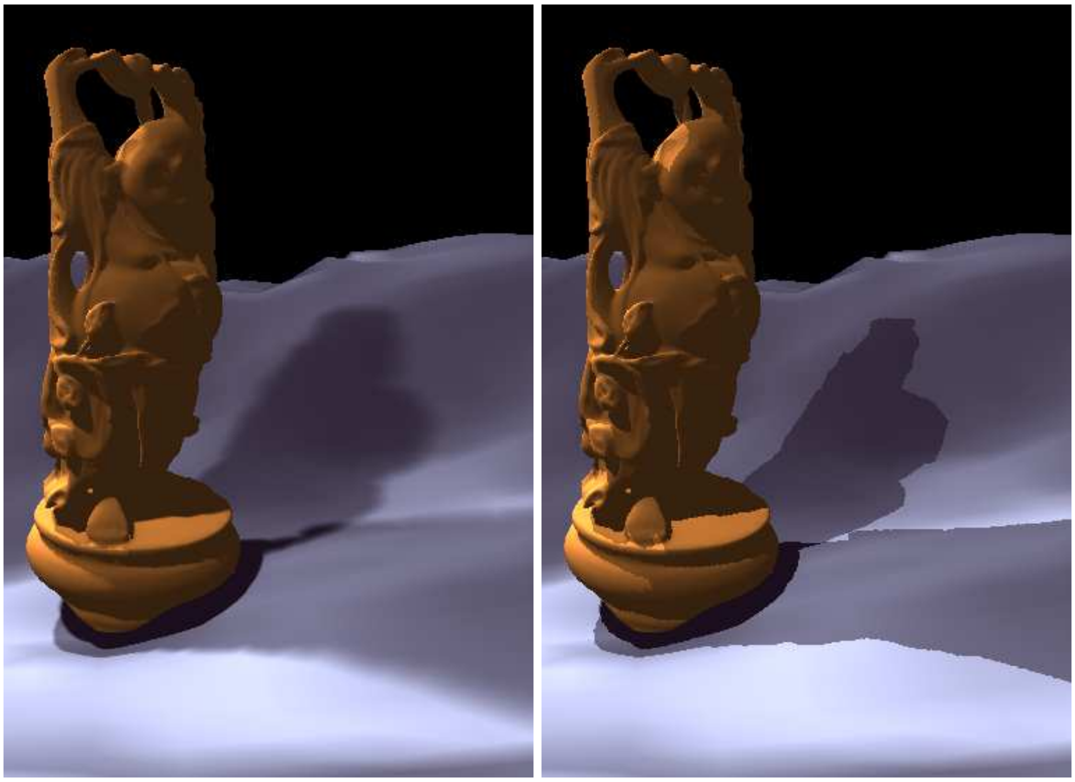

Figure 1: *With two penumbra maps, this scene runs at 11 fps for 1024x1024 images (left). Compare to shadow maps (right) which only render hard shadows.*

2 Previous Work

This section provides an overview of previous work in rendering interactive shadows. As complete coverage of other shadow techniques is beyond the scope of this paper, refer to Woo *et al.*[7] and Akenine-Möller and Haines[8] for a more complete review.

Researchers have proposed soft shadow techniques which run quickly, but do not handle dynamic scenes interactively. For instance, Soler and Sillion[9] convolve images of hard shadows and the light source to approximate soft shadows for nearly parallel configurations. Stark and Riesenfeld[10] use vertex tracing to compute exact shadows for polygonal scenes. Various backprojection techniques[11] can generate soft shadows via discontinuity meshing.

Parker *et al.*[12] use a point light source and a "soft-edged object" to raytrace soft shadows using only a single sample. They created this technique for interactive raytracing, limiting use to applications with significant computational resources.

The two most common techniques for real-time shadows are shadow volumes and shadow mapping. Shadow volumes[2] create a polygonal shadow model based on object silhouettes as seen from the light. Heidmann[13] implements this technique in hardware using a stencil buffer. Shadow mapping[3] renders the light's view of a scene into a depth map. When rendering, each fragment's depth is compared to the depth map to determine its visibility from the light. Segal *et al.*[14] show a hardware implementation of shadow maps.

As used today, shadow volumes and shadow mapping only allow hard shadows. However, various researchers have proposed extensions which allow them to render soft shadows in certain cases. Reeves *et al.*[15] introduce *percentage closer filtering*, which reduces aliasing by blurring the shadow map. This blurring can give the impression of softer shadows. Heidrich *et al.*[16] use the two end points of a linear light to compute a non-binary visibility map of a scene, allowing for soft shadows. However, computing a visibility map can take a couple seconds.

Haines[17] presents a technique to render a shadow texture on a receiving plane. He suggests approximating umbral regions using standard hard shadow techniques and extending these regions with an approximate penumbra. These penumbrae are computed using the following process (see Figure 2). From the center of the light, object silhouettes are found and a hard shadow is rendered onto the texture plane. Next, through each silhouette vertex a cone is drawn with the tip at the vertex and the base at the plane. Finally, hyperboloid sheets are drawn connecting each silhouette edge and the adjacent cones. The radii of the cones are based on the distance between the silhouette and the plane, and the color rendered in the shadow texture ranges from black (fully shadowed) to white (fully illuminated) as the cones and sheets approach the plane.

Akenine-Möller and Assarsson[18] extend the shadow volume technique using a method similar to Haines. Instead of computing a shadow sheet at each silhouette edge, they generate a *penumbra wedge* consisting of four planar sides. A per-fragment program renders these wedges to a light buffer, which can be used to render the scene with various shadow intensities. To get sufficient intensity gradations in their penumbrae, however, they need a 16-bit stencil buffer for use as a light buffer. Such stencil buffers are not available on current generations of graphics cards, though the functionality can probably be emulated. Additionally, they are limited to occluders whose silhouettes form closed loops, with exactly two silhouette edges per vertex. Arbitrary objects can have more complex silhouette behavior. We found that vertices with three or four adjacent silhouettes edges are not uncommon in typical models, and some pathological vertices can have up to eight.

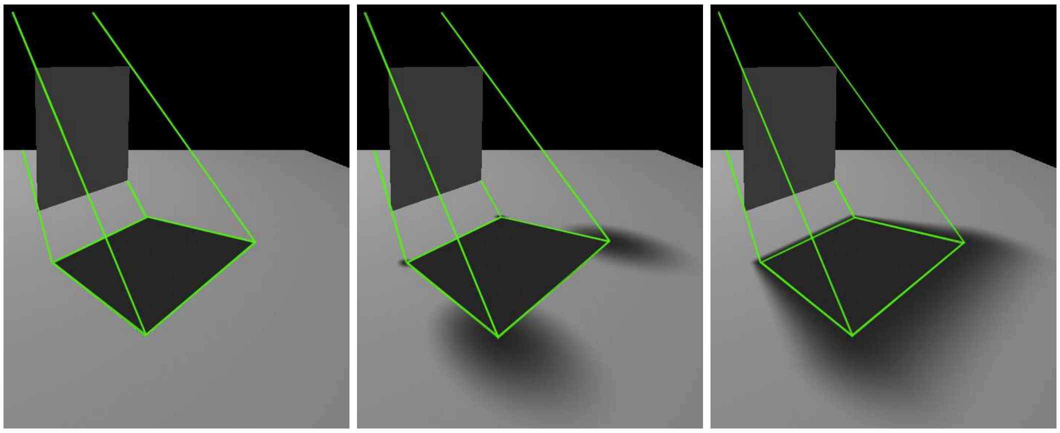

Figure 2: *Haines generates soft shadows by (left) rendering a hard shadow, (middle) rendering cones at each silhouette vertex, and (right) rendering sheets connecting the cones.*

Brabec and Seidel[19] approximate soft shadows using a single depth map. They transform an eye-space coordinate to light-space using the standard shadow map technique, then search a neighborhood around the transformed point to find nearby objects which may partially occlude the light. This technique can generate approximate soft shadows quickly, but since it uses object IDs, soft self shadowing is not possible. Additionally, the neighborhood search may not be plausible for high resolution depth maps.

3 Penumbra Maps

As people are often poor judges of soft shadow shape[20], plausible soft shadows should suffice in interactive environments. Haines'[17] shadow plateaus give compelling shadows quickly enough to use with dynamic occluders, but lack the ability to shadow arbitrary surfaces. The penumbra map technique draws heavily from this work.

Two observations allow us to develop an algorithm to shadow arbitrary surfaces. First, a shadow map can easily create the hard shadow used to approximate an umbra. Second, if one assumes this hard shadow approximates the umbra, then the entire penumbra is visible from the point on the light used for the hard shadow. This allows the penumbra information to be stored in a single texture we call the *penumbra map*. This texture stores the penumbral intensity on the foremost polygons visible from the light, just as a shadow map stores depth information about these surfaces.

Rendering with penumbra maps is a three-pass process. The first pass renders a standard shadow map from the viewpoint of a point light source at the approximate center of the light. The second pass renders the penumbra map. The third pass combines depth information from the shadow map and intensity information from the penumbra map to render the final image.

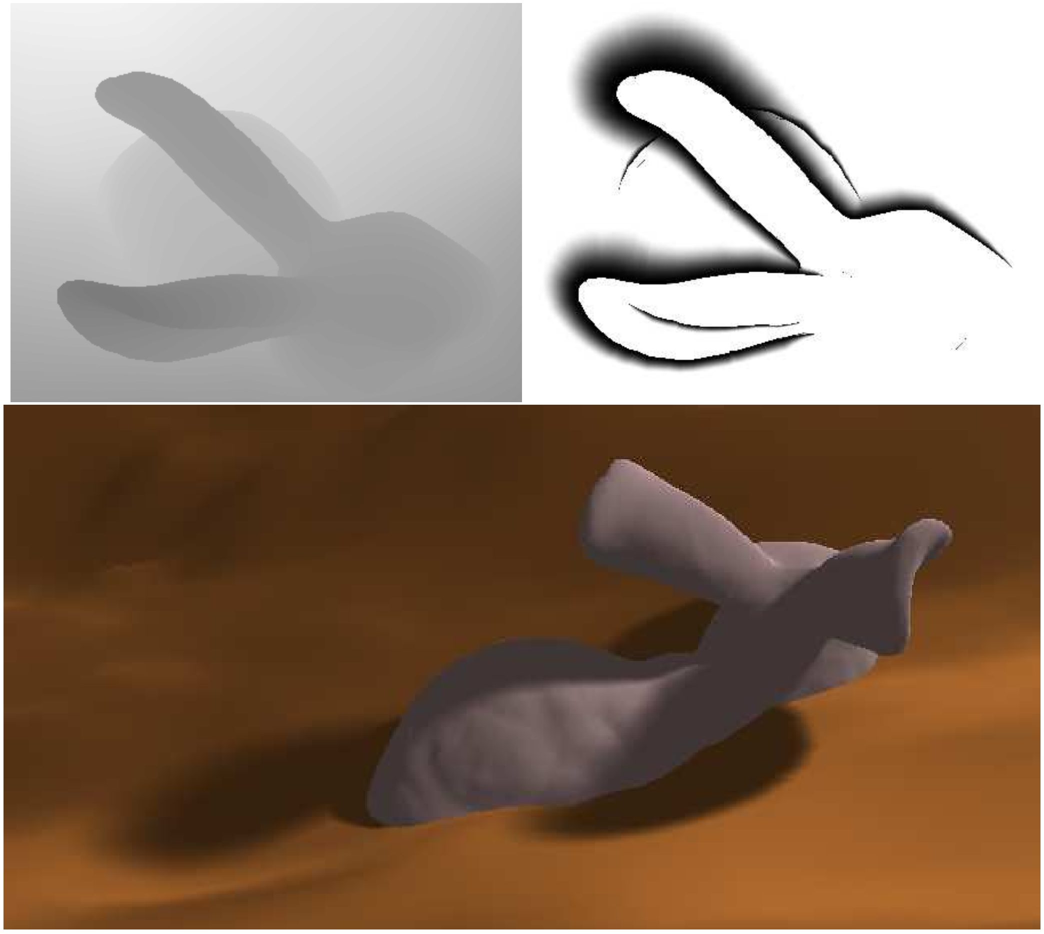

Figure 3: *An example shadow map (top left), corresponding penumbra map (top right), and the final rendered result.*

Let $\mathcal{V} \equiv \{v_1, v_2, \ldots\}$ and $\mathcal{E} \equiv \{e_1, e_2, \ldots\}$ be the set of silhouette vertices and edges, as seen from the light. Let L_r be the light radius, Z_{v_i} the depth value of vertex v_i from the light, and Z_{far} be the distance to the light's far plane. Then, to generate a penumbra map (such as in Figure 3):

- Clear the penumbra map to white.
- Find $\mathcal{V}$ and $\mathcal{E}$ for the current light.

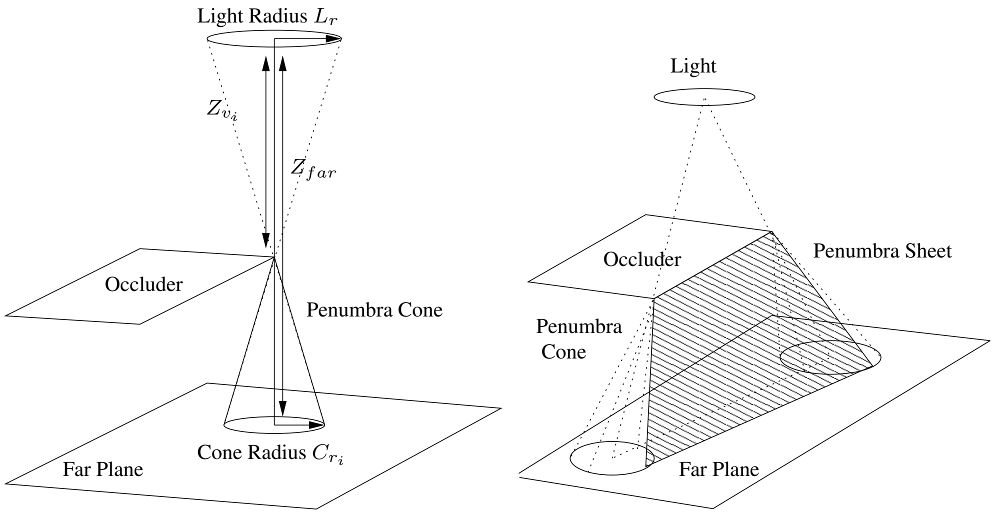

Figure 4: *Each cone's tip is located at a vertex v_i with the base located at the far plane (left). Using simple geometry, we compute the cone radius C_{r_i}. Each sheet (right) connects two adjacent cones.*

- $\forall v_i \in \mathcal{V}$, draw a cone with tip at v_i and base at the far plane (see Figure 4). The cone radius $C_{r_i} = \frac{(Z_{far} - Z_{v_i})L_r}{Z_{v_i}}$. We subdivide each cone into a number of triangles with one vertex at v_i and two on the far plane.
- $\forall e_i \in \mathcal{E}$, draw a sheet connecting adjacent cones. Depending on the cone radii, this quad may be non-planar. We subdivide extremely non-planar quads to avoid artifacts.

Each pixel in the penumbra map corresponds to a pixel in the shadow map. Each penumbra map pixel stores the shadow intensity at the corresponding surface in the shadow map. A fragment program applied to the penumbra sheets and cones computes this intensity using the simple geometry shown in Figure 5. The idea is that by using Z_{v_i}, the depth of the current cone or sheet fragment Z_F, and depth of the corresponding shadow map pixel Z_P, we can compute the light intensity at point P. Equation 1 specifies this computation.

$$I = 1 - \frac{Z_P - Z_F}{Z_P - Z_{v_i}} = \frac{Z_F - Z_{v_i}}{Z_P - Z_{v_i}} \tag{1}$$

We compute Z_{v_i} on the CPU on a per-vertex basis. For cones Z_{v_i} is constant, and for sheets we use the rasterizer to interpolate between the Z_{v_i} values of the two adjacent cones. Z_P can be computed by referencing the shadow map, and Z_F is automatically computed by the rasterizer when processing fragment F.

This process gives us a linear intensity gradient through our approximate penumbra. Parker *et al.* note that for spherical lights this intensity should vary sinusoidally. They approxi-

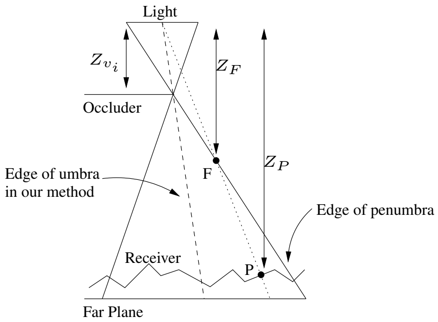

Figure 5: *Each fragment F on a cone or sheet corresponds to some surface location P visible in the shadow map. By using Z_{v_i}, Z_F and Z_P, we compute the intensity I using Equation 1.*

mate this sinusoidal falloff using the Bernstein interpolant $s = 3\tau^2 - 2\tau^3$. We use their approximation in our results.

Pseudocode for a fragment program to compute a penumbra map follows:

FragmentProgram(Z_{v_i}, *F,* S_{map} *)*

F_{coord} = *GetWindowCoord(F)*

Z_P = *TextureLookup(* S_{map}, F_{coord} *)*

$Z_F = F_{coord_z}$

if ($Z_F > Z_P$*) DiscardFragment()*

Z'_P = *ConvertToWorldSpace(* Z_P *)*

Z'_F = *ConvertToWorldSpace(* Z_F *)*

$I = (Z'_F - Z_{v_i}) / (Z'_P - Z_{v_i})$

$I' = 3I^2 - 2I^3$

$Output_{color} = I'$

$Output_{depth} = I'$

Since both the shadow map, S_{map}, and the penumbra map are rendered with the same viewing matrices, the window coordinates of fragment F can be used to find its corresponding point P in the shadow map. Due to the non-linearity of z-buffer values, Z_F and Z_P must be converted back to world-space distances (Z'_F and Z'_P) before use. Note that Z'_F can be computed on a per-vertex basis and can be interpolated by the rasterizer to save fragment instructions. Since the penumbra map only requires a single color channel, further savings can be achieved by storing the shadow map and penumbra map in different channels of the same image.

Rendering soft shadows with a penumbra map is simple. For each pixel rendered from the camera's viewpoint, a comparison with the depth in the shadow map determines if the pixel is completely shadowed. If not fully shadowed, a lookup into the penumbra map gives an approximation of the light reaching the surface. Like shadow mapping, penumbra maps work in scenes with multiple light sources. Instead of computing a single shadow map and penumbra map, each light requires one of each.

4 Implementation

When writing our application, we made a number of implementational choices which affect our results. First, we use a spherical light source because people often cannot distinguish between shadows from lights of various shapes. As Haines[17] notes, this algorithm need not be limited to spherical light sources. For example, in the case of a triangular source the cones generated for the penumbra map would have triangular bases.

Second, our application detects silhouettes using a brute force algorithm. We did not use a more intelligent silhouette extraction technique because we expected the graphics card would be the bottleneck. Surprisingly, we found our silhouette code takes 30% of the render time. Obviously, fast silhouette techniques would be used for interactive applications.

One detail which complicates implementation is how to deal with overlapping shadows. Given two silhouette edges with overlapping penumbral regions, there are multiple ways of counting their contributions (see Figure 6). When one shadow completely contains another only the darkest shadow should be used. If just the penumbrae overlap the shadow contributions should be summed. Often when the object silhouettes intersect, multiplication best approximates the true interaction. Unfortunately, there does not seem to be a straightforward way to determine which of the three methods to use on a per-fragment basis during cone and sheet rasterization. Our implementation uses a modified depth test to determine which cone or sheet shades a particular fragment in the penumbra map. As the pseudocode above shows, we store the penumbra intensity in the depth channel, and use `glDepthFunc( GL_LESS )` to always choose the darkest shadow in a given pixel. This leads to artifacts in the shadows. As in Haines' work, these are most noticeable at silhouette concavities. Such artifacts worsen as the size of the penumbra increases.

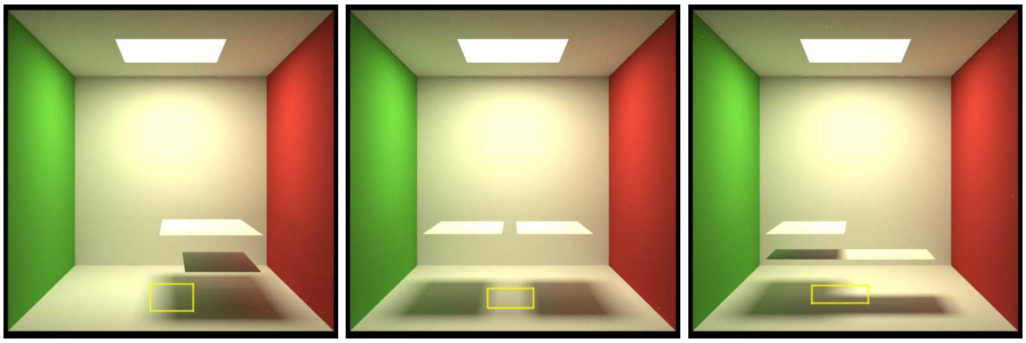

Figure 6: *Three different types of interactions between overlapping penumbra. At left, only the darkest contribution is needed. In the center, shadow contributions should be summed. At right, multiplying the contributions from the two polygons best approximates the result.*

4.1 Discussion of the Penumbra Maps

Before discussing our results, we note what limitations the assumptions inherent in the penumbra map technique impose. We assume that silhouettes of an object remain constant over the area of a light and that the umbra can be approximated by a hard shadow. Akenine-Möller and Assarsson[18] and Haines[17] also use silhouettes computed at a single point on the light. Brabec and Seidel's[19] technique implicitly makes this assumption by using only a single depth map. Obviously as an area light increases in size, silhouettes vary more over the light so the generated shadows will become less realistic.

We believe approximating the umbra by a hard shadow is reasonable in many cases, as most people are poor judges of soft shadow shape[20]. If plausible soft shadows are required in an interactive application, using a hard shadow for the umbra may be sufficient. As a shadow's umbra size shrinks, our approximation leads to noticeably larger, darker shadows. Shadow umbras shrink as light size grows and as occluders and receivers move further apart. Thus, our method works best for relatively small light sources and objects occluding nearby objects.

5 Results

We implemented penumbra maps in an interactive application using OpenGL. Our results were obtained using a Windows 2000 system with a 2.0 GHz Pentium 4 processor and an ATI Radeon 9700 PRO graphics accelerator. We use OpenGL ARB vertex and fragment

	Bunny	Dragon	Buddha (1 light)	Buddha (2 lights)
Penumbra Maps	18.1	14.5	18.3	11.0
Shadow Maps	42.0	48.1	48.1	27.4

Table 1: *Framerate comparison between soft shadows using penumbra maps and hard shadows using shadow maps.*

program extensions for our shaders. Both the shadow and penumbra maps are rendered into p-buffer textures so they can be used directly without reading them back into main memory.

All our scenes are rendered at 1024 x 1024 with shadow and penumbra maps of the same size. For complex models such as the bunny, buddha, and dragon we found we could get equivalent quality shadows with simplified models, as soft shadows effectively blur detail. This increases aliasing artifacts, though we reduce them by adding a larger bias. We used 10,000 polygons to generate shadows for the bunny (Figure 7) and the dragon (Figure 8). Buddha's shadow (Figure 1) uses 5,000 polygons. Table 1 shows framerates for these models using penumbra and shadow maps. Note that for comparison purposes, the hard shadows were timed using a fragment program similar to the one used for penumbra maps. This program computes the Phong lighting and performs the lookup into the shadow map, which is significantly slower than using other capabilities of the hardware designed specifically for those operations.

Thirty percent of our computation time is used by our brute force silhouette extraction code. Thirty-five percent is spent rendering the penumbra map and the remaining time is used during the render pass. Note the render pass includes fragment code to perform lighting computations and check light visibility using the shadow map. These operations take 15 of the 22 instructions in our ARB fragment program. To render penumbra maps, we use a fragment program with 24 assembler instructions.

6 Conclusions and Future Work

In this paper, we presented the *penumbra map*, a new technique for rendering approximate soft shadows in real-time. Penumbra maps allow dynamically moving polygonal models to cast soft shadows onto themselves and other complex objects. These results work best for relatively small penumbrae. Penumbra maps provide a simple multi-pass extension to shadow mapping for easy incorporation into existing shadow map-based systems.

While penumbra maps give plausible results, there are still areas we wish to improve in future work. First, we believe it may be possible to approximate a full penumbra using vertex programs to adjust the silhouette edge positions. However, this is complicated by the fact that penumbrae will no longer lie on the foremost polygons in the shadow map.

We also wish to explore the possibility of moving the entire algorithm into hardware. Future graphics accelerators will have the ability to render to a vertex array. If this allows us to create new primitives, we believe we can combine our work with that of McCool[21] to move the silhouette extraction and cone and sheet generation onto the graphics card, greatly reducing the burden on the CPU.

Acknowledgments

Without the help and advice of Aaron Lefohn, Milan Ikits, and Joe Kniss, our code would have been infinitely harder to write. The authors would also like to thank the numerous people who read and reviewed our work. Their invaluable comments have significantly improved this paper. This material is based upon work supported by the National Science Foundation under Grants: 9977218 and 9978099.

References

[1] L. Wanger, J. Ferwerda, and D. Greenberg, "Perceiving spatial relationships in computer-generate images," *IEEE Computer Graphics & Applications*, vol. 12, pp. 44–58, May 1992.

[2] F. Crow, "Shadow algorithms for computer graphics," in *Proceedings of SIGGRAPH*, pp. 242–248, 1977.

[3] L. Williams, "Casting curved shadows on curved surfaces," in *Proceedings of SIGGRAPH*, pp. 270–274, 1978.

[4] D. Hart, P. Dutré, and D. Greenburg, "Direct illumination with lazy visibility evaluation," in *Proceedings of SIGGRAPH*, pp. 147–154, 1999.

[5] P. Heckbert and M. Herf, "Simulating soft shadows with graphics hardware," Tech. Rep. CMU-CS-97-104, Carnegie Mellon University, January 1997.

[6] M. Stamminger and G. Drettakis, "Perspective shadow maps," in *Proceedings of SIGGRAPH*, pp. 557–562, 2002.

[7] A. Woo, P. Poulin, and A. Fournier, "A survey of shadow algorithms," *IEEE Computer Graphics & Applications*, vol. 10, pp. 13–32, November 1990.

[8] T. Akenine-Möller and E. Haines, *Real-time Rendering*. Massachusetts: AK Peters, second ed., 2002.

[9] C. Soler and F. Sillion, "Fast calculation of soft shadow texture using convolution," in *Proceedings of SIGGRAPH*, pp. 321–332, 1998.

[10] M. Stark and R. Riesenfeld, "Exact illumination in polygonal environments using vertex tracing," in *Eurographics Rendering Workshop*, pp. 149–160, 2000.

[11] G. Drettakis and E. Fiume, "A fast shadow algorithm for area light sources using backprojection," in *Proceedings of SIGGRAPH*, pp. 223–230, 1994.

[12] S. Parker, P. Shirley, and B. Smits, "Single sample soft shadows," Tech. Rep. TR UUCS-98-019, University of Utah, October 1998.

[13] T. Heidmann, "Real shadows, real time," *Iris Universe*, pp. 23–31, November 1991.

[14] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, "Fast shadows and lighting effects using texture mapping," in *Proceedings of SIGGRAPH*, pp. 249–252, 1992.

[15] W. Reeves, D. Salesin, and R. Cook, "Rendering antialiased shadows with depth maps," in *Proceedings of SIGGRAPH*, pp. 283–291, 1987.

[16] W. Heidrich, S. Brabec, and H.-P. Seidel, "Soft shadow maps for linear lights," in *Eurographics Rendering Workshop*, pp. 269–280, 2000.

[17] E. Haines, "Soft planar shadows using plateaus," *Journal of Graphics Tools*, vol. 6, no. 1, pp. 19–27, 2001.

[18] T. Akenine-Möller and U. Assarsson, "Approximate soft shadows on arbitrary surfaces using penumbra wedges," in *Eurographics Rendering Workshop*, pp. 309–318, 2002.

[19] S. Brabec and H.-P. Seidel, "Single sample soft shadows using depth maps," in *Proceedings of Graphics Interface*, pp. 219–228, 2002.

[20] L. Wanger, "The effect of shadow quality on the perception of spatial relationships in computer generated imagery," in *Proceedings of Symposium on Interactive 3D Graphics*, pp. 39–42, 1992.

[21] M. McCool, "Shadow volume reconstruction from depth maps," *ACM Transactions on Graphics*, vol. 19, pp. 1–26, January 2000.

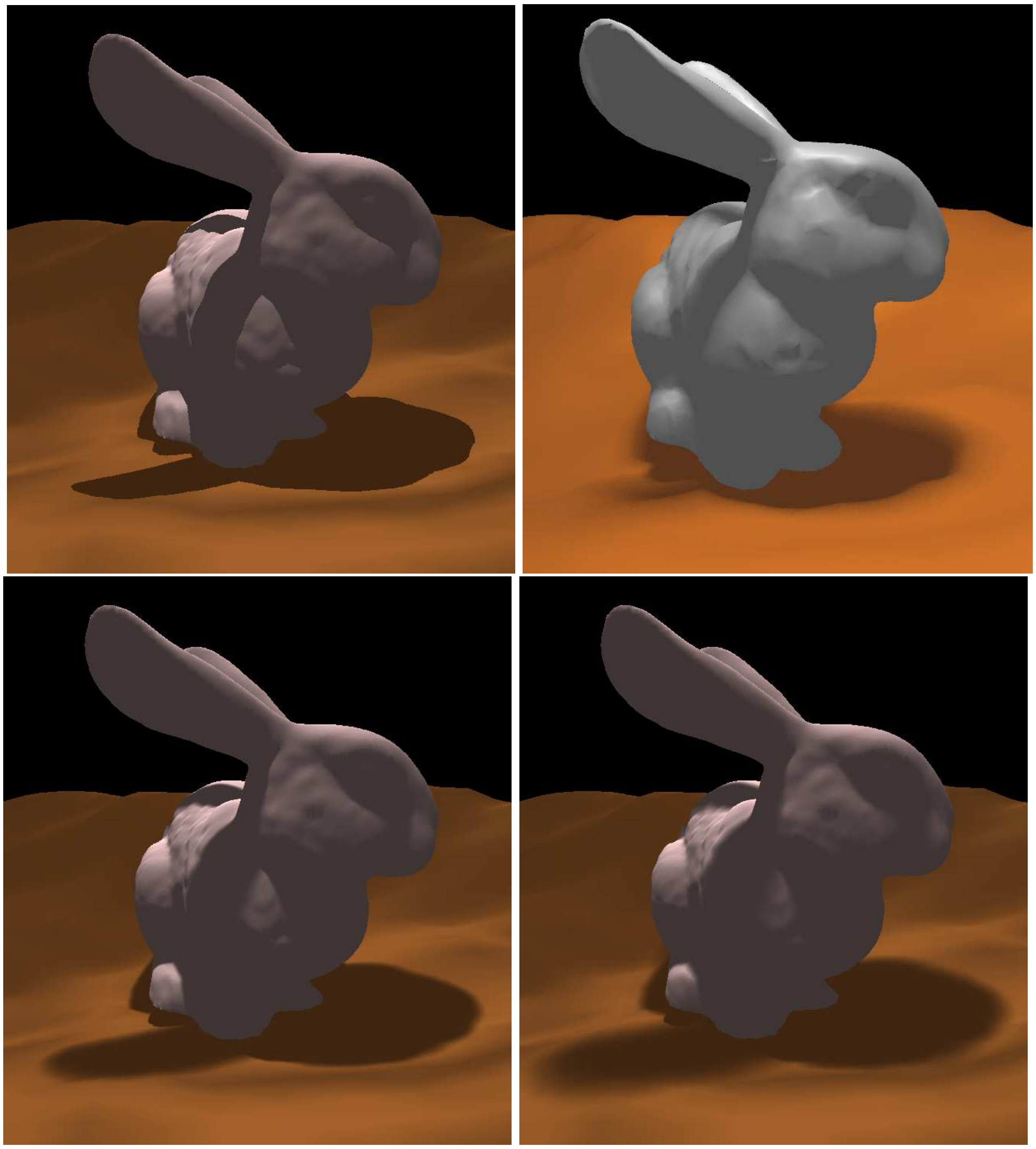

Figure 7: *Comparison of the Stanford Bunny with shadow maps (top left), pathtraced soft shadows (top right), and penumbra maps with two different sized lights (bottom). For this data set, we generate shadows using a 10k polygon model and render the shadows onto the full ($\sim$70k polygon) model. The pathtraced image uses the 10k polygon model.*

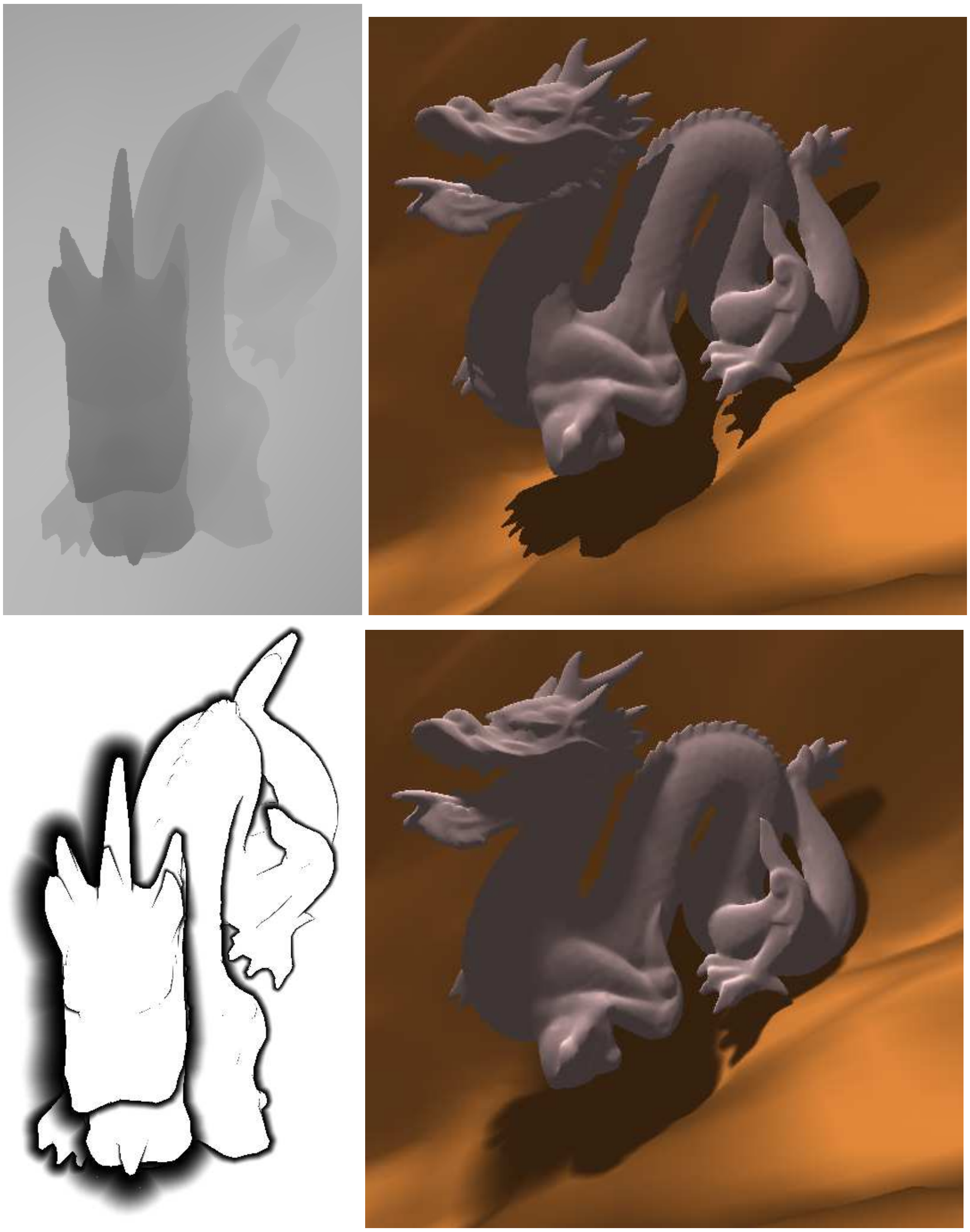

Figure 8: *Using a standard shadow map results in hard shadows (top), add a penumbra map to get soft shadows (bottom). Using a 10k polygon dragon model for the shadows and a 50k polygon model to render, we get 14.5 fps at 1024x1024.*