

Soft Planar Shadows Using Plateaus

Eric Haines
Autodesk, Inc.
Ithaca, New York
erich@acm.org
June 18, 2001

Abstract: This paper presents an algorithm for rapidly creating and rendering soft shadows on a plane. It has the advantages of rapid generation on current hardware, as well as high quality and a realistic feel on a perceptual level. The algorithm renders the soft shadow to a texture. It begins with a hard drop shadow, then uses the object's silhouette edges to generate penumbræ in a single pass. Though the shadows generated are physically inaccurate, this technique yields penumbræ that are smooth and perceptually convincing.

1 Introduction

Projective shadow techniques are commonly used in real-time rendering. Blinn [1] discusses how to generate and render shadows by projecting the shadow casting object's polygons onto a ground plane. A related, more general method is to generate a texture containing the shadow cast, which is then projected onto the objects receiving the shadow [7, 8, 11].

One drawback of these methods is that the shadows have a hard edge, i.e. there are no penumbræ. This is often perceived as being unrealistic and prone to misinterpretation. A number of techniques have been developed to synthesize penumbræ for projective shadows. One method is to blur the shadow texture generated as a post-process. Soler and Sillion [12] improve upon this technique by using the shape of the light itself as the filtering kernel, so giving a variety of penumbræ types. However, such blurring is uniform over the shadow edges, which is valid only when the edges of the object casting the shadow are all the same distance from the shadow receiver. In reality, if an object touches the ground, the penumbra should be non-existent at this point, and widen out with the object's height (see Figure 1).

Heckbert and Herf [5, 6] take the approach of rendering the hard edged planar projection of the shadow caster a large number of times, sampling the area light source across its surface. The average of these shadows gives a smooth, realistic penumbra. The main drawback of this method is that a large number of passes (typically 16 or more) are needed for smooth penumbræ; too few, and the hard-edged shadow edges do not blend, creating distracting artifacts. Gooch et al. [4] ameliorate these artifacts by generating a series of concentric shadows. However, multiple passes are still needed, sharp shadow boundaries are still present, and visibly incorrect penumbræ are generated where the object touches the ground.

2 Algorithm

The algorithm presented here gives penumbræ in a single pass that are perceptually acceptable. The idea is to cast a hard drop shadow onto a texture, then create an approximation of the penumbra along the silhouette edges. The resulting image is used as a texture on the ground plane. This algorithm was inspired by the work of Parker et al. [9], in which the authors use ray tracing for shadow testing and extend each shadowing object's edges outwards as gradients, then use the darkest shadowing object found.

Figure 1 shows the basic idea behind the algorithm. The vertices and edges of the object are cast onto the ground plane. The interior of the projected object is rendered in the darkest color, as is normally

done when casting a hard shadow. Where a silhouette vertex or edge lands is where the soft shadow is painted onto the ground plane. A vertex causes a radial gradient circle to be painted, with the vertex location being the darkest shade and going to white at the circle's edge. The radius of the circle is dependent on the height of the casting vertex above the ground plane. Similarly, each silhouette edge creates a quadrilateral or triangle, which is shaded from the darkest shade along the hard shadow edge and fading to white along the other edge.

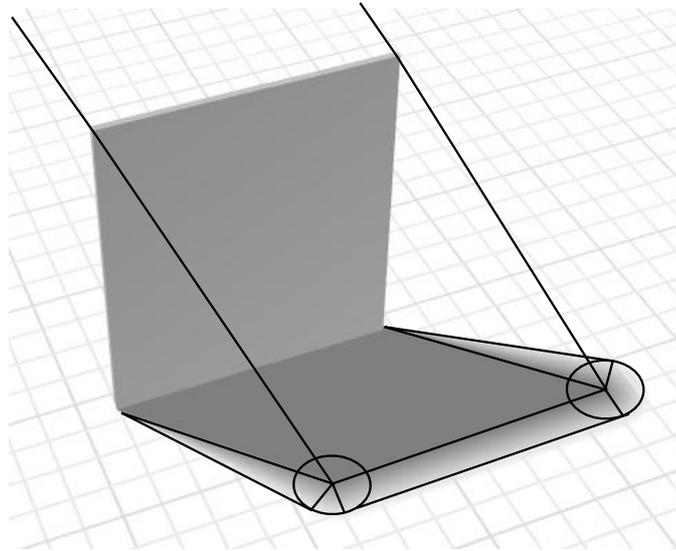


Figure 1: The object's vertices and edges are projected onto the ground. Silhouette vertices form radial gradient circles and edges form gradient quadrilaterals or triangles.

If these circles and polygons were drawn without regards to each other the results would be unconvincing at best, with light and dark areas intermingling at random. To implement Parker's idea of rendering the darkest shadowing object, the two-dimensional circle and polygonal painting primitives are turned into three-dimensional cones and hyperbolic sheets, with the darkness of the object determining its closeness in the Z -buffer. In this way when two drawing primitives overlap the darkest one will always then be the one seen.

The algorithm consists of drawing 3D primitives to form a texture approximating the soft shadow. First the frame buffer where the shadow texture image will be formed is cleared to white, and the Z -buffer cleared as usual. The view for this shadow texture is set to look straight down onto the ground plane (i.e. a plan view).

Next, shadow casting objects in the scene are projected onto the ground plane to create the umbra. In addition to rendering the projected polygons to the color buffer in the darkest desired shadow color, these polygons are also rendered at a depth equal to the hither plane and their z -depths are stored in the Z -buffer. In this way they are the closest objects and so will cover any penumbra-forming objects, thereby ensuring that the darkest shadowing object is visible in the texture.

All silhouette edges of the shadow casters compared to the light source are then found. There are a variety of strategies for doing this process [10]. A simple scheme is to create a list of edges for each shadow casting object, with each edge pointing to the two polygons it shares. Each edge is then compared to the light source and if one polygon faces the light and the other faces away, then the edge is a silhouette edge. The silhouette edges are not needed in any particular order, so the processing cost is minimal.

The heart of the algorithm is drawing the penumbrae. For each projected silhouette-edge vertex, draw a cone with the tip pointing directly toward the viewer. The apex of the cone is at the hither, at the vertex's projected location on the texture. The base of the cone is centered around this shadow-cast location, and at the yon plane. The radius of the cone is proportional to the height of the original vertex above the ground. The cone is shaded by using a color per vertex: the apex is black and the base is white.

For each edge, the two cones formed by the edge's vertices are connected by a sheet at their outer tangents. The sheet is a flat quadrilateral when the cones have the same radius (i.e. are at the same height above the ground). When the radii differ, the quadrilateral's tangent edges are not in the same plane and the sheet formed is a hyperboloid. Each sheet is rendered with its closest edge (on the hither) as black and its far edge (on the yon) as white. Figure 2 shows the objects formed and rendered.

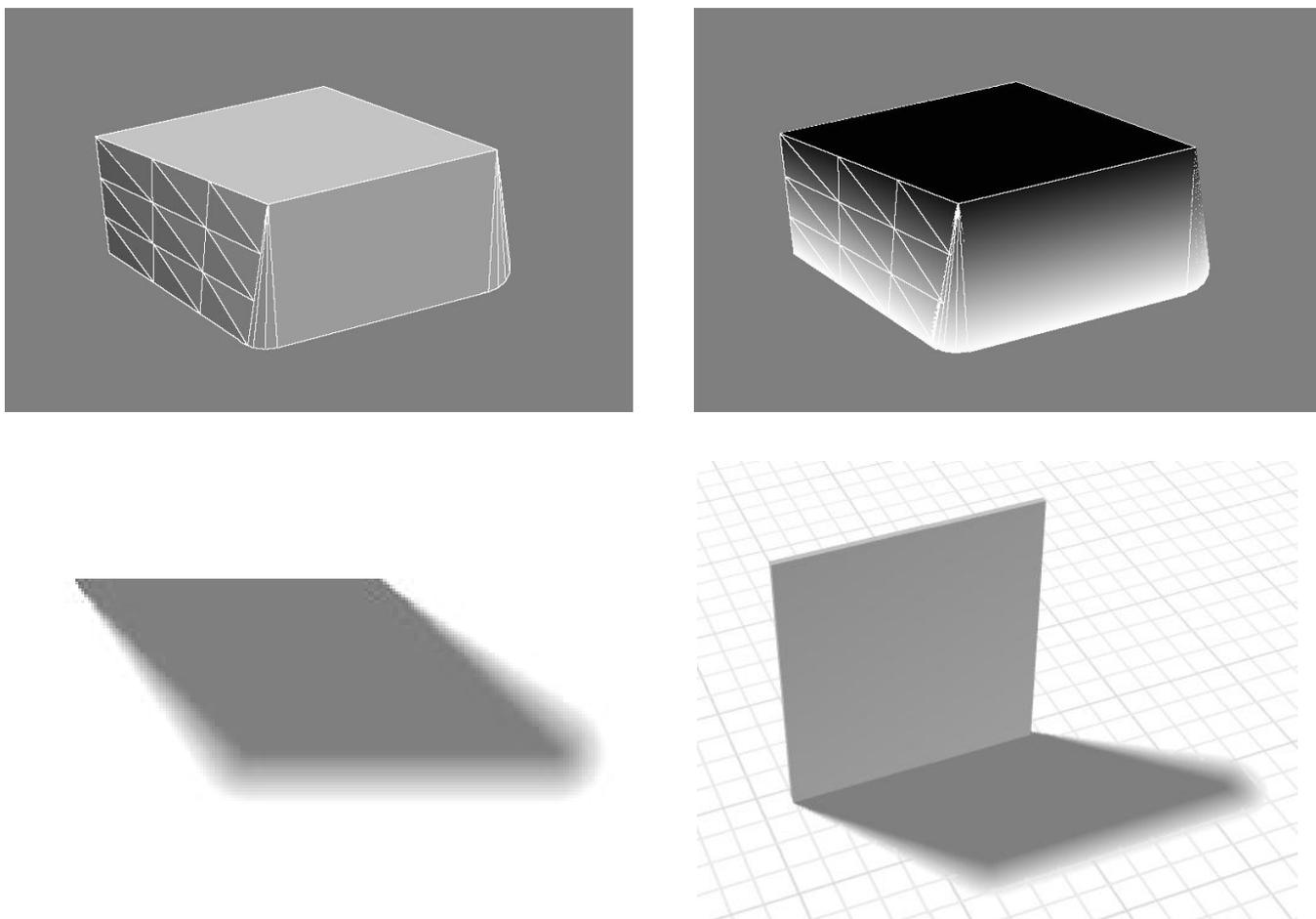


Figure 2: To ensure that the darkest gradient color covers overlapping lighter colors, the gradient circles are turned into cones and the quadrilaterals into 3D sheets. The upper left image shows a side view of the 3D object formed to create the shadow texture. The hard drop shadow forms the plateau's top, with cones and sheets forming the penumbra sides. The upper right image shows the gradient coloration applied to the plateau object's primitives. In the lower left this plateau object is then rendered from above, creating a soft shadow texture. The lower right is the final result, made by applying the shadow texture to the ground plane.

The resulting image is a reasonable approximation of a penumbra for the object (see Figure 3). This grayscale texture can be used as a projected lightmap to modify the illumination on the receiving

surfaces. Shadow receivers do not have to be planar, but the further removed they are from the target plane used to generate the shadow texture, the less convincing the effect.

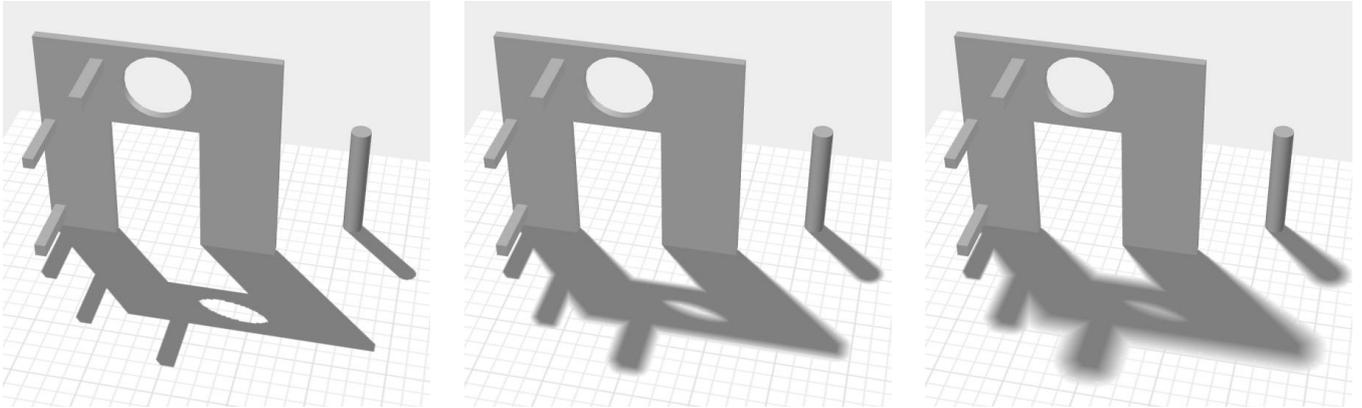


Figure 3: On the left is a hard shadow, the middle shows the effect of a small area light source, the right a larger light source.

3 Implementation

A few details are worth mentioning in controlling and implementing this algorithm on graphics hardware.

The radii of the cones generated depend on the height of the vertex above the ground plane times some constant factor. This factor is proportional to the size of the area light source. In most of the images in this paper a value of 0.1 was used, i.e. the penumbra's width is one tenth of the height of the object casting the shadow.

The basic algorithm makes the assumption that the light source is not near the horizon. As a light nears the horizon the penumbræ will become more non-uniform, i.e. they will become elongated. This stretching can be simulated by using and connecting cones with elliptical bases, with the major axis of the ellipse pointing towards the light.

Plateau objects do not have to be rendered to extend from the hither to yon; some renderers may not properly handle such large changes in z -depth. In reality, the number of bits needed for z -depths does not have to exceed the number of gray levels displayable, since the depths are only used to differentiate these levels.

Since each silhouette edge is processed separately, a shared vertex would normally generate its cone twice. A mailbox scheme can be used to render the cone for each silhouette vertex only once. When an edge generates two cones and a sheet, each vertex record used stores the current frame number. If another silhouette edge is encountered that would re-render one of these vertices, the vertex can be skipped if the current frame number is equal to its stored frame number.

Silhouette edges which are fully inside the hard shadow could be discarded, as they have no effect on the result. That is, only silhouette edges which mark a true, visible shadow boundary are needed by the algorithm. However, such edges are usually more effort than they are worth to detect; it is simpler to let all silhouette edges create cones and sheets. Simpler yet, all polygons (or, somewhat more efficiently, all polygons facing the light) in a shadow caster could be used to generate cones and the two sheets joining

the cones. While inefficient, this method has the advantage of needing no model connectivity information nor silhouette testing whatsoever.

Conversely, if full connectivity information is available, vertex cones could be trimmed in shape. By knowing the two silhouette edges sharing a vertex, some or all of the vertex cone's triangular faces are hidden and so do not need to be generated. For some (but not all) concave angles the cone is entirely hidden by the two edge sheets connected to it.

Sheets are generated by connecting two cones at their tangents. These tangent edges are easy to find when the cones have the same radius: the first vertex is at the tip of the cone, and the second vertex is on the base's edge at a point on a line perpendicular to the silhouette edge. When the cone radii differ, the tangent line between the two base circles is computed and the points used to generate the tangent edges [3]. See Figure 4.

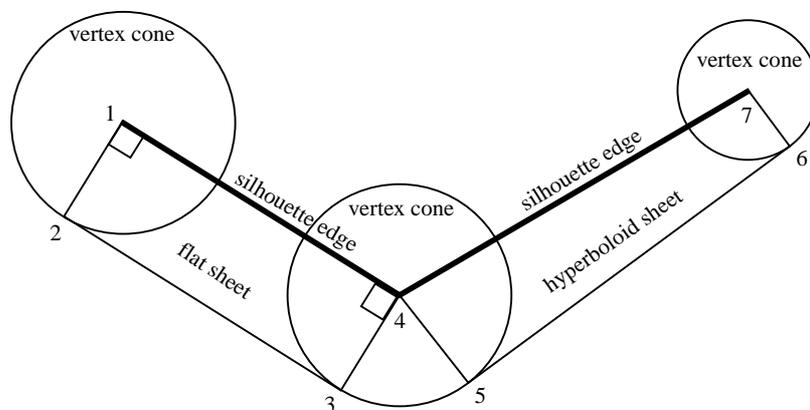


Figure 4: Sheet formation. If the cones have the same radius (i.e. the vertices are at the same height above the plane), the sheet 1–2–3–4 is formed by generating points along lines perpendicular to the original edge 1–4. If the radii differ, the sheet edges 4–5–6–7 are generated at the tangent points for line 5–6.

If only two triangles are used to render a hyperboloid sheet, the penumbra can contain artifacts. It is worth tessellating such sheets when the radii differ by a noticeable factor. In Figure 2 the right side of the plateau (shown in the left images) corresponds with the penumbra formed by the right edge of the standing slab. This sheet is tessellated in order to avoid warping artifacts due to Gouraud interpolation [13]. Such sheets are a type of Coons surface, bounded and defined by four straight lines. The surface is tessellated by linearly interpolating along each axis and generating the grid points.

A number of techniques can be used to shade the gradient polygons. Using a color per vertex is one method. Another is to turn linear fog on, going from darkest gray for the objects at the hither to full white at the yon. A third is to use the Z -buffer depth values directly as grayscale color. A fourth method is to apply a 1D grayscale texture to the plateau object. This last method allows other falloff patterns to be used in the penumbra. As Parker et al. [9] note, the intensity gradient of the penumbra for a disk area light is sinusoidal, not linear. So, by using a texture map which translates the height of plateau surfaces into a sine curve falloff, a more realistic penumbra can be achieved, while also avoiding Mach banding at the penumbra's borders. See Figure 5.

For simplicity, a projected shadow vertex is given a circular projection. Similar to Soler and Sillion's use of different filtering kernels to represent different light shapes [12], different geometric projectors might be used here. For example, instead of creating cones, pyramidal shapes with rounded corners might be used as a basis to represent rectangular area lights, or lozenge shapes might represent linear lights such as fluorescent tubes. No experiments have been done with such variants at this time, so it is unclear

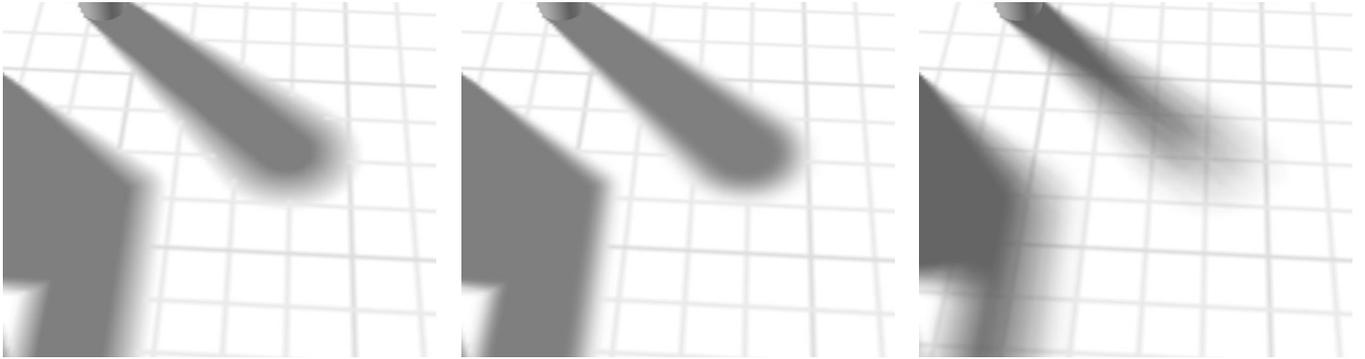


Figure 5: On the left the penumbra is formed with a linear gradient. The middle shows a sinusoidal drop off. On the right is a Heckbert and Herf style rendering for comparison.

whether this is a viable approach.

4 Analysis

The major drawback of the algorithm is that shadows always have fully dark umbra regions, similar to Parker et al. [9]. Each object will cast a hard shadow umbra, no matter how far it is from the ground plane. Using a proper sinusoidal falloff has the effect of making the umbra appear even larger overall (see Figure 5). Another artifact is that where silhouette edges meet to form a concavity there is a sharp, unnatural change in the penumbra (see Figure 3).

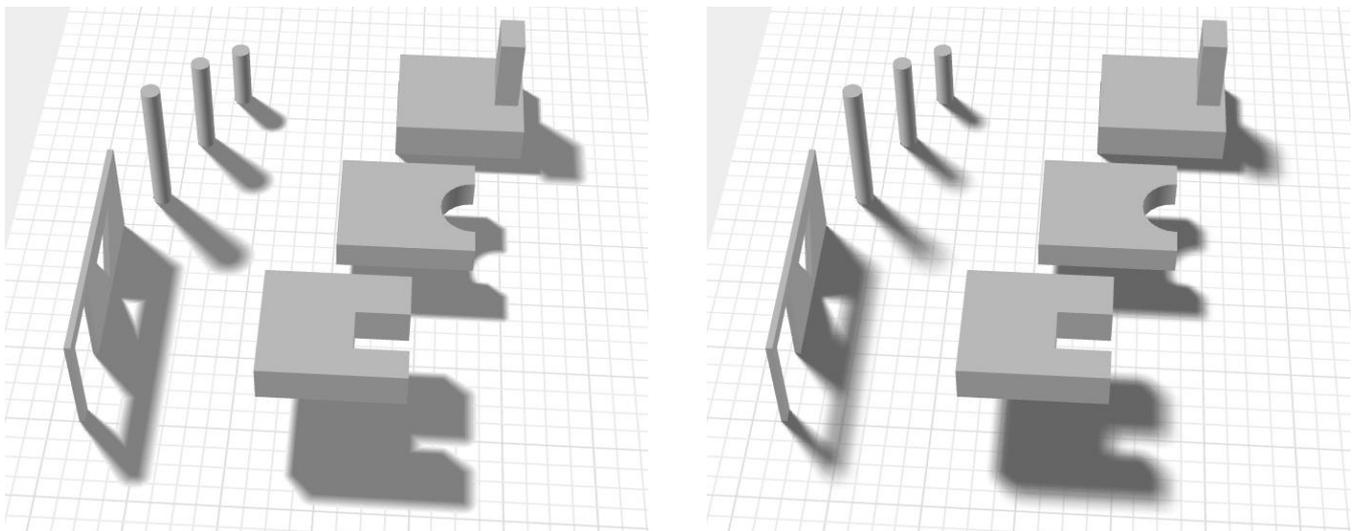


Figure 6: Plateau shadows on the left, Heckbert and Herf shadows on the right. A Monte-Carlo sampled disk area light source using 256 shadow passes was used for Heckbert and Herf.

Figure 6 shows comparable views of the same scene using plateaus and Heckbert and Herf's scheme. The image on the right is more physically correct and shows where the plateau scheme is a poor match. There are still some overlap artifacts visible when using the Heckbert and Herf algorithm, even with 256 passes (see the images on the web site).

5 Conclusions

There are a number of advantages of the plateau algorithm over other real-time rendering shadow techniques. The penumbra correctly varies with the object's distance from the plane, while creating no noise, Moiré, or overlap artifacts. Umbrae are larger than is physically correct, but overall the effect is perceptually convincing. The algorithm itself efficiently uses graphics hardware and costs relatively little extra time.

References

- [1] Blinn, Jim, "Me and My (Fake) Shadow," *IEEE Computer Graphics and Applications*, vol. 8, no. 1, pp. 82–86, January 1988. Also collected in [2].
- [2] Blinn, Jim, *Jim Blinn's Corner: A Trip Down the Graphics Pipeline*, Morgan Kaufmann Publishers, Inc., San Francisco, 1996.
- [3] Glassner, Andrew, "Useful 2D Geometry," Andrew S. Glassner, ed., *Graphics Gems*, Academic Press Inc., pp. 3–11, 1990.
- [4] Gooch, Bruce, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld, "Interactive Technical Illustration," *Proceedings 1999 Symposium on Interactive 3D Graphics*, pp. 31–38, April 1999. <http://www.cs.utah.edu/~bgooch/ITI/>
- [5] Heckbert, Paul S., and Michael Herf, *Simulating Soft Shadows with Graphics Hardware*, Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.
- [6] Herf, Michael, and Paul S. Heckbert, "Fast Soft Shadows," sketch in *Visual Proceedings (SIGGRAPH '96)*, p. 145, August 1996.
- [7] McReynolds, Tom, David Blythe, Brad Grantham, and Scott Nelson, "Advanced Graphics Programming Techniques Using OpenGL," *Course notes at SIGGRAPH '99*, 1999. <http://reality.sgi.com/blythe/sig99/>
- [8] Nguyen Hubert Huu, "Casting Shadows on Volumes," *Game Developer*, vol. 6, no. 3, pp. 44–53, March 1999.
- [9] Parker, Steven, Peter Shirley, and Brian Smits, "Single Sample Soft Shadows," Tech. Rep. UUCS-98-019, Computer Science Department, University of Utah, October 1998. <http://www.cs.utah.edu/~bes/papers/coneShadow/>
- [10] Sander, Pedro V., Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder, "Silhouette Clipping," *Proceedings of SIGGRAPH 2000*, pp. 327–334, July 2000. <http://www.deas.harvard.edu/~pvs/research/silclip/>
- [11] Segal, M., C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, "Fast Shadows and Lighting Effects Using Texture Mapping," *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, no. 2, pp. 249–252, July 1992.
- [12] Soler, Cyril, and François Sillion, "Fast Calculation of Soft Shadow Textures Using Convolution," *Computer Graphics (SIGGRAPH '98 Proceedings)*, pp. 321–332, July 1998. <http://www-imagis.imag.fr/Membres/Francois.Sillion/Papers/Index.html>

- [13] Woo, Andrew, Andrew Pearce, and Marc Ouellette, “It’s Really Not a Rendering Bug, You See...,” *IEEE Computer Graphics and Applications*, vol. 16, no. 5, pp. 21–25, September 1996.
<http://computer.org/cga/cg1996/g5toc.htm>