Eurographics Symposium on Rendering 2003
Per Christensen and Daniel Cohen-Or (Editors)

Penumbra Maps: Approximate Soft Shadows in Real-Time

Chris Wyman and Charles Hansen

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

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.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Soft shadows, shadow map, graphics hardware, shadow algorithms

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[21]. 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[6], shadow maps[23], 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[10] or merging hard shadows rendered from various points on the light[11]. 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[19] or adaptive shadow maps[8] to help reduce 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.

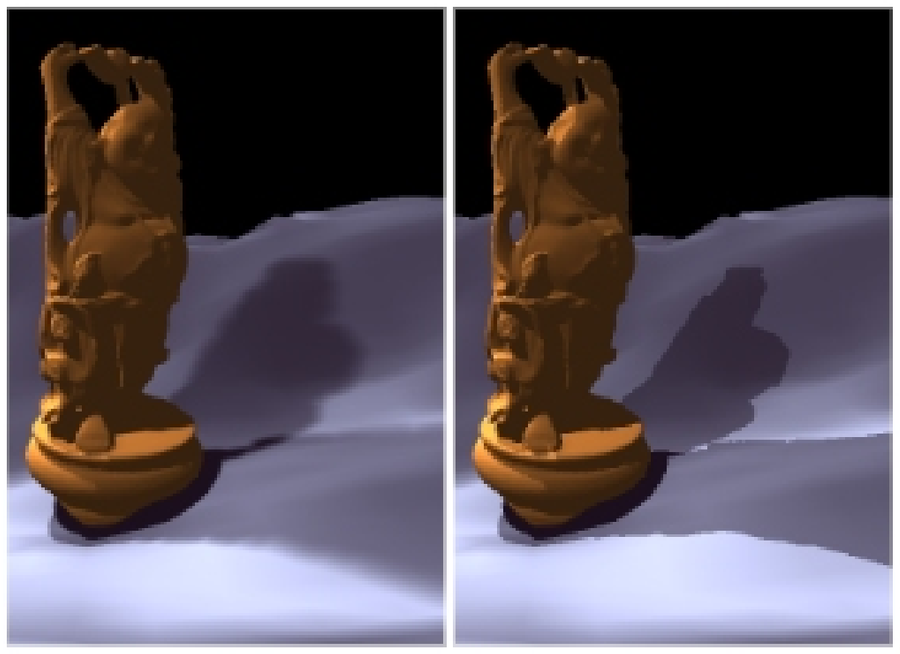

Figure 1: *With two penumbra maps, this scene runs at 11 fps for 1024x1024 images (left). Compare to shadow maps (right) which only render hard shadows.*

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.*[24] and Akenine-Möller and Haines[3] 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[18] convolve images of hard shadows and the light source to approximate soft shadows for nearly parallel configurations. Stark and Riesenfeld[20] use vertex tracing to compute exact shadows for polygonal scenes. Various backprojection techniques[7] can generate soft shadows via discontinuity meshing. Using layered depth images[1] allows rendering soft shadows interactivity, but moving lights or objects requires costly recomputation.

Parker *et al.*[15] 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[6] create a polygonal shadow model based on object silhouettes as seen from the light. Heidmann[12] implements this technique in hardware using a stencil buffer. Shadow mapping[23] 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.*[17] 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.*[16] introduce *percentage closer filtering*, which reduces aliasing by blurring the shadow map. This blurring can give the impression of softer shadows. Heidrich *et al.*[13] 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[9] 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.

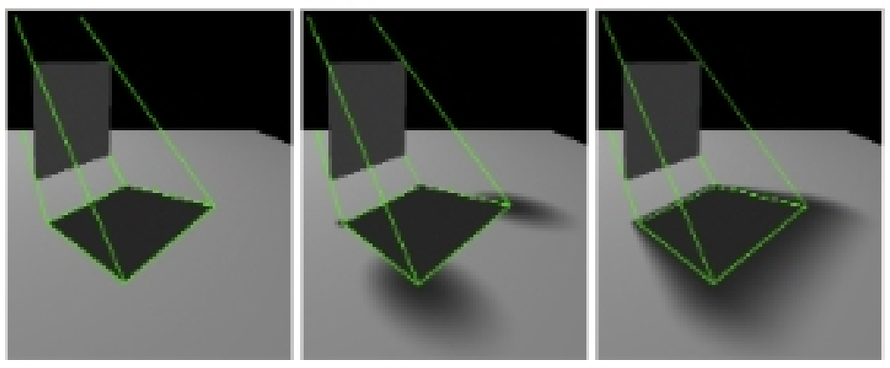

Figure 2: *Haines generates soft shadows by (left) rendering a hard shadow, (middle) rendering cones at each silhouette vertex, and (right) rendering sheets connecting the cones.*

Akenine-Möller and Assarsson[2] 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, non-closed 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.

Brabec and Seidel[4] 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[22], plausible soft shadows should suffice in interactive environments. Haines'[9] 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. These observations led to similar, concurrent work by Chan and Durand[5], allowing them to render approximate soft shadows using new geometric primitives called *smoothies*.

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.

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.
- $\forall v_i \in \mathcal{V}$, 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.

$$I = 1 - \frac{Z_P - Z_F}{Z_P - Z_{v_i}} = \frac{Z_F - Z_{v_i}}{Z_P - Z_{v_i}} \quad (1)$$

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.

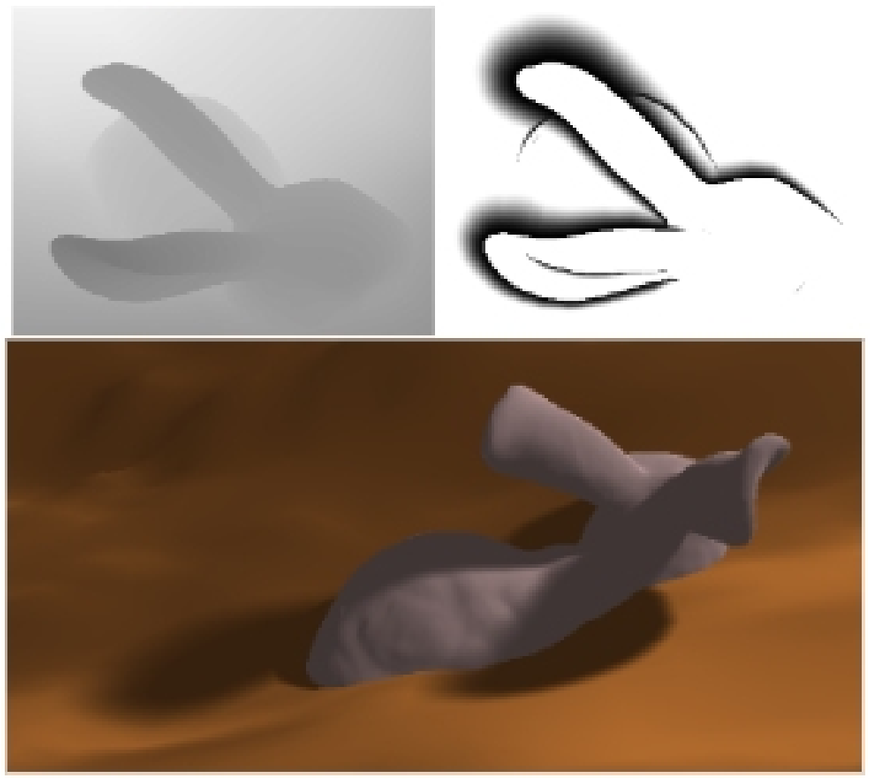

Figure 3: *An example shadow map (top left), corresponding penumbra map (top right), and the final rendered result.*

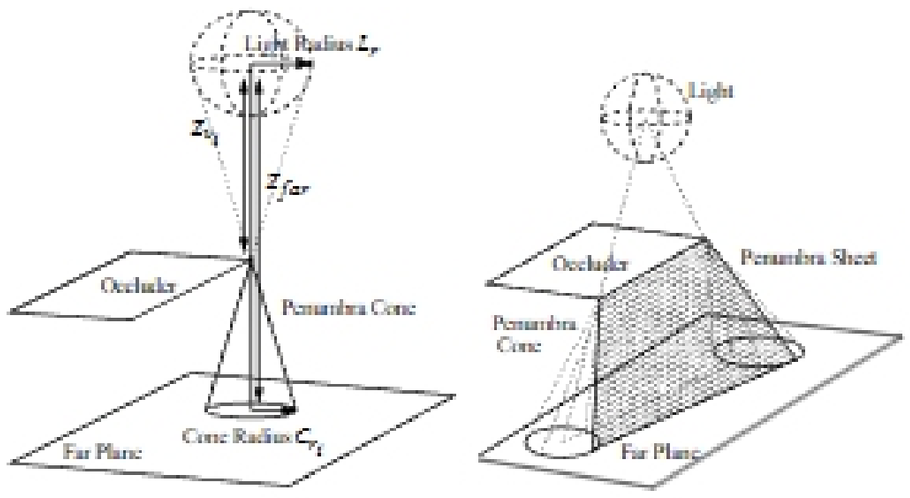

Figure 4: *Each cone's tip is located at a vertex v_i with the base located at the far plane (left). Using simple geometry, we compute the cone radius C_{r_i}. Each sheet (right) connects two adjacent cones.*