Single Sample Soft Shadows

Steven Parker Peter Shirley Brian Smits
University of Utah

October 27, 1998

Abstract

A simple extension to ray tracing is presented that creates visually plausible "soft" shadows with little extra computation. Although these soft shadows are approximate, they are robust and have penumbra widths that behave in a believable way, including accurate placement of singularities where penumbra width is zero. The method has continuous behavior in space and time, so it is appropriate for both static and dynamic image generation.

1 Introduction

As processing power increases, ray tracing becomes increasingly popular because of its clean mechanisms for shadowing and specular reflection. The basic algorithm has remained largely unchanged since it was introduced by Whitted [4]. One of ray tracings' chief limitations is the hard edges it computes for shadows. Soft edged shadows are preferred to hard shadows independent of aesthetics because soft shadows aid in accurate spatial perception [3]. Ray tracing methods that produce accurate soft shadows such as ray tracing with cones [1] or probabilistic ray tracing [2] stress accurate soft shadows, so they dramatically increase computation time relative to hard shadow computation. In this paper we introduce an inexpensive algorithm for soft shadows that can be plugged into a conventional ray tracer. The algorithm sacrifices accuracy to attain speed, but the important qualitative features of soft shadows are preserved, and only one shadow sample is needed per pixel.

2 Algorithmic Constraints

Our basic goal is to get the perceptual benefits of soft shadows without significantly increasing the runtime compared to Whitted-style shadow ray testing. This goal is achievable if some accuracy is sacrificed. However, to be both convincing and fast, approximate shadows must have three basic characteristics:

- Only one sample should be used per pixel/light.
- Shadow penumbra width should behave in a believable way, starting at zero at the occluder and increasing linearly with distance from the occluder.

- The algorithm should be visually smooth for both static and dynamic scenes.

It is generally accepted that it is hard for observers to tell the difference between shadows cast by differently shaped lights. For this reason we assume roughly spherical lights. We do a rough calculation at each illuminated point of what fraction s of the light is visible, and attenuate the unshadowed illumination by s. Thus our goal is to estimate s in way that is efficient and is consistent with the three requirements above.

3 Algorithm

Rather than creating a correct soft-edged shadow from an area source, the algorithm creates a shadow of a *soft-edged object* from a point source (Figure 1). The penumbra is the shadow of the semi-opaque (outer) object that is not also shadowed by the opaque (inner) object. The transparency of the outer object increases from no transparency next to the inner object to full transparency at the boundary of the outer object. For an isolated object, we can use inner and outer offsets of the real object to achieve believable results. We also need to make the intensity gradient in the penumbra natural. This can be achieved by computing a variable τ that begins at $\tau = 0$ on the penumbra/umbra boundary (the surface of the inner object) and increases linearly with distance to $\tau = 1$ on the outer boundary of the penumbra (the surface of the outer object). This can control an interpolation of the illumination attenuation function s. For diffuse spherical lights and occluders with a straight edge, the attenuation is a sinusoid: $s = (1 + sin(\pi\tau - \pi/2))/2$. To mimic this behavior with a polynomial we use the Bernstein interpolant: $s = 3\tau^2 - 2\tau^3$, which has the same values and derivatives as the sinusoid at $\tau = 0$ and $\tau = 1$.

While almost any inner and outer surfaces are practical for an isolated object, we would like our algorithm to remain simple and robust for multiple objects. For example, if a point is in the penumbra region for two objects we can make a composite attenuation factor based on the individual factors s_1 and s_2. Obvious candidates are addition: $s = 1 - ((1 - s_1) + (1 - s_2))$, multiplication: $s = s_1 s_2$, and thresholding: $s = \min(s_2, s_2)$. All will yield continuous intensity transitions and visually pleasing results for the shadow of two objects. However, thresholding is more conservative when many distinct objects are grazed by a shadow ray, resulting in shadows that are never darker than they should be.

The remaining important issue is how the inner and outer objects are generated for an input object. Figure 2 shows why the inner object must be at least as big as the input object to prevent "light leaks": any gap between the inner objects would result in a nonzero s and a lightening in the middle of the shadow. For this reason we use the object itself as the inner object. For the outer object, we use an image of the input object that is expanded in a direction natural for the particular primitive. For example, for a sphere, we use a larger sphere. For a polygon, we use a larger polygon. Our rationale for choosing the size of the outer object is shown in Figure 3. We would like the penumbra width in the approximation to be approximately W. Since $W \approx aD/(A-a)$ and $b/(A-a) = W/A$, a reasonable penumbra size will occur when $b = aD/A$. Note that the umbra region will be larger than in a physically-based computation, but when

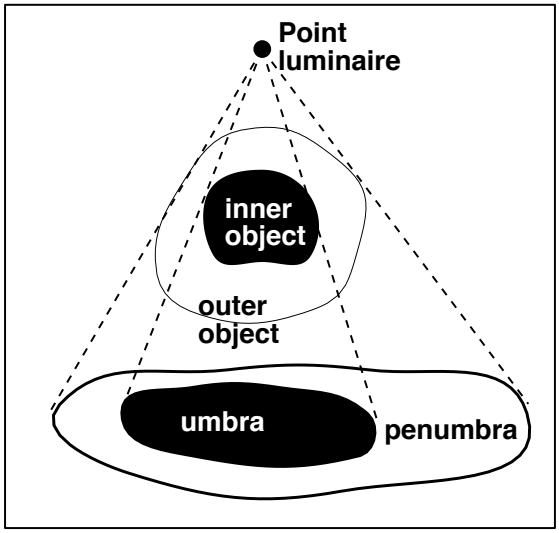

Figure 1: *The inner object is opaque and the outer object's opacity falls off toward its outer boundary.*

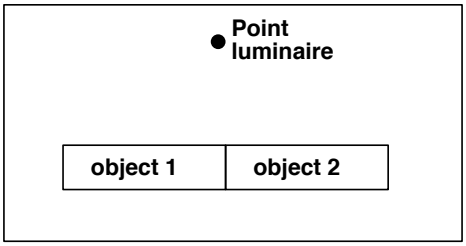

Figure 2: *For this case to work using local computations without light leaking between the objects requires that the inner objects be at least as large as the objects themselves.*

the occluder size is large relative to the light size, this should not be too noticeable. An example of the mechanics of the algorithm is discussed in the Appendix.

4 Results

The algorithm was implemented in a matter of hours in an existing ray tracer. The computations of b and the offset for the outer surface are only a few extra lines of code. Values for τ came out as a side-effect of the intersection computations. The ray tracer was approximately 20% slower after the change. Most of this change in runtime is caused by the increased number of shadow-ray/bounding-volume tests resulting from the addition of the outer objects. The results are shown in Figure 4. One reasonable concern about the algorithm is that its errors are highest for small objects whose real

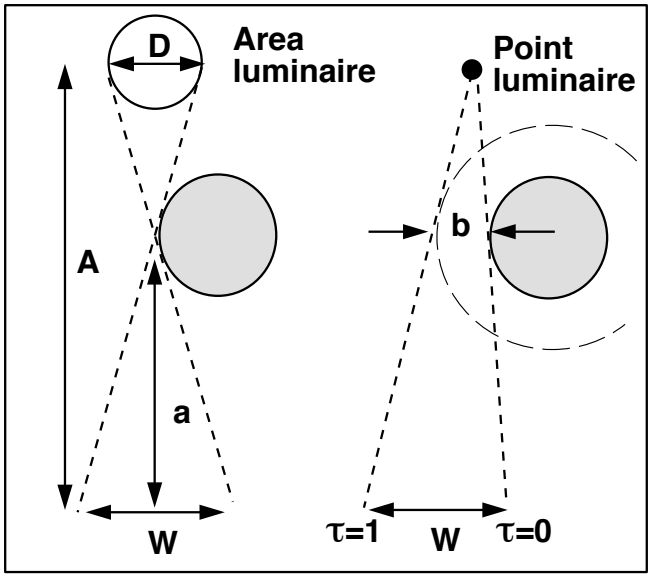

Figure 3: *Choosing the size of the outer object for a given configuration.*

shadows would have no umbra. In Figure 4 this is the case for the small spheres. In Figure 5 this is the case for the finely tesselated legs, and the shadows are too prominent. There are many plausible solutions to this problem, but even for the difficult cases shown in the figures the naive algorithm produces reasonable images. Note that fine tesselations only cause this problem if the entire tesselated object is small relative to the light.

There are no obvious dynamic artifacts in a real-time implementation of the soft-shadow algorithm on a 30 processor Origin 2000. The visual improvement caused by including soft shadows is even more apparent in the dynamic case. So although our algorithm is not designed to be accurate, it has large utilitarian benefits, particularly for animation and interactive applications. Because it does illumination and shading computations in software, it is also straightforward to use our method as a previewer with appropriate illumination levels and reflectance properties for global illumination programs.

Appendix: Example occluders

For a shadow ray $\mathbf{p} = \mathbf{o} + t\hat{\mathbf{v}}$ toward a light with position $\mathbf{l}$ and diameter D, and a sphere with center $\mathbf{c}$ and radius R, we need to decide whether we are in the penumbra region, and if so, what is the value of τ, the fractional distance between umbra and penumbra boundaries. We first compute the distance t_0 to the point on the shadow ray closest to $\mathbf{c}$: $t_0 = (\mathbf{c} - \mathbf{o}) \cdot \hat{\mathbf{v}}$. If t_0 is negative, then $s = 1$. We then compute b by

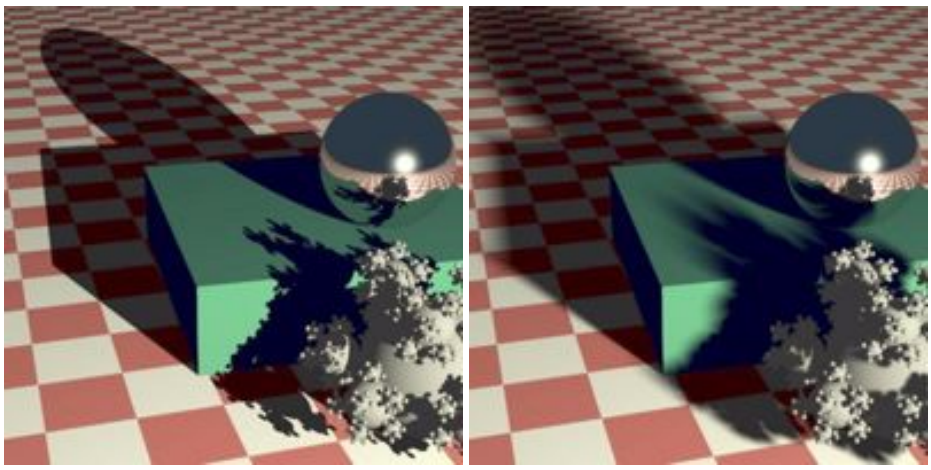

Figure 4: *Left: one sample per pixel with hard shadows. Right: one sample per pixel with soft shadows.*

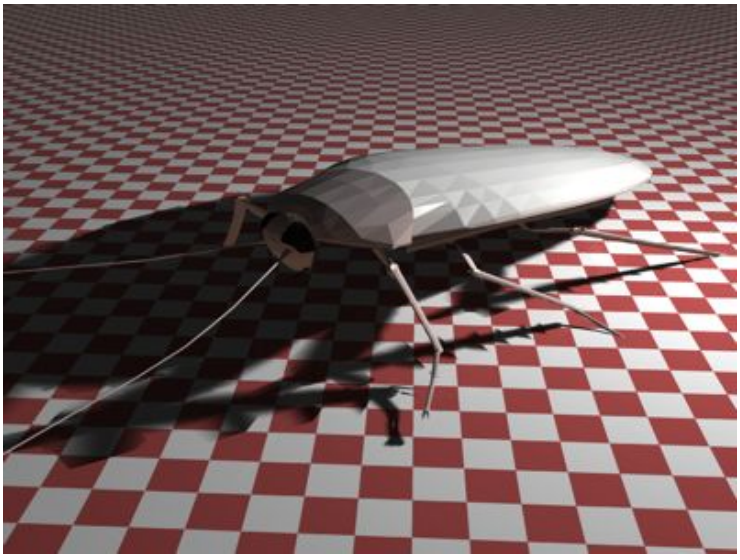

Figure 5: *Soft shadows on a tesselated model.*

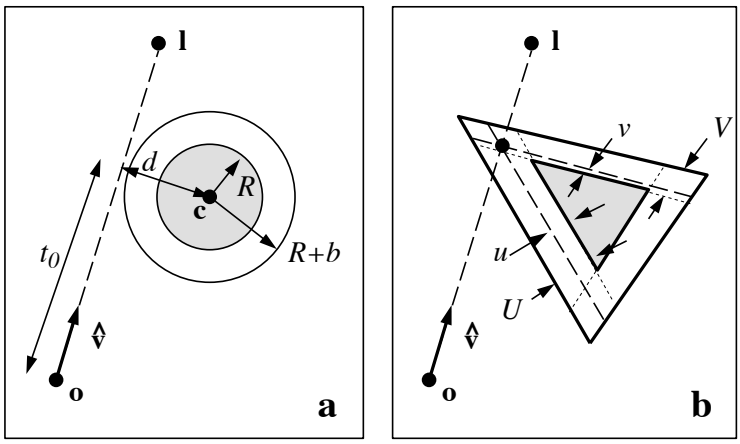

Figure 6: *a: geometry for spherical occulder. b) geometry for triangular occluder.*

assuming $a \approx t_0$, $A \approx ||\mathbf{l} - \mathbf{o}||$, so we use

$$b = \frac{Dt_0}{||\mathbf{l} - \mathbf{o}||}.$$

We compute the value of the minimum distance d from $\mathbf{c}$ to the ray:

$$d = ||t_0 \hat{\mathbf{v}} - \mathbf{c} + \mathbf{o}||.$$

If this distance is between R and $R + b$ then we compute $\tau = (d - R)/b$ and then compute $s(\tau)$. If $d < R$, $s = 0$, and if $d > R + b$, $s = 1$. The radius of the bounding volume for the sphere for shadow ray testing is $R + b_{max}$ where b_{max} is a function of the largest light and cannot be larger than D_{max}.

For triangles, the outer object is a triangle with rounded corners. The offset is different for each side of the triangle and is proportional to the cosine of projected triangle normal and the vector $\hat{\mathbf{v}}$.

References

[1] J. Amanatides. Ray tracing with cones. *Computer Graphics*, pages 129–135, July 1984. ACM Siggraph '84 Conference Proceedings.

[2] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(4):165–174, July 1984. ACM Siggraph '84 Conference Proceedings.

[3] D. Kersten, D. C. Knill, Mamassian P, and I. Bülthoff. Illusory motion from shadows. *Nature*, 379:31, 1996.

[4] T. Whitted. An improved illumination model for shaded display. *CACM*, 23(6):343–349, June 1980.