Ambient Aperture Lighting

Christopher Oat
3D Application Research Group, AMD

Pedro V. Sander
Hong Kong University of Science and Technology

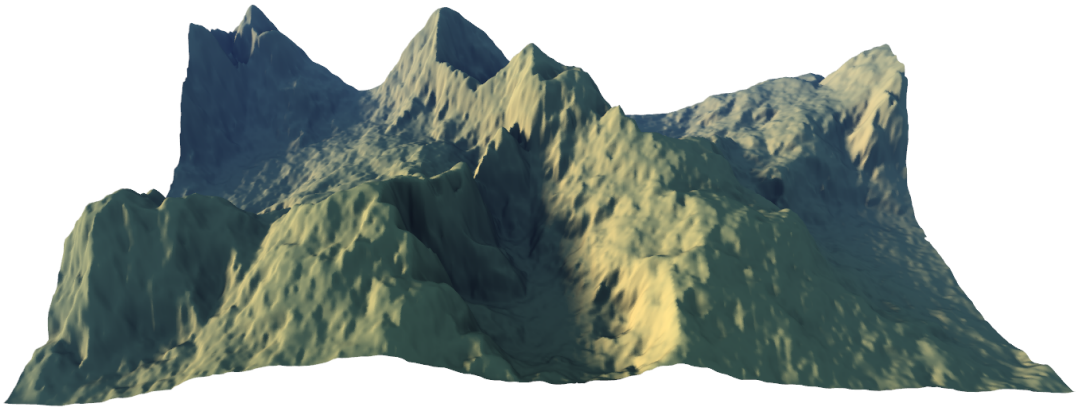


Figure 1: A terrain is rendered in real-time and shaded using the ambient aperture lighting technique described in this paper.

Abstract

This paper introduces a new real-time shading model that uses spherical cap intersections to approximate a surface's incident lighting from dynamic area light sources. Our method uses precomputed visibility information for static meshes to compute illumination with approximate high-frequency shadows in a single rendering pass. Because this technique relies on precomputed visibility data, the mesh is assumed to be static at render time. Due to its high efficiency and low memory footprint this method is highly suitable for games.

1 Introduction and related work

The problem of lighting complex scenes realistically and efficiently is of vital importance to real-time computer graphics applications. In this paper we present a new shading model that approximates incident lighting from a dynamic area light source. This method provides approximate high-frequency shadows for static models while only requiring a single rendering pass. We achieve this by approximating both the light source and surface visibility using spherical caps. At render time, we compute their intersection to estimate the amount of incident light. The ambient aperture shading model was developed with real-time terrain rendering in mind (see Figure 1 for an example), but it may be used for other applications where fast, approximate lighting from dynamic area light sources is desired.

There are several methods for determining whether a particular point on the surface is in shadow. Shadow mapping [Wil78] is a very common approach to perform such computation in real-time applications. However, shadow mapping large terrains is frequently impractical because it requires transforming and rasterizing the terrain data multiple times per frame. Additionally, shadow maps are ill suited for terrain rendering as they do not allow for area light sources such as the sun and the sky. Finally, shadow maps exhibit aliasing artifacts, particularly when used with large scenes such as terrains. Horizon mapping techniques for rendering self-shadowing terrains in real-time only allow point or directional light sources [Max88]. Precomputed Radiance Transfer (PRT) allows for area light sources but assumes low-frequency lighting environments [SKS02].

Our approach allows for object self-shadowing from dynamic area light sources with the caveat that the object is not deformable (only rigid transformations are supported). Our technique stores a small amount of precomputed visibility data for each point on the mesh. The visibility data approximates contiguous regions of visibility over a point's upper hemisphere using a spherical cap (Figure 2). This spherical cap acts as a circular aperture for incident lighting. The aperture is oriented on the hemisphere in the direction of average visibility and prevents light originating from occluded directions from reaching the surface point being rendered. At render time, area light sources are projected onto the point's upper hemisphere and are also approximated as spherical caps. Direct lighting from the area light source is computed by determining the area of intersection between the visible aperture's spherical cap and the area light's spherical cap. The overall diffuse response at the point is computed by combining the area of intersection with a diffuse falloff term.

The algorithm comprises of two stages. In a preprocessing stage, a contiguous circular region of visibility is approximated for each point on the surface of the model (Section 2). The data can be computed per texel and stored in a texture map, or per vertex and incorporated into the model's vertex buffer. Then, at render time, a pixel shader computes the amount of light that enters this region of visibility, and uses this result to shade the model (Section 3).

2 Preprocessing

Given a surface point – represented by a vertex on a mesh or by a texel in the texture domain – we wish to find its approximate visible region. The visible area at a point p is found by integrating a visibility function over the hemisphere. The visibility function evaluates to 1 for rays that do

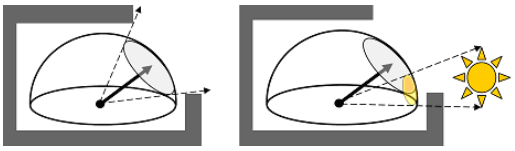


Figure 2: A spherical cap approximates visibility at a surface point (left). The visibility region acts as an aperture which blocks light from occluded directions (right).

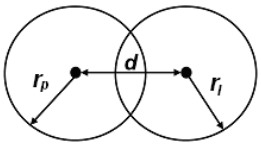


Figure 3: The intersection area of two spherical caps is a function of the arc length of their radii (r_p, r_l) and arc length of the distance d between their centroids.

not intersect the scene and to 0 otherwise. The percentage of visibility is then scaled by the area of a unit hemisphere and results in the solid angle of the visible region:

$$A_p = 2\pi \int_\Omega V(p,\omega)\ d\omega$$

This visible area is used as our aperture area. We do not store the area directly but instead store the arc length of a spherical cap of equivalent area, which is used at render time:

$$r_p = \cos^{-1}\left(\frac{-A_p}{2\pi} + 1\right)$$

The spherical cap's radius is a single floating point number that must be stored with the mesh (either per-vertex or per-texel).

The visible region's orientation on the hemisphere is determined by finding the average direction for which the visibility function evaluates to 1 (*a.k.a.* bent normal):

$$\vec{V_p} = \int_\Omega V(p,\omega)\omega\ d\omega$$

This gives us an average direction of visibility and serves as our aperture's orientation on the surrounding hemisphere. This direction is stored as a three component vector.

3 Rendering

At render time, for each pixel, the aperture visibility data is retrieved from a texture look-up or passed in per vertex and interpolated by the rasterizer. For highly tesselated models such as the terrain from Figures 1 and 6, storing the information per vertex produces acceptable results.

Next we describe our rendering algorithm which, for each pixel, seeks to compute the intersection between the precomputed circular visibility region and the circular light source (projected area light). Then, we describe an optimization which significantly reduces computation time while achieving nearly identical qualitative results.

3.1 Intersecting spherical caps

For each pixel, the spherical area light source is projected onto the hemisphere and the radius of the light's enclosing spherical cap is computed. The amount of light that reaches a point is determined by computing the area of intersection between the precomputed spherical cap representing the

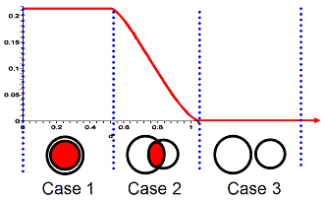


Figure 4: Spherical cap intersection as a function of distance between spherical cap centers. Case 1: One cap entirely overlaps the other, full intersection occurs. Case 2: The full intersection function is evaluated to find the area of intersection between partially overlapping spherical caps. Case 3: Their intersection area is zero.

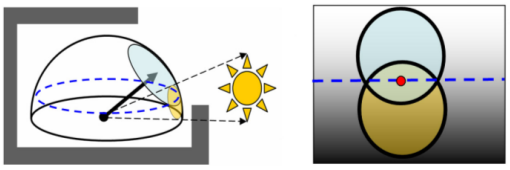


Figure 5: The aperture's spherical cap is intersected with a light source's spherical cap (left). A plot of the two spherical caps along with the Lambertian falloff function (right).

aperture of visibility, and the spherical light source as shown in Figure 3. The area of intersection for two spherical caps is computed as follows:

$$\begin{cases} 2\pi(1-\cos(\min(r_p,r_l))) & \text{if } \min(r_p,r_l) \le \max(r_p,r_l) - d, \\ 0 & \text{if } r_p + r_l \le d, \\ L(d,r_p,r_l) & \text{otherwise} \end{cases}$$

where $d = \cos^{-1}(V_p \cdot V_l)$, $\vec{V_p}$ and r_p are the precomputed parameters described in the previous section, and $\vec{V_l}$ and r_l are light vector and light source radius, respectively. The three cases of the piecewise function above represent full intersection, no intersection, and partial intersection. In the case of full intersection, the intersection area is the area of the smaller of the two caps. In the case of partial intersection the intersection area is defined by

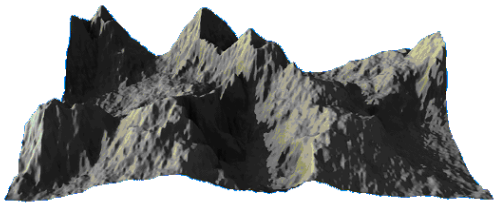
$$\begin{aligned} L(d,r_p,r_l) &= 2\pi - 2\pi\cos(r_p) - 2\pi\cos(r_l) \\ &- 2\cos^{-1}\left(\frac{\cos(d) - \cos(r_p)\cos(r_l)}{sin(r_p)\sin(r_l)}\right) \\ &+ 2\cos(r_p)\cos^{-1}\left(\frac{-\cos(r_l) + \cos(d)\cos(r_p)}{sin(d)\sin(r_p)}\right) \\ &+ 2\cos(r_l)\cos^{-1}\left(\frac{-\cos(r_p) + \cos(d)\cos(r_l)}{sin(d)\sin(r_l)}\right) \end{aligned}$$

which is a simplified form of the result from [TV01].

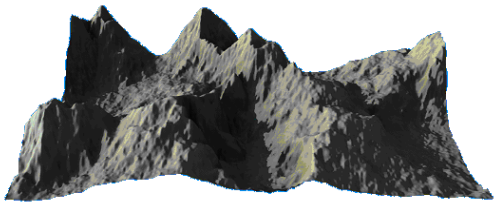
Once the area of intersection is found, the net diffuse response is approximated by scaling the area of intersection by a Lambertian coefficient (described in Section 3.3).

3.2 Optimization

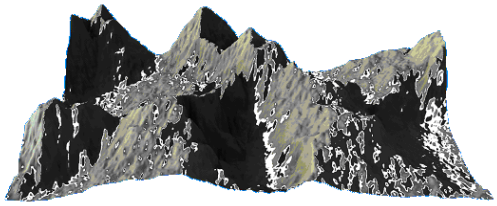
The spherical cap intersection function $L(d,r_p,r_l)$ may be too expensive to solve directly. Instead of solving this expensive function, it may be desirable to use a less expensive approximation. We considered various approximation functions. Because the intersection function exhibits a smooth falloff with respect to increasing cap distance (Figure 4), the smoothstep() function is an appropriate approximation:



(a) Exact intersection (22fps)



(b) Approximate intersection (39fps)



(c) Color-coded penumbra region

Figure 6: Comparison between the exact and approximate intersection computation on a 300,000 triangle terrain model.

$$(2\pi - 2\pi\cos(\min(r_p, r_l))) \; * \\ \text{smoothstep}\left(0, 1, 1 - \frac{d - |r_p - r_l|}{r_p + r_l - |r_p - r_l|}\right)$$

where

$$\text{smoothstep}(a, b, x) = -2\left(\frac{x-a}{b-a}\right)^3 + 3\left(\frac{x-a}{b-a}\right)^2$$

The smoothstep(a, b, x) function makes a gradual transition from 0 to 1 between the values a and b, and has a slope of 0 at these threshold points. It returns 1 when a full intersection occurs and 0 when no intersection occurs.

The result of smoothstep() is scaled by the area of the smaller spherical cap so that the end result is a smooth transition between full intersection and no intersection.

Optimization results Empirically, we did not notice any significant qualitative difference between using this approximation and the actual intersection function. Figure 6 compares the exact intersection function and its approximation. We used an ATI Radeon X800 and a screen resolution of 1024×768 for this comparison. Note that the visual difference is insignificant and the approximate result is 77% faster on a mainstream GPU.

3.3 Lighting computation

Direct lighting Knowing the area of intersection between the aperture's spherical cap and the light source's spherical cap is not enough to compute lighting at a surface point. It simply represents how much unoccluded light arrives at the point. A given area of intersection results in approximate diffuse reflection depending on its orientation relative to the point's surface normal. Therefore we must take Lambert's Cosine Law into account. The exact solution would involve convolving the intersection area with a cosine kernel, but this would be too expensive to compute. Instead, we account for the Lambertian by first finding a vector from the point we are shading to the intersection region's centroid (Figure 5):

$$\begin{aligned} w &= \frac{r_l - r_p + d}{2d} \\ \vec{V_i} &= (w)\vec{V_p} + (1-w)\vec{V_l} \end{aligned}$$

We then compute the dot product between this intersection vector and the point's geometric normal. This dot product is used to scale the incoming light and it results in a Lambertian falloff as the area of intersection approaches the horizon. This approach assumes a constant Lambertian term in the area of intersection and is a reasonable approximation for light sources that cover a small portion of the hemisphere, such as the sun.

We have also considered using a 2D lookup table that contains the integrated Lambertian over the spherical cap for varying altitudes and radii, as described in [KL05]. However, as opposed to their application which uses this method to add shadows, we use this result to add light from small light sources. For our application, we have found that by simply using the centroid approximation we get nearly identical results for reasonable light source radii, such as that of the sun.

Ambient lighting The above result gives us a nice approximation of high-frequency, direct, diffuse lighting on our mesh but it does not handle indirect lighting. For outdoor lighting scenarios, indirect light scattering into our aperture could be significant and this needs to be accounted for in our shading model. The indirect sky light can be approximated by filling the empty portion of our aperture with ambient light from the sky. Once the area of the light-aperture intersection is found, we can subtract this area from the aperture's total area to determine how much of the aperture is not covered by the sun. We can use an average sky color for our ambient light or another suitable approximation. This method's advantage is that it does not destroy scene contrast by adding ambient light in areas that really should be dark because they are mostly occluded from the outside world.

Lighting results Figure 7 shows our approach compared to PRT with 6th order spherical harmonic coefficients, bent normal, and the exact result, computed by ray tracing. The first three methods have relatively low storage requirements and only require a single pass on the geometry at runtime, thus being very practical. Note that while our spherical cap approximation is not strictly correct in many cases, such as this cylinder example (Figure 7, left), it does yield results that have more realistic and well defined shadows when compared to PRT and bent normal. It is specially useful for terrain scenes (Figure 7, right).

4 Limitations

Ambient aperture lighting makes simplifying assumptions and approximations in order to reduce lighting computations when rendering with dynamic area light sources. There are certain cases where our assumptions break down and the shading model produces artifacts such as incorrect shadowing. The most significant assumption is that the visible region for any point on a mesh is both contiguous and circular. This is generally a good approximation for terrains, but it

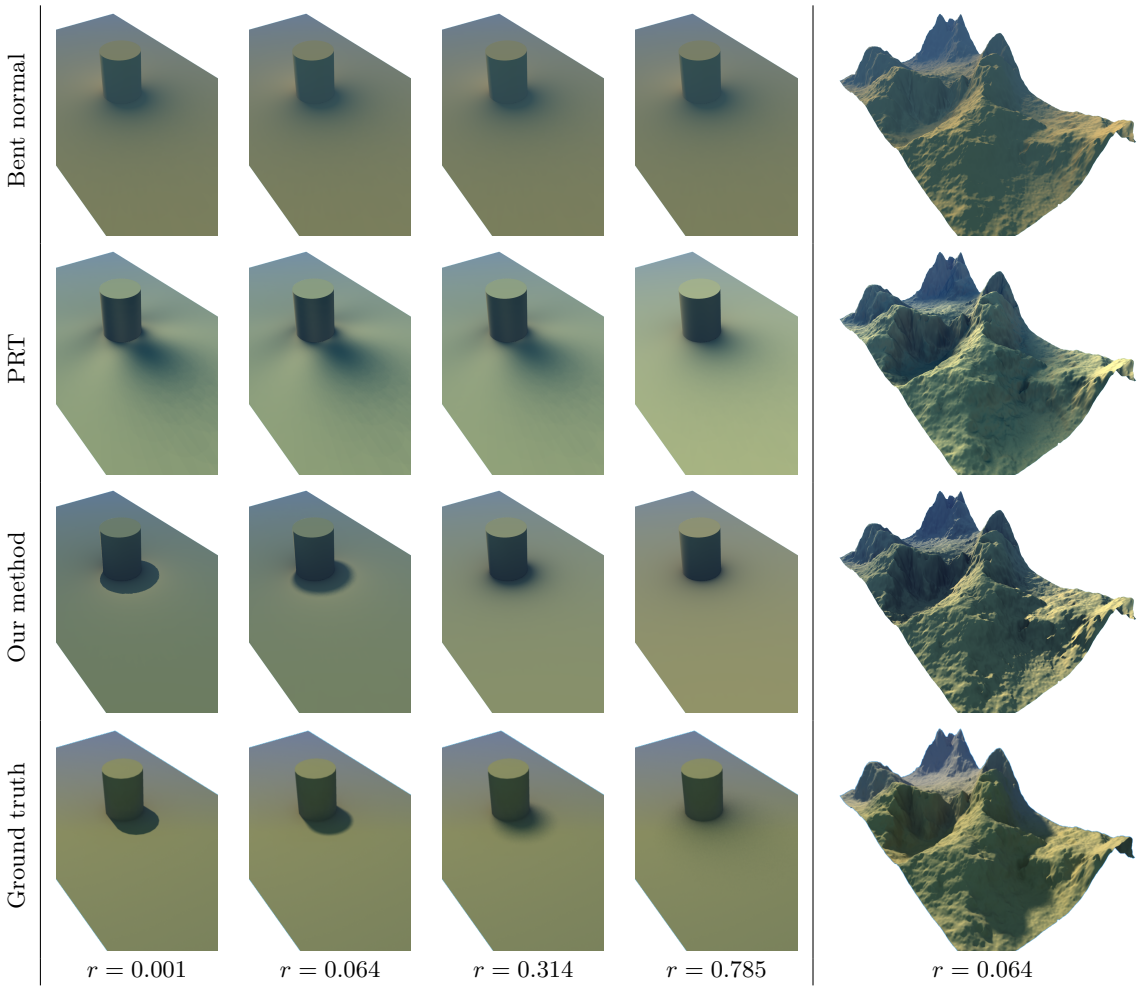


Figure 7: A simple didactic scene illuminated by spherical light sources of varying sizes using our ambient aperture approach, bent normal, PRT with 6th order spherical harmonic coefficients and ground truth (left). The same comparison for a terrain mesh using a light source that is roughly the size of the sun (right). Note that when using the bent normal method, the light source size does not affect the result of the computation.

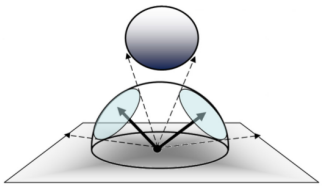


Figure 8: The visible region is a band around the horizon which cannot be closely approximated by a spherical cap.

fails to handle other cases such as that of a sphere over a plane (Figure 8).

5 Summary and future work

A real-time shading model that allows for dynamic area light sources was presented. Our method provides approximate high-frequency shadows while only requiring a single rendering pass of the geometry. This technique is well suited for outdoor environments lit by dynamic spherical area light sources (such as the sun), and is particularly useful for real-time terrain rendering. Ambient aperture lighting makes several simplifying assumptions and mathematical approximations to reduce the computational complexity and storage costs associated with other real-time techniques that allow for dynamic area light sources.

Acknowledgements

We would like the thank the members of AMD's 3D Application Research Group for very fruitful discussions.

References

Kontkanen J., Laine S.: Ambient occlusion fields. In *ACM Symposium on Interactive 3D Graphics and Games* (2005).

Max N. L.: Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer 4*, 2 (1988), 109–117.

Sloan P.-P. J., Kautz J., Snyder J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3 (2002), 527–536.

Tovchigrechko A., Vakser I. A.: How common is the funnel-like energy landscape in protein-protein interactions? *Protein Sci. 10* (2001), 1572–1583.

Williams L.: Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1978), ACM Press, pp. 270–274.