Opacity Shadow Maps

Tae-Yong Kim Ulrich Neumann
taeyongk@usc.edu uneumann@usc.edu
Computer Graphics and Immersive Technology Laboratory
Integrated Media Systems Center
University of Southern California

Abstract

Opacity shadow maps approximate light transmittance inside a complex volume with a set of planar opacity maps. A volume made of standard primitives (points, lines, and polygons) is sliced and rendered with graphics hardware to each opacity map that stores alpha values instead of traditionally used depth values. The alpha values are sampled in the maps enclosing each primitive point and interpolated for shadow computation. The algorithm is memory efficient and extensively exploits existing graphics hardware. The method is suited for generation of self-shadows in discontinuous volumes with explicit geometry, such as foliage, fur, and hairs. Continuous volumes such as clouds and smoke may also benefit from the approach.

1. Introduction

Rendering self-shadows inside volumetric objects (hair, fur, smoke, and cloud) is an important but hard problem. Unlike solid objects, a dense volume made of many small particles exhibits complex light propagation patterns. Each particle transmits and scatters rather than fully blocks the incoming rays. The strong forward scattering properties as well as the complex underlying geometry make the shadow generation difficult. However, self-shadows are crucial to capture effects such as backlighting.

Two techniques are generally used for volumetric shadows[1]; shadow maps [10, 12, 13] and ray tracing [5, 6]. In traditional depth-based shadow maps (DBSM), the scene is rendered from the light's point of view and the depth values are stored. Each point to be shadowed is projected to the light's camera and the point's depth is checked against the depth in the shadow map. In ray tracing, a ray is shot from a scene point to the light. If the ray intersects any particle on its way, shadows are detected and accumulated. Despite its accuracy, a complete ray tracing of dense volumetric objects can be prohibitive in terms of rendering time. In practice, shadow maps are often used in conjunction with ray tracing for efficiency.

A good property of DBSM is that it can be accelerated with hardware by rendering the scene and storing the depth buffer. However, severe aliasing artifacts can occur with small semi-transparent objects. In a dense volume made of small primitives, depths can vary radically over small changes in image space. The discrete nature of depth sampling limits DBSM in handling such objects. Moreover, small particles are often semi-transparent due to forward scattering. The binary decision in depth testing inherently precludes such transparency. Thus, DBSM is unsuited for dense volumes.

Rendering volumetric objects requires a precise measure of a transmittance function for any point in space. The transmittance $\boldsymbol{\tau}(p)$ of a light to a point p can be written as

[1] For other types of shadows, refer to [14] and survey sections in [10, 15] for more recent ones.

$$\tau(p) = \exp(-\Omega), \text{ where } \Omega = \int_0^l \rho(l')dl' \quad (1)$$

In (1), l is the optical depth from the light to the point, ρ is an extinction (or a density) function along the path [1, 6, 10], and Ω is the *opacity* value at the point[2].

In the deep shadow maps (DSM) [10], each pixel stores a piecewise linear approximation of the transmittance function instead of a single depth, yielding more precise shadow computation than DBSM. Despite the compactness and quality, however, DSM requires a significant amount of data initialization time. When the volume changes in time with regard to the light (e.g. hair animation or moving lights), the generation cost can cancel out the computational benefit of the algorithm.

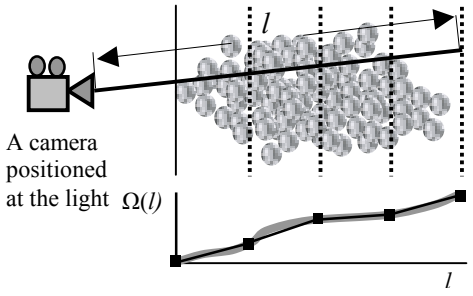


Fig. 1. The opacity function Ω*(l)* shown in a solid gray curve is approximated by a set of opacity maps.

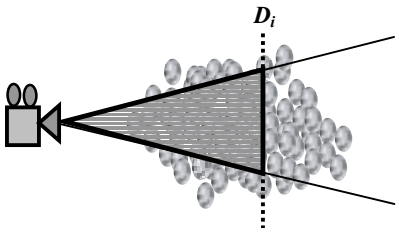


Fig. 2. The volume is rendered on each opacity map, clipped by the map's depth (D_i). The transparent region illustrates the clipped volume.

Opacity shadow maps (OSM) use a set of parallel opacity maps oriented perpendicular to the light's direction (Figure 1). By approximating the transmittance function with discrete planar maps, opacity maps can be efficiently generated with hardware. On each opacity map, the scene is rendered from the light's point of view, clipped by the map's depth (Figure 2). Instead of storing depth values, each pixel stores Ω, the line integral of densities along the path from the light to the pixel. The opacity values from adjacent maps are sampled and interpolated during rendering.

Although OSM resembles volume rendering [3, 8], a major distinction is that OSM uses a volume of explicit geometry primitives (points, lines, and polygons) instead of a sampled scalar-field (voxels). Thus, it does not incur high memory requirements for a full voxel data array. Also, OSM maintains the object space accuracy of the scene model unlike volume rendering where precision is subject to sampling density.

The idea of opacity maps was exploited in feature films. For example, in the movie *'Mission to Mars'*, a software renderer was used to render the sand-vortex sequence using an SGI origin machine with 16 processors [9]. These attempts did not consider hardware acceleration and hence the rendering took a substantial amount of time (about an hour per frame). To our knowledge, none of these techniques were published.

---

[2] For brevity, we use the term *opacity* to denote the accumulative extinction function. Note that the term opacity is often used to denote the actual shadow. Thus, more precise term for Ω will be *opacity thickness* as described in [11].

2. Algorithm

Opacity shadow maps heavily rely on graphics hardware and operate on any bounded volumes represented by standard primitives such as points, lines and polygons. (For example, hairs can be represented as a cluster of lines.) The volume is sliced with a set of opacity map planes perpendicular to the light's direction. The scene is rendered to the alpha buffer, clipped by each map's depth. Each primitive contributes its associated alpha value.[3] Each pixel in the map stores an alpha value that approximates the opacity relative to the light at the pixel's position. The opacity values of adjacent maps are sampled and linearly interpolated at the position of each shadow computation point, to be used in a shadowed shading calculation.

The pseudo code below uses the following notation. P is the set of all the shadow computation sample points (or simply *shadow samples*). N is the number of maps and M is the number of shadow samples. D_i is the distance from the opacity map plane to the light ($1 \le i \le N$). P_i is a set of shadow samples that reside between D_i and D_{i-1}. p_j is j_{th} shadow sample ($1 \le j \le M$). *Depth*(p) returns a distance from p to the light. $\Omega(p_j)$ stores the opacity at p_j. $\tau(p_j)$ is the transmittance at p_j and $\Phi(p_j)$ is the computed shadow. B_{prev} and $B_{current}$ are the previous and current opacity map buffers.

Pseudo Code

1. D_0= Min (*Depth*($\mathbf{p_j}$)) for all $\mathbf{p_j}$ in **P** ($1 \le j \le M$)
2. *for* ($1 \le i \le N$) *(Loop 1)*
3. Determine the opacity map's depth $\boldsymbol{D_i}$ from the light (Figure 3).
4. *for* each shadow sample point $\mathbf{p_j}$ in **P** ($1 \le j \le M$) *(Loop 2)*
5. Find i such that $\boldsymbol{D_{i-1}} \le$ *Depth*($\mathbf{p_j}$) < $\boldsymbol{D_i}$
6. Add the point $\mathbf{p_j}$ to $\mathbf{P_i}$.
7. Clear the alpha buffer and the opacity maps $\boldsymbol{B_{prev}}$, $\boldsymbol{B_{current}}$.
8. *for* ($1 \le i \le N$) *(Loop 3)*
9. Swap $\boldsymbol{B_{prev}}$ and $\boldsymbol{B_{current}}$.
10. Render the scene clipping it with $\boldsymbol{D_{i-1}}$ and $\boldsymbol{D_i}$.
11. Read back the alpha buffer to $\boldsymbol{B_{current}}$.
12. *for* each shadow sample point $\mathbf{p_k}$ in $\mathbf{P_i}$ *(Loop 4)*
13. $\boldsymbol{\Omega}_{prev}$ = ***sample***($\boldsymbol{B_{prev}}$, $\mathbf{p_k}$)
14. $\boldsymbol{\Omega}_{current}$ = ***sample***($\boldsymbol{B_{current}}$, $\mathbf{p_k}$)
15. $\boldsymbol{\Omega}$ = ***interpolate*** (*Depth*($\mathbf{p_k}$), $\boldsymbol{D_{i-1}}$, $\boldsymbol{D_i}$, $\boldsymbol{\Omega}_{prev}$, $\boldsymbol{\Omega}_{current}$)
16. $\boldsymbol{\tau}(\mathbf{p_k}) = e^{-\kappa\boldsymbol{\Omega}}$
17. $\boldsymbol{\Phi}(\mathbf{p_k}) = 1.0 - \boldsymbol{\tau}(\mathbf{p_k})$

In loop 1, the depth of each map is determined. Uniform slice spacing is reasonable for evenly distributed volumes (Figure 3a). When the structure of the volume is known, adaptive slicing (1D BSP) can be used such that tighter spacing is used in denser or more detailed regions (Figure 3b). Considering that regions farther from the light have ever-decreasing variations of shadows, a nonlinear partition conforms to perceptual sensitivity analogous to gamma correction (Figure 3c).

---

[3] The alpha value is a user-controllable parameter that depends on the size (thickness) and the optical property (albedo) of the primitive. It also depends on the resolution of the opacity maps.

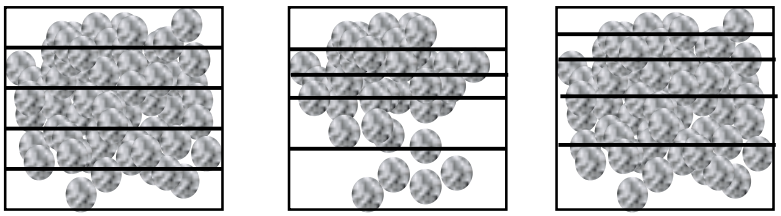


Fig. 3. Choice of slicing schemes. (a) Uniform slicing (b) Density based slicing (1D BSP) (c) Non-uniform slicing similar to Gamma Correction [4].

Prior to shadow rendering, shadow samples are produced for the primitives. In volumetric clusters, the primitives tend to be very small and thus the end points of lines and the vertices of polygons often suffice. More samples can be taken if needed. When many samples are required for each primitive, it may be useful to pre-compute the visibility and use only the visible points as shadow samples as in [15]. Loop 2 prepares a list of shadow samples that belong to each buffer. The procedure makes the shadow computation time linear in the number of samples.

Each pixel in the map stores the opacity value, which is a summation that produces the integral term Ω in equation (1). Thus each primitive can be rendered antialiased with hardware support in any order[4]. The alpha buffer is accumulated each time the volume is drawn with the OpenGL blend mode *glBlendFunc(GL_ONE,GL_ONE)*. The depth buffer is disabled. Clipping in line 10 ensures correct contribution of alpha values from the primitives and culls most primitives, speeding up the algorithm.

As loop 3 and 4 use only two opacity map buffers at a time, the memory requirement is independent of the total number of opacity maps computed. In loop 4, the shadow is computed only once for each sample. So, the amortized cost of the algorithm is linear in the number of samples. The overall complexity is *O(NM)* since the scene is rendered for each map, but the rendering cost is low with hardware acceleration.

The sample function in loop 4 can be any standard pixel sampling function such as a box filter, or higher-order filters such as a Bartlett filter and a Gaussian filter [4]. We use a 3x3 averaging kernel. Such filtering is possible because alpha values are stored instead of depths. The sampled opacity values $\Omega_{prev,}$ $\Omega_{current}$ are linearly interpolated for each point $\mathbf{p_k}$[5]. A higher order interpolation may be used. For example, four buffers will be needed for a cubic-spline interpolation.

A volume turns opaque as the opacity Ω reaches infinity. The quantization in the alpha channel limits the maximum amount of opacity that a pixel can represent. A constant κ in line 16 controls the scaling of opacity values such that $e^{-\kappa} = 2^{-d}$, where d is the number of bits per pixel (for example, κ is about 5.56 for 8 bit alpha buffer). Thus, an opacity value of 1.0 represents a complete opaqueness. The transmittance function for an opaque surface is a step function [10]. Although step functions are

[4] The order can be arbitrary due to the commutative nature of addition (or integral).

[5] $\Omega(\mathrm{p_k}) = t\Omega_{\mathrm{current}} + (1.0 - t)\Omega_{\mathrm{prev}}, t = (Depth(p_k) - D_{i-1})/(D_{\mathrm{i}} - D_{\mathrm{i\text{-}1}})$

approximated with a large number of maps, the integration of opaque objects is more efficiently achieved by adding a traditional depth buffer shadow map (Figure 5).

3. Results

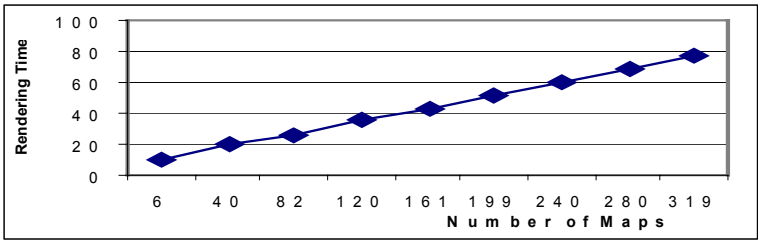


Fig. 4. Rendering time as a function of the number of maps

The performance of the algorithm is tested with a hair model of about 340,000 lines (Plate 1). Opacity maps are created at a resolution of 512 x 512 with uniform slicing scheme. The rendering time is linear in the number of the maps (Figure 4). Loop 3 and 4 account for most of the calculation time. Other steps such as the bounding box calculation (0.09 sec), assigning shadow samples to the maps (2.21 sec), and object shadow map generation (0.24 sec) use constant time. The slope of the graph in Figure 4 indicates the hardware's performance. About 4 opacity maps per second are computed using an SGI 540 NT Workstation with a 550Mhz Pentium CPU that can render about five million triangles per second. Figure 5 shows selected opacity maps. Plate 2 shows the images of a hair model of about 500,000 lines at various lighting conditions. Each hair strand is shaded using a lighting model in [7].

Fig. 5. Opacity shadow maps. Each *i*th map is shown from left to right, top to bottom (i = 1,6,14,26,40,52,60). The last figure shows a depth map for the opaque head and torso.

4. Discussion and Conclusion

Opacity shadow maps (OSM) provide a fast and memory efficient solution to rendering time-varying volumetric self-shadows without requiring any expensive preprocessing. Well suited for discontinuous volumes with explicit geometry (hair, fur, grass, and particle systems), it may be also used for continuous volumes (clouds and smoke) that are often sampled in voxel grids or implicitly defined. For example, clouds can be represented with independently moving point samples. Smoke can be

represented with a set of lines that vary their thickness and length in time.

A tradeoff between the speed and the quality is achieved by varying the number of maps. In complex geometry, the self-shadowing procedure can be viewed as a convolution of two high frequency signals, one from the geometry, and the other from the shadows. The study of the human visual system indicates that humans do not easily separate one signal from such mixed signals [2]. OSM mimics a low-pass filter for shadow signals. Due to the perceptual effects of visual masking, the degradation in perceived quality may be lower than quantitative measures might predict (Plate 3).

The opacity map farthest from the light can be used to compute shadows cast from the volume to other scene objects. A scene with many volumes can be spatially partitioned and the opacity maps can be composited. Instead of drawing every volume, the line 7 of the algorithm can be rewritten as 'Load the stored alpha buffer for the volumes closer to the light'. The partitioning scheme can improve the performance and reduce memory requirements for the scene graph. The simplicity of OSM may also allow further acceleration with multi-texture or 3D-texture hardware.

Acknowledgement

This work was funded by DARPA and the Annenberg Center at USC. Funding and research facilities were also provided from the NSF through its ERC funding of the Integrated Media Systems Center. The authors thank anonymous reviewers for many valuable comments.

References

[1] J. F. BLINN, Light reflection functions for simulation of clouds and dusty surfaces, *SIGGRAPH Proceedings*, Vol. 16, pp. 21-29, 1982.

[2] M. BOLIN AND G. W. MEYER, A frequency based ray tracer, *SIGGRAPH Proceedings*, Vol. 29, pp. 409-418, 1995.

[3] R. DREBIN, L. CARPENTER, AND P. HANRAHAN, Volume rendering, *SIGGRAPH Proceedings*, Vol. 22, pp. 65 – 74, 1988.

[4] J. FOLEY, A. VAN DAM, S. K. FEINER, AND J. F. HUGHES, Computer graphics, principles and practice, Second Edition, *Addison-Wesley*, July, 1995.

[5] A. S. GLASSNER, An introduction to ray tracing, *Academic Press,* 1993

[6] J. KAJIYA AND B. P. HERZEN, Ray tracing volume densities, *SIGGRAPH Proceedings*, Vol. 18, pp. 165-174, 1984.

[7] J. KAJIYA AND T. KAY, Rendering fur with three dimensional textures, *SIGGRAPH Proceedings*, Vol. 23, pp. 271-280, 1989.

[8] M. LEVOY, Display of surfaces from volume data, Ph.D. thesis, University of North Carolina at Chapel Hill, 1989.

[9] J. P. LEWIS, *Disney TSL, Personal communication.*

[10] T. LOKOVIC AND E. VEACH, Deep shadow maps, *SIGGRAPH Proceedings*, Vol. 34, pp. 385-392, 2000.

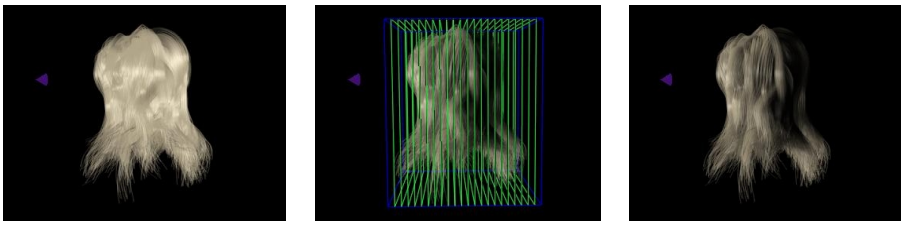
[11] S. N. PATTANAIK AND S. P. MUDUR, Computation of global illumination in a participating medium by Monte Carlo simulation, *The Journal of Visualization and Computer Animation*, Vol. 4(3), pp. 133-152, John Wiley & sons, 1993.

[12] W. T. REEVES, D. H. SALESIN, AND R. L. COOK, Rendering antialiased shadows with depth maps, *SIGGRAPH Proceedings*, Vol. 21, pp. 283-291, 1987.

[13] L. WILLIAMS, Casting curved shadows on curved surfaces, *SIGGRAPH Proceedings,* Vol. 12, pp. 270-274, August, 1978.

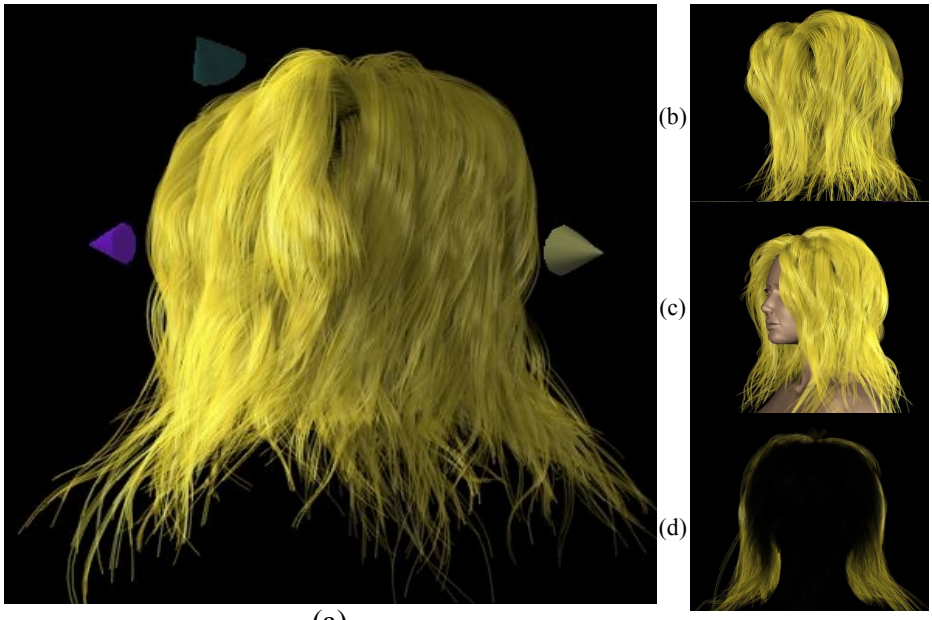
[14] A. WOO, P. POULIN, AND A. FOURNIER, A survey of shadow algorithms, *IEEE Computer Graphics and Applications, 10(6),* pp. 13-32, November, 1990.

[15] H. ZHANG, Forward shadow mapping, *Rendering Techniques '98*, Vol. 9, pp. 131-138, Springer-Verlag, 1998.



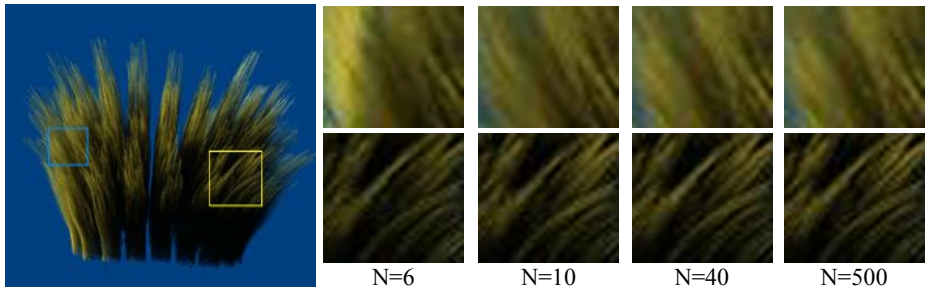
(a) (b) (c)

Plate 1: (a): Hair rendering without self-shadows. (b) Each opacity map is illustrated as a green rectangle. (c) Shadowed hair model (about 340,000 lines).



(a) (b) (c) (d)

Plate 2: (a) A hair model lit by three lights (N = 80). (b) Different view of the model of about 500,000 lines (c) The opaque head and torso is shown. (d) Backlighting effect.



N=6 N=10 N=40 N=500

Plate 3: A fur model rendered with 500 maps (left). One light is positioned at the left side of the image. Close-up views are shown on the right side, upper rows for a bright area (blue rectangle) and lower rows for a dark area (yellow rectangle). Note that artifacts are visible at brighter regions, but only with relatively small number of maps (N = 6).