Interactive Voxelized Epipolar Shadow Volumes

Chris Wyman*
University of Iowa

Abstract

Current algorithms for rendering shadows inside participating media hit a bottleneck when computing light visibility throughout the media. These algorithms rely either on sampling along viewing rays, often thrashing memory caches, or slower analytic solutions using object-space computations, such as shadow volumes.

We present a new cache-coherent sampling technique that computes volumetric light visibility that requires as little as one texture lookup per pixel. We efficiently voxelize shadow volumes in epipolar-space using standard parallel scan operations. The only step dependent on geometric complexity is an image-space voxelization [Eisemann and Décoret 2006] that often takes under a millisecond. This allows us to render shadows in participating media at up to 300 frames per second.

Keywords: shadows, participating media, interactive, voxelization, epipolar space

1 Introduction

While illuminating participating media usually presents a challenge, commonly-used simplifications reduce this complexity. With an aim to interactivity, many developers assume homogeneous single-scattering media, allowing inscattering to be computed as follows [Nishita et al. 1987]:

$$L_{scatter} = \int_0^{d_s} \beta\kappa(\alpha(x)) \frac{V(x) I_0 e^{-\beta d(x)}}{d(x)^2} e^{-\beta x} dx. \qquad (1)$$

See Figure 1 for definitions. Ignoring visibility (i.e., $V(x) = 1$) allows real-time solutions using a table lookup [Sun et al. 2005]. To add visibility researchers have proposed using a geometric representation of the shadow volume [Billeter et al. 2010] or sampling the visibility along each ray [Imagire et al. 2007]. As when shadowing surfaces, analytic shadow volume computations add significant fill-rate costs, so we do not consider them further in this sketch.

Sampling visibility along each ray transforms Equation 1 to:

$$L_{scatter} = \sum_{i=0}^{N} \beta\kappa(\alpha(x_i)) \frac{V(x_i) I_0 e^{-\beta d(x_i)}}{d(x_i)^2} e^{-\beta x_i}, \qquad (2)$$

where $x_i = d_s i/N$ (see Figure 1), though some work uses adaptive sampling for somewhat improved performance.

Sampling techniques treat each computation (e.g., at x_i and x_{i+1}) independently, often using shadow mapping to determine $V(x_i)$. Thus, for each pixel hundreds of shadow map queries occur, without any consideration for cache coherence or potential reuse between neighboring pixels.

2 Voxelized Shadow Volumes

Instead of independently determining visibility at every sample, we propose a screen space voxelization of the shadow volume that allows a *simultaneous* query of the light visibility at all samples along a view ray. We first describe a simple example in Cartesian space before moving to epipolar space.

Figure 2 shows the Buddha inside the Cornell Box. We create a binary screen space voxelization, giving a 3D volume with 1's representing the presence of geometry and 0's elsewhere. Screen space voxelization aligns the voxel grid to view rays, allowing us to simultaneously lookup the presence of geometry for all samples along a view ray with a single texture lookup.

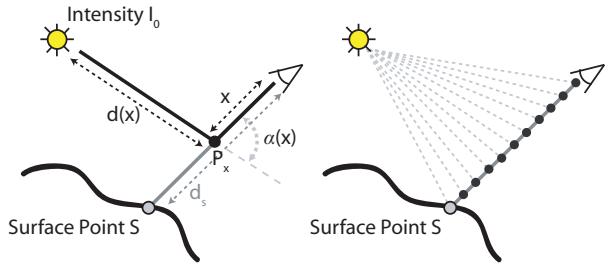

Figure 1: *(Left) Definitions of geometric quantities. $V(x)$ is the light visibility at P_x, β is the scattering coefficient, and $\kappa(\alpha(x))$ is the probability light scatters towards the viewer at P_x. (Right) Typically visibility is sampled independently at multiple points along each view ray.*

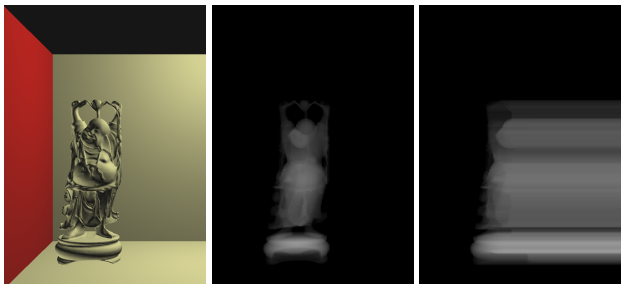

Figure 2: *(left) The Buddha inside the Cornell Box, (center) a voxelized representation, and (right) a voxelized representation of the shadow volume under a directional light (in the $+x$-axis). We visualize voxel grids as a count of filled voxels along each view ray.*

Imagine adding a directional light source pointing down the positive x-axis. We voxelize the Buddha's shadow volume. In this volume, 1's represent shadowed regions and 0's represent lit regions. This enables queries of light visibility simultaneously at all samples along a view ray. The key problem is efficient voxelization of the shadow volume. Clearly, we could voxelize using the shadow volume quads, but this reintroduces standard shadow volume fill-rate constraints.

We use a voxel representation of the original geometry as input to a parallel scan, with the scan counting the geometry-filled voxels between each sample and the directional light. Because we only need binary visibility, our parallel scan uses an *or* (instead of an *add*) operator. This is similar to creating a 1D summed area table [Hensley et al. 2005] for each row of voxels.

3 Epipolar Voxelization

The key problem is that scans occur along voxel grid axes, limiting applicability in Cartesian space to axis-aligned directional lights. However, if we move our voxel representation to epipolar space, then *by definition* one axis of the grid is aligned with the light direc-

*E-mail: cwyman@cs.uiowa.edu

SIGGRAPH Asia 2010, Seoul, South Korea, December 15 – 18, 2010.
ISBN 978-1-4503-0439-9/10/0012

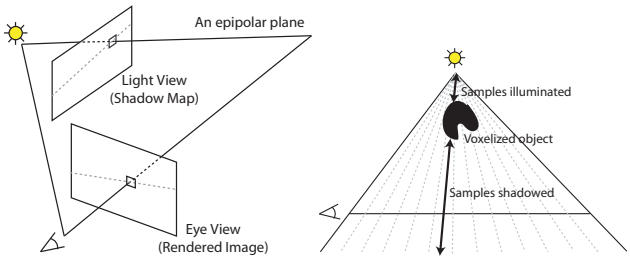

Figure 3: *(Left) Every ray through a pixel in image space corresponds to a line in light space (a dotted gray line), and every pixel in light space corresponds to a line in image space. View rays along these dotted lines fall on an epipolar plane, which contains the eye and the light. (Right) For each epipolar plane, we can voxelize into perspective grid where the axis in one direction is defined by rays emanating from the light. We then use the parallel scan from Section 2 to efficiently compute light visibility.*

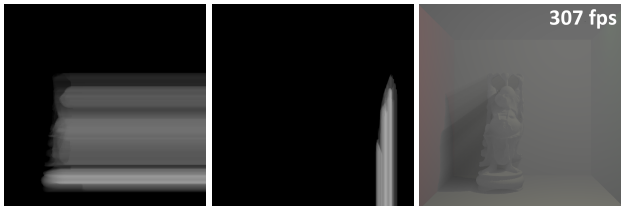

Figure 4: *(Left) Shadow volume voxelized in Cartesian space, (center) a similar shadow volume voxelized in epipolar space, (left) the volumetric shadows corresponding to the epipolar shadow volume.*

tion (see Figure 3), allowing a parallel scan to efficiently compute our voxelized shadow volume (see Figure 4).

Currently, we use a vertex shader to transform into epipolar space (as described below) and use the GPU's fixed function rasterizer to voxelize the geometry. Because this non-linear transform occurs per-vertex rather than per-fragment it can introduce distortion artifacts, though we did not encounter noticeable errors.

We transform each vertex from Cartesian coordinates to epipolar space, where the x-coordinate defines the epipolar plane and the y- and z-coordinates define the location on that plane. Epipolar planes correspond to image-space lines radiating from the light, so possible x values include $[0..360)$ depending on eye-space vertex location relative to the light.

Within each epipolar plane, our y-coordinate corresponds to distance from the light. Because distance to the light changes along a viewing ray, our vertex y-coordinate uses the minimal distance between the light and the ray passing through the eye and the vertex. Valid y-values include $[0..\infty]$, though the distance between the light and the view frustum boundaries provides our upper bound. An (x,y) pair defines a view ray in epipolar space.

Along each viewing ray, the z-coordinate specifies how far along the view ray a vertex is located. Looking at Figure 3, right, each triangular region between gray dotted lines corresponds to a different z value. In order to keep z-values from adjacent pixels consistent, this implies the z-value is actually an angle. We compute z as the dot product between $\hat{v}_{EL}$, the vector between the eye and light, and $\hat{v}_{VL}$, the vector between the current vertex and the light. This covers the range $[-1..1]$.

4 Results

We tested our prototype on an GeForce GTX 480 running at a resolution of 1024^2. The key hardware requirement is integer buffers, allowing bit manipulations for computing per-ray voxel bitstreams. One integer buffer gives a z-resolution of 128 samples, though our

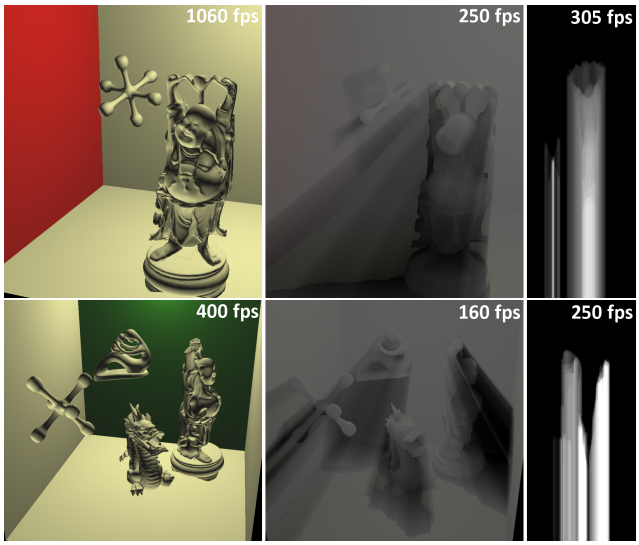

Figure 5: *Two different scenes rendered using voxelized epipolar shadow volumes. The top scene has 51,000 triangles whereas the bottom has 780,000. (Left) Simple Phong rendering, (center) voxelized shadow volumes, and (right) a subset of the corresponding epipolar voxel grid.*

results in Figure 5 used two buffers for a $1024^2 \times 256$ grid. Geometry with extra fine detail may require additional resolution.

In our tests on geometry of various complexity, voxelization takes 0.5-2.5 ms, the parallel scan to compute the voxelized shadow volume takes a constant 0.65 ms per layer (i.e., using two integer buffers for a z-depth of 256 requires 1.3 ms), and illumination computations for the participating media take another 1.0 ms.

Our results demonstrate performance up to 300 fps, and even complex scenes with over a million polygons still remain above 100 fps. We believe this demonstrates volumetric shadowing can be added to virtually all interactive applications, and with additional work adaptive computations could improve performance further.

References

BILLETER, M., SINTORN, E., AND ASSARSSON, U. 2010. Real time volumetric shadows using polygonal light volumes. In *Proc. High Performance Graphics*, 39–45.

EISEMANN, E., AND DÉCORET, X. 2006. Fast scene voxelization and applications. In *Proc. Symp. Interactive 3D Graphics and Games*, 71–78.

HENSLEY, J., SCHEUERMANN, T., COOMBE, G., SINGH, M., AND LASTRA, A. 2005. Fast summed area table generation and its applications. *Computer Graphics Forum 24*, 3, 547–555.

IMAGIRE, T., JOHAN, H., TAMURA, N., AND NISHITA, T. 2007. Anti-aliased and real-time rendering of scenes with light scattering effects. *Visual Computer 23*, 9, 935–944.

NISHITA, T., MIYAWAKI, Y., AND NAKAMAE, E. 1987. A shading model for atmospheric scattering considering luminous distribution of light sources. In *Proc. SIGGRAPH*, 303–310.

SUN, B., RAMAMOORTHI, R., NARASIMHAN, S., AND NAYAR, S. 2005. A practical analytic single scattering model for real time rendering. *ACM Trans. Graph. 24*, 3, 1040–1049.