Voxelized Shadow Volumes

Chris Wyman*
University of Iowa

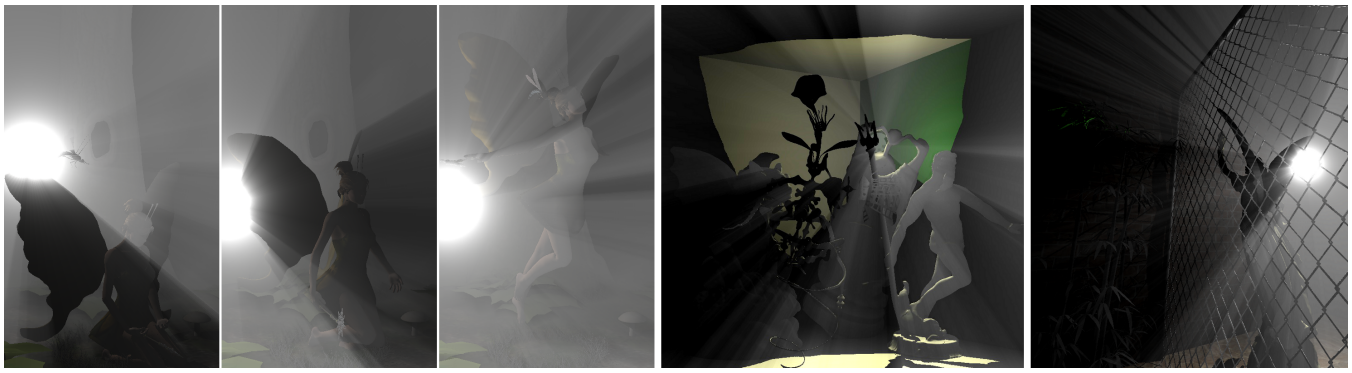

Figure 1: *Shadows in single-scattering homogeneous media, interactively rendered using voxelized shadow volumes. (Left) A sequence from the 175k triangle Fairy Forest animation, rendered at 118 fps. (Center) Numerous models totaling 1.7M triangles at 102 fps. (Left) The Lucy statue behind a chain-link fence at 119 fps.*

Abstract

Efficient shadowing algorithms have been sought for decades, but most shadow research focuses on quickly identifying shadows on surfaces. This paper introduces a novel algorithm to efficiently sample light visibility at points inside a volume. These *voxelized shadow volumes* (VSVs) extend shadow maps to allow efficient, simultaneous queries of visibility along view rays, or can alternately be seen as a discretized shadow volume. We voxelize the scene into a binary, epipolar-space grid where we apply a fast parallel scan to identify shadowed voxels. Using a view-dependent grid, our GPU implementation looks up 128 visibility samples along any eye ray with a single texture fetch. We demonstrate our algorithm in the context of interactive shadows in homogeneous, single-scattering participating media.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

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

1 Introduction

Computing visibility presents an enormous challenge for rendering. Despite decades of effort and orders of magnitude more compute power, many interactive applications still have problems rendering high quality shadowed surfaces. Only very recently have researchers introduced efficient algorithms that extend this visibility beyond surfaces and into volumetric media for dynamic scenes.

*e-mail: chris.wyman@acm.org

HPG 2011, Vancouver, British Columbia, Canada, August 5 – 7, 2011.
© 2011 ACM 978-1-4503-0896-0/11/0008 $10.00

Traditional surface shadowing algorithms can extend to volumes, e.g., repeatedly sampling shadow maps along view rays [Dobashi et al. 2002] or analytically computing visibility with shadow volume quads [Biri et al. 2006; Max 1986]. Unfortunately, repeatedly sampling shadow maps results in redundant, incoherent memory accesses and shadow volumes continue to consume excessive fill-rate. This means naive extensions of surface shadow techniques lead to insufficiently sampled visibility, poor scaling as geometric complexity grows, or unacceptable performance.

Recent research explores techniques to remove these limitations, including reducing shadow map sampling using simplified shadow volumes [Wyman and Ramsey 2008], extruding shadow maps [McCool 2000] to reduce shadow volume fill rate [Billeter et al. 2010], sampling at image-space discontinuities along epipolar rays [Engelhardt and Dachsbacher 2010], or using a min-max mipmap [Tevs et al. 2008] to accelerate ray tracing through a rectified shadow map [Chen et al. 2011]. Our work takes a similar approach to the last two techniques, sampling visibility in epipolar space.

This paper introduces a novel algorithm for sampling volumetric visibility that uses a binary voxel grid in epipolar space. We voxelize the scene into this grid, perform a parallel scan along the grid axis running along epipolar rays, then lookup visibility along the grid axis parallel to the view rays. Our performance stems from carefully designing the grid so computations access memory efficiently. This means visibility queries occur via a few **cache coherent** lookups rather than hundreds of cache oblivious shadow map queries. We also address numerous robustness issues and compare different techniques to efficiently sample scenes in epipolar space.

2 Voxelized Shadow Volumes

We propose creating a *voxelized shadow volume* (VSV) that discretizes space, storing binary values representing light visibility inside each voxel. Figure 2 shows a simplistic example of this process. Our **key contribution** lies in constructing a voxel grid enabling efficient creation and queries of these voxelized volumes.

Figure 2 uses standard Cartesian coordinates, giving an easily understandable visualization, but seriously limits applicability as effi-

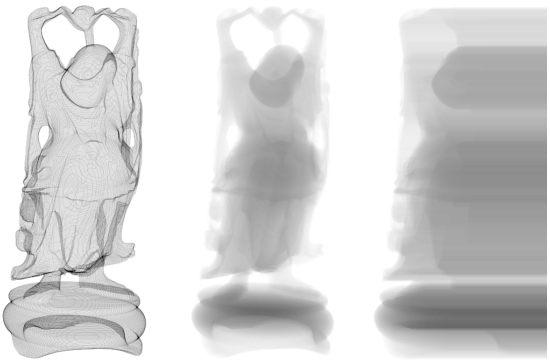

Figure 2: *A Buddha using (left) conservative surface voxelization [Schwarz and Seidel 2010] and (center) solid voxelization [Eisemann and Décoret 2006], displayed using a heatmap. In either case, (right) scanning from left to right gives a* voxelized shadow volume*, where a voxel stores the binary OR of all voxels to its left.*

cient scans only occur along grid axes. Rarely are lights perfectly oriented along an axis. Our key question is then: can we construct a grid where light always travels parallel to a grid axis?

Hunt and Marks [2008] showed such a grid, as an acceleration structure for tracing shadow rays. However, their grid only enables efficient visibility *computation*. We also want to *query* visibility efficiently. For general grids, identifying voxels intersecting a ray requires a ray-grid traversal. While impressive optimizations for traversals exist [Laine and Karras 2010], they still remain a rendering bottleneck. But if view rays are oriented along a grid axis, no traversal is required; we can simply retrieve a row of voxels. Eisemann and Decoret [2006] use a perspective grid that allows quick queries, but their grid fails to allow efficient scans along light rays.

We introduce an epipolar grid that combines these approaches, creating a "doubly" perspective grid where light rays travel parallel to one grid axis and view rays travel parallel to another axis.

2.1 Voxel Grid in Epipolar Space

As in Chen et al. [2011], Engelhardt and Dachsbacher [2010], and Max [1986], we compute shadow visibility in an epipolar coordinate space (see Figure 3). Unlike a Cartesian coordinate system, where coordinates are defined relative to an origin, our epipolar space defines coordinates relative to the *epipole*, the line connecting the eye and light points.

An infinite number of planes, called *epipolar planes*, contain both the eye, the light, and the epipole that connects them. Epipolar planes can be parameterized by angle θ around the epipole. We define θ relative to some up vector (e.g., the camera's up).

Within an epipolar plane, we need to ensure efficient implementation of a parallel scan to quickly create our voxelized shadow volumes. This requires grid axes to run parallel to light rays, implying a radial set of angular samples emanating from the light (see Figure 4). We define this angle, ϕ, relative to the epipole.

Similarly, to look up visibility along a view ray in a cache-friendly manner the eye rays should travel parallel with one set of axes. This implies a radial set of angular samples emanating from the eye. We also define this angle, α, relative to the epipole.

This gives us an epipolar space (α,θ,ϕ) that we can discretize, and allows efficient scanning along all light and eye rays. Initially, we choose (α,θ,ϕ) $\in [0..\pi] \times [0..2\pi) \times [0..\pi]$. In other words, we use

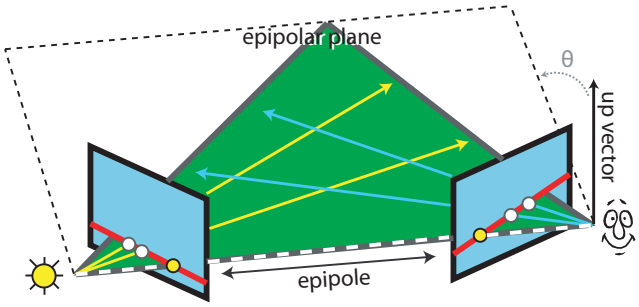

Figure 3: *The epipole connects the eye and light points. The epipole plus any ray from either the light or eye forms an* epipolar plane*. Epipolar planes intersect both the image and shadow map as a straight line (in red). Any pixel on these lines corresponds to an* epipolar line *that lies in that epipolar plane. We define epipolar planes by an angle θ relative to some global vector (e.g., the camera up vector).*

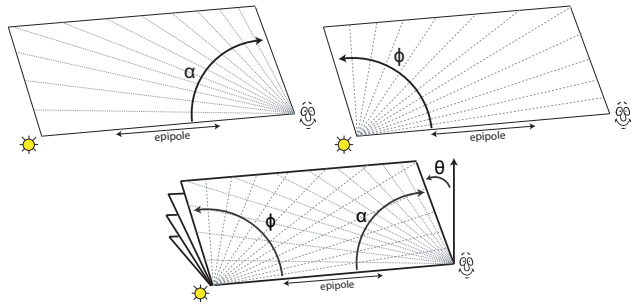

Figure 4: *We specify locations using three angles α, θ, and ϕ. α is the angle between the epipole and view ray, θ defines the epipolar plane, and ϕ is the angle between the epipole and light ray.*

epipolar half-planes, with α and ϕ in $[0..\pi]$ on each half plane. Our visibility technique below is independent of sampling rates along α,θ, and ϕ. Interestingly we found uniform sampling of all three angles worked well, though Section 4.4 discusses alternate sampling schemes.

Our code converts an eye-space vertex to epipolar space as follows:

Given eye-space light and vertex position: `esLPos`, `esVPos`

```
vec3 toLight = normalize( esLPos );
vec3 toVert = normalize( esVPos );
vec3 upVec = normalize( cross( toLight, vec3(0,0,-1) ) );
vec3 forwardVec = cross( upVec, toLight );

float α = acos( dot( toLight, toVert ) );
float θ = atan( dot( forwardVec, toVert ), dot( upVec, toVert ) );
float ϕ = acos( dot ( -toLight, normalize(esVPos-esLPos) ) );
```

2.2 Voxelized Shadow Volumes in Epipolar Space

Now that we have an epipolar space that meets our constraints for fast shadow evaluation and queries, we discuss creation of our voxelized shadow volumes. The approach is straightforward, using the same method outlined in Figure 2, only in epipolar space instead of Cartesian coordinates.

VSVs require three passes, visualized in Figure 5. The first two create the volume, and the third queries the volume during rendering:

1. Voxelize scene geometry into epipolar space.
2. Do a prefix scan (with a bitwise OR) along light rays.

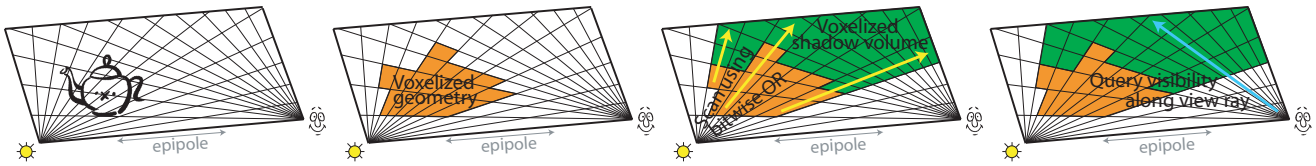

Figure 5: *Creating voxelized shadow volumes, visualized on a single epipolar plane. The original geometry (left) is voxelized into a binary representation (left center). Each radial row of voxels from the light is scanned using a bitwise* or *operator (right center) to create the voxelized shadow volume. This consists of green and orange voxels, which are regions shadowed by the original geometry. During rendering, voxels can be queried along the axis emanating from the eye (right). Voxels along these rays are packed into 128-bit unsigned RGBA texels to allow simultaneous lookups of 128 visibility samples.*

3. Query the volume by reading 128-bit RGBA texels, each corresponding to 128 visibility samples along the view ray.

2.2.1 Step 1: Voxelizing geometry

Researchers have proposed various voxelization techniques [Dong et al. 2004; Eisemann and Décoret 2006; Schwarz and Seidel 2010]. Other than voxelizing into epipolar space, we do nothing novel in this step. We use the code from Section 2.1 to transform geometry from eye-space into epipolar space and use existing voxelizers.

As in many non-Cartesian domains, the epipolar transform does not preserve straight lines. This means triangles have curved edges, and a rasterizer may incorrectly voxelize large triangles. For the relatively small triangles in most meshes, artifacts are imperceptible. We subdivide meshes with larger triangles, like big walls, as a preprocess. Tessellation stages on modern GPUs could instead do this dynamically and adaptively. We discuss further implementation issues arising from curved epipolar space in Section 4.1.

We tested both Eisemann and Decoret's [2006] screen-space voxelization and Schwarz and Seidel's [2010] conservative surface voxelization (see Figure 2). Screen-space voxelization requires watertight meshes, limiting use to well behaved models. Conservative voxelization works for all meshes but greatly degrades performance. In Section 4.1.3 we propose a shadow map resampling to replace voxelization for populating the epipolar grid.

2.2.2 Step 2: Scan along light rays

Figure 6 shows an example scene voxelized in epipolar space. As in Eisemann and Decoret [2006], we store voxels in a 2D texture, with each 128-bit RGBA texel containing 128 samples in the ϕ dimension. Each row in the texture corresponds to one epipolar plane θ. Along each row, α varies between 0 to π from left to right.

With this voxel representation, creating a VSV only requires scanning from left to right along each row. Each sample in a VSV stores a binary value identifying if that voxel is shadowed. Given a scene voxelization in epipolar space, a particular voxel (α_i, θ_j, ϕ_k) is shadowed when an occluder exists on the same epipolar plane θ_j along the same light ray ϕ_k. This can be written mathematically:

$$\exists s \in [0..i], \text{ such that } \mathbb{V}(\alpha_s, \theta_j, \phi_k) == 1, \tag{1}$$

where $\mathbb{V}(\alpha, \theta, \phi)$ is the voxel value at (α,θ,ϕ). Equivalently, this test can be written as a sum:

$$\sum_{s=0}^{i} \mathbb{V}(\alpha_s, \theta_j, \phi_k) > 0, \tag{2}$$

which suggests using a prefix sum. A sum requires more than one bit per voxel, so we instead reformulate this scan using an *or* (instead of addition) operator. Our test for occlusion becomes:

$$\mathbb{V}(\alpha_0, \theta_j, \phi_k) \mid \mathbb{V}(\alpha_1, \theta_j, \phi_k) \mid ... \mid \mathbb{V}(\alpha_i, \theta_j, \phi_k) == 1. \tag{3}$$

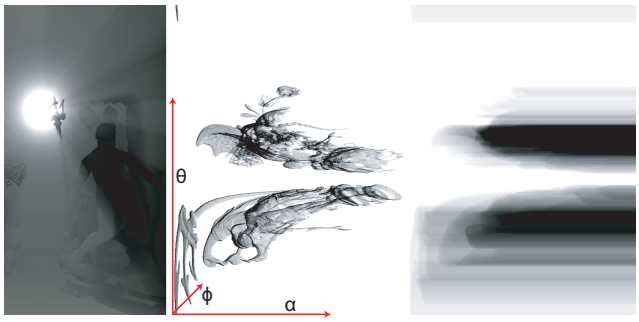

Figure 6: *(Left) A scene with four complex models. (Center) The models voxelized in epipolar space. Each row represents a single epipolar plane θ_i, with α running horizontally with the light (α=0) at left. Each texel stores multiple bits, representing samples in ϕ. Visualized pixel intensity corresponds to the number of ϕ bits set for each (α, θ) pair. (Right) We scan along rows to create a VSV, storing the bitwise* or *of all pixels to the left.*

This can also be understood visually, in Figure 5, asking for each voxel, "is there an orange voxel between myself and the light?"

We perform our scan via a multipass OpenGL-based variant of Harris et al. [2008]. Our variant creates a mip-chain of VSVs as a side effect; we do not currently use these, though we expect future work on VSVs may utilize such mip-chains.

2.2.3 Step 3: Query visibility

Given a voxelized shadow volume, as in Figure 6, each uvec4 texel stores 128 binary visibility samples along the corresponding view ray (i.e., a row of voxels with only varying ϕ). Thus a single texture fetch reads 128 visibility samples. Using bit twiddling we can determine light visibility at various points along the ray. For instance, Figure 7 counts shadowed bits to visualize the shadow volume and uses a single-bit lookup to render surface shadows.

We described VSV construction using 128 ϕ-samples, using a single 128-bit RGBA texture. With uniform sampling, 128 samples rarely suffice to avoid aliasing. All our results use 512 ϕ-samples. Larger volumes are easily created using multiple render targets to store additional bits per texel (e.g., as in Eisemann [2006]).

3 Shadows in Participating Media with VSVs

For inhomogeneous single-scattering media, the airlight equation describes the light scattered towards the eye [Nishita et al. 1987]:

$$L_s = \int_0^d \sigma_s e^{-t\sigma} V(t) \rho(t) L_{in}(t) dt, \tag{4}$$

where $\sigma = \sigma_s + \sigma_a$ is the extinction coefficient, σ_s and σ_a are the media's scattering and absorption coefficients, $\rho(t)$ is the phase

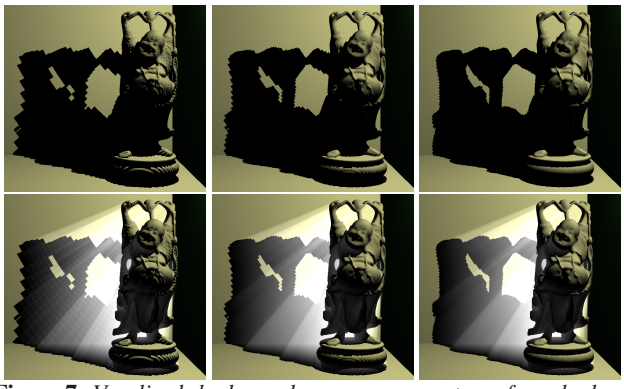

Figure 7: *Voxelized shadow volumes can compute surface shadows (top) by querying the voxel containing each fragment. (Bottom) The VSV is visualized by counting shadowed voxels along each view ray. From left to right, we use 128, 256, or 512 angular samples in ϕ.*

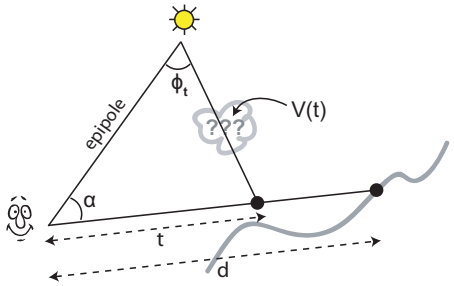

Figure 8: *The inscattering L_s depends on distance d to the surface and the angle α between the epipole and view ray. Along this ray, we integrate over t values between 0 and d. Each t had a corresponding angle from the light ϕ_t, and possible occlusion events are represented by V(t). We compute V(t) by querying our VSV, $\mathbb{V}(\alpha, \theta, \phi_t)$, where θ is the epipolar plane containing this ray.*

function, $L_{in}(t)$ is the light intensity, and $V(t)$ is a binary visibility accounting for shadows (see Figure 8).

Ignoring visibility, Equation 4 can be solved analytically [Pegoraro et al. 2009; Sun et al. 2005]. Including visibility, the integral can be split into separate pieces [Biri et al. 2006], one per lit segment along a view ray, or sampled via ray marching:

$$L_s \approx \sum_{i=0}^{N} \sigma_s e^{-t_i \sigma} V(t_i) \rho(t_i) L_{in}(t_i) \Delta t_i. \quad (5)$$

Dobashi et al. [2002] suggested an approximation that pulls visibility out of the summation. Baran [2010] and Chen [2011] improved the decoupling between visibility and scattering with a low rank SVD factorization. Since the sum varies smoothly along epipolar rays, Engelhardt and Dachsbacher [2010] sample L_s sparsely along epipolar rays and interpolate. VSVs can provide the visibility for any of these techniques, including the brute force Riemann sum in Equation 5.

In our examples, we compute scattering using either the Hoffman and Preetham [2003] or uniform ($\rho = \frac{1}{4\pi}$) phase functions.

4 Implementation Details

VSV implementation requires some care, due to various robustness issues. Some insignificant numerical precision issues in perspective space become showstoppers in epipolar space. Other precision issues cause aliasing that is fixable with careful parameter choices.

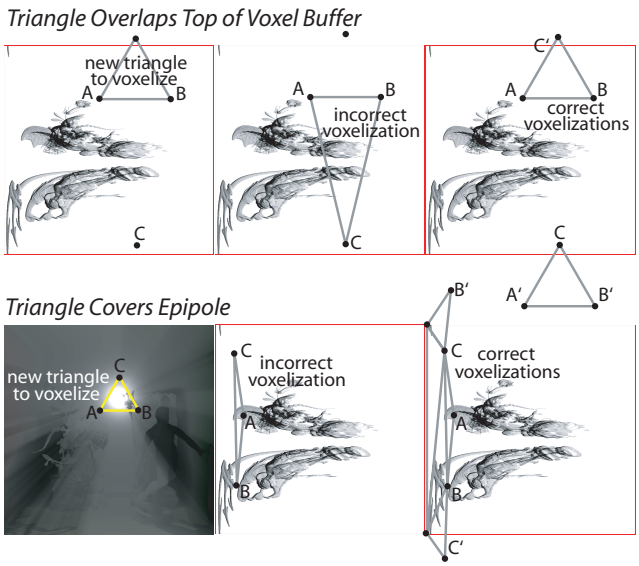

Figure 9: *(Top) A triangle overlapping the top or bottom of the voxel buffer requires splitting for correct voxelization. (Bottom) A triangle occluding the light must be split to include the entire epipolar singularity. (Left) A problem triangle, (center) the naive, incorrect voxelization, and (right) the correctly subdivided triangle.*

4.1 Rendering to a Cylindrical Texture

As we store our voxel grid in a 2D texture, the left edge represents $\alpha = 0$, the singularity at the epipole (see Figure 6). As geometry approaches the light, it spreads to cover the entire left column. Additionally, the top and bottom edges both represent $\theta = 0$. As geometry disappears off the top texture edge, it reappears at the bottom.

This section describes two key details needed to ensure correct rendering under these distortions: triangle splitting and conservative voxelization. We also describe an alternative to voxelization that avoids artifacts by resampling the shadow map.

4.1.1 Triangle Splitting

While tessellating triangles to avoid distortion from curved edges in epipolar space is not strictly necessary, two other types of triangle splits must occur to ensure correctness (see Figure 9).

When triangles overlap the top or bottom of the voxel texture they wrap around, occupying voxels near both buffer edges. Voxelizing triangle ΔABC fails when individually converting A, B, and C into epipolar space and connecting them. Instead, we voxelize triangles $\Delta ABC'$ and $\Delta A'B'C$, where A', B', and C' are virtual vertices outside the texture. These virtual vertices have identical α and ϕ coordinates, but the θ coordinates vary: $\theta' = \theta \pm 2\pi$.

A triangle shadowing the eye includes the epipole. Ideally, our epipolar singularity spreads such triangles over the left side of the voxel buffer, but naive voxelization fails to include any of the singularity. Instead, we subdivide such triangles to explicitly include the entire epipolar singularity. Since, by definition, any triangle occluding the epipole intersects $\theta = 0$, we also need to introduce two virtual vertices.

Failing to subdivide problem triangles in both cases introduces light leakage (see Figure 10). Failure to subdivide triangles intersecting $\theta = 0$ also introduces spurious shadows, as the occluder triangle flips to cover wrong epipolar planes.

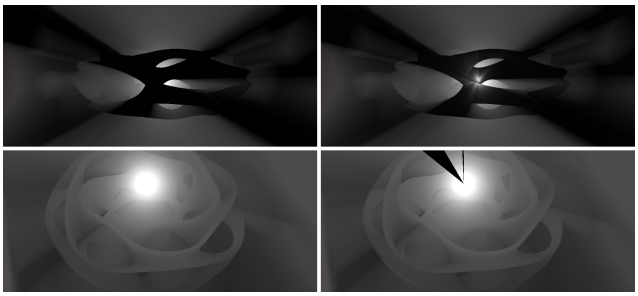

Figure 10: *The heptagonal toroid with (left) correct VSVs and (right) artifacts from failing to subdivide problem triangles. (Top) Light leakage through the surface. (Bottom) Triangles overlap the epipole, behind the light, introducing spurious shadows.*

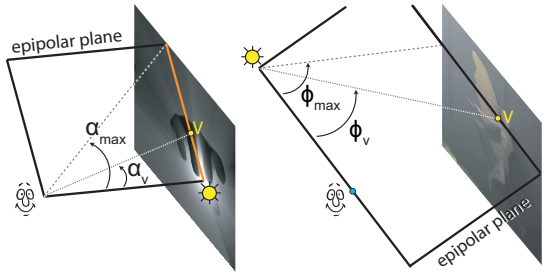

Figure 11: *By limiting the ranges of α and ϕ, per frame, based on light position and viewing parameters, we can focus samples on occluders affecting the image.*

4.1.2 Conservative Voxelization

As in any sampling, geometry may fall between voxels in the epipolar grid. Yet this geometry still casts shadows, and from frame to frame shadows from poorly sampled triangles flicker in and out, as movement causes occasional voxel coverage. Additionally, screen-space voxelization via a hardware rasterizer may cull silhouette triangles (i.e., those with constant α), despite potentially large shadows. Using conservative voxelization [Schwarz and Seidel 2010] addresses these issues, reducing artifacts.

More importantly, as geometry approaches the epipole singularity, it gets stretched quite thin in our epipolar voxel buffer. As with naturally tiny geometry, these stretched triangles are poorly sampled. This contributes to the light leakage seen in Figure 10. Conservative voxelization guarantees coverage of appropriate nearby voxels, and when combined with the triangle subdivisions from Section 4.1.1 eliminates all light leakage.

4.1.3 Voxelization Via Shadow Map Resampling

We see voxelization into the epipolar domain as the most flexible approach for VSV creation, e.g., allowing natural extensions to handle deep shadow maps [Lokovic and Veach 2000]. However, the robustness issues from Sections 4.1.1 and 4.1.2 plus the performance demonstrated below suggest using a more specialized approach when only opaque occluders are needed.

Consider the requirements for valid VSVs. For the scan along a light ray to produce correct results, voxels containing geometry closest to the light must be marked occupied. Shadowed surfaces need not be voxelized, as the scan sets those voxels.

By definition, shadow maps store the geometry closest to the light along any ray. Thus, we can voxelize these surfaces simply by resampling the shadow map, looking up the closest occluder along each light ray (θ_j, ϕ_k) and setting the voxel bit at $\mathbb{V}(\alpha_d, \theta_j, \phi_k)$. Here α_d is the α sample corresponding to the shadow map depth in direction (θ_j, ϕ_k).

To do this we render an array of $N_\theta \times N_\phi$ points, where N_θ and N_ϕ are the voxel buffer resolutions in the θ and ϕ dimensions. For each $j \in [0..N_\theta]$ and $k \in [0..N_\phi]$, we compute the light ray in direction (θ_j, ϕ_k) and lookup the corresponding depth d_{jk} in the shadow map. We unproject this shadow map sample back into eye space, and use the transform from Section 2.1 to convert this back to epipolar space to find the α_d corresponding to d_{jk}. We then set the bit at $\mathbb{V}(\alpha_d, \theta_j, \phi_k)$.

This approach robustly handles any rasterizable geometry and naturally handles the epipolar singularity. This approach is similar to Billeter et al. [2010]; instead of generating an analytical light volume, we generate a discrete shadow volume.

4.2 Precisions Issues Converting to Epipolar Space

Our code to convert eye-space to epipolar coordinates in Section 2.1 must be carefully implemented to avoid precision errors. In particular, despite normalized vectors, the parameters to the `acos` functions may lie outside the valid $[-1..1]$ range, and must be clamped. Depending on voxelization engine implementation, NaNs from an out-of-range `acos` either lead to ignored triangles, increasing light leakage, or introduce large triangles covering all epipolar space, incorrectly shadowing the scene and consuming significant fill rate. When generating input vertices for our voxelizers, we carefully clamped inputs to all trigonometry functions to avoid such precision issues.

4.3 Dynamic, Angular Sampling Ranges

Many algorithms relying on screen-space sampling limit themselves to geometry visible on screen. One advantage to sampling $\alpha, \phi \in [0..\pi]$ and $\theta \in [0..2\pi)$ is that our epipolar voxelization captures all scene geometry. Unfortunately, this wastes voxel samples on geometry with no contribution to visible pixels, reducing sampling quality in more important regions. Dynamic, per-frame range selection along all three angles allows more focused samples near occluders that shadow visible geometry (see Figure 11).

First, the angle between light and view rays ranges $\alpha \in [0..\pi]$. However, a limited subset of this range is visible on screen. Scenes with visible lights only use samples with α less than the field of view. Because we scan left-to-right along epipolar planes, only samples α_s for $0 \leq s \leq i$ affect current ray α_i. We sample $\alpha \in [0..\alpha_{max}]$, for $\alpha_{max} = max(\alpha_{x,y})$ the maximum angle over all image pixels (x, y). This allows more dense sampling along the α range, greatly reducing angular aliasing (see Figure 12).

Second, sampling epipolar planes $\theta \in [0..2\pi)$ is vital only when the epipole lies inside the view frustum; we only need compute the VSV for visible epipolar planes. Given our parallel scan consumes a significant percentage of frame time, limiting scans to important visible epipolar planes significantly improves performance. We did not implement a generally applicable dynamic sampling in θ, partly because more adaptive sampling schemes seem promising. Our results all use $\theta \in [0..2\pi)$.

Finally, ϕ samples lie in $[0..\pi]$ whenever the view frustum contains the light, as in most of our scenes. However, when the light lies behind the camera, visible ϕ samples may be limited to a tiny range. As the ϕ sampling rate strongly affects VSV quality, a fixed ϕ sampling severely undersamples shadows in such scenarios. Instead,

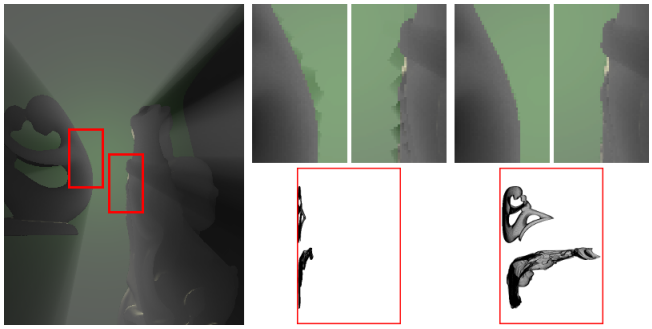

Figure 12: *Insets show (center) sampling $\alpha \in [0..\pi]$, resulting in angular jaggies along silhouettes and (right) sampling α dynamically, reducing artifacts by focusing samples in the view frustum.*

we sample $\phi \in [0..\phi_{max}]$, for $\phi_{max} = max(\phi_{x,y})$ the maximum angle for geometry visible at pixels (x, y).

4.4 Alternative, Non-Uniform Angular Sampling

Our implementation, even using the per-frame dynamic range sampling from Section 4.3, relies on uniformly-spaced angular sampling in all three dimensions. In other words, if we use N samples for $\alpha \in [0..\alpha_{max}]$, then $\alpha_i = \alpha_{i-1} + \frac{1}{N}\alpha_{max}$.

Engelhardt [2010] and Baran [2010] non-uniformly distribute θ samples to ensure epipolar planes always lie less than one pixel apart. This is orthogonal to our work; we did not implement it, though it could easily be incorporated and would improve quality.

Our initial work sampled ϕ using various non-uniform schemes. Most gave significantly worse quality. One obvious approach might sample ϕ more densely close to 0, perhaps based on $\cos\phi$. We expected this to reduce angular aliasing, but our experiments failed to show a noticeable improvement. We also searched for a ϕ parameterization allowing cancellation of additional terms from Equation 5, though due to close coupling of α and ϕ in the scattering equation, we failed.

Non-uniform sampling in α dramatically impacts image quality, as samples are directly visible in the rendering. Engelhardt and Dachsbacher [2010] found samples at depth discontinuities most important, as shadows change there. They sample at these discontinuities and interpolate between samples, essentially using a scan to find nearby samples. Since voxelization finds geometry discontinuities and VSV construction performs a scan, we essentially generalize their sampling scheme to accurately compute shadow bounds. We found other non-uniform α samplings increased aliasing at object boundaries.

5 Results and Discussion

We implemented voxelized shadow volumes using OpenGL and CUDA. All timings (see Table 1) use a 2.6 GHz quad-core Xeon X5335 and a GeForce 580 GTX. Unless otherwise specified, timings use a 1024^2 resolution display, a 2048^2 dual paraboloid shadow map [Brabec et al. 2002], and epipolar voxel buffers with 512 samples in α and ϕ and 2048 samples in θ. Figure 13 shows VSVs applied to various scenes.

Table 1 compares VSV creation cost using all three methods: resampling the shadow map, a CUDA-based conservative voxelization, and screen-space voxelization. Resampling the shadow map consistently performs best. Screen-space voxelization has similar performance characteristics but is mostly unusable, as in practice many models are not watertight. Schwarz and Seidel [2010] kindly provided conservative voxelization code, but it performs poorly on scenes with high depth complexity or many small triangles in a concentrated region.

Rendering Step	**Scene: Complex Models in Box** Using varying shadow map resolution				
	512^2	1024^2	2048^2	4096^2	8192^2
Shadow Map	2.5	2.5	2.6	2.7	3.6
Voxelize: From SM	1.8	1.9	1.9	2.3	2.3

Table 2: *Varying shadow map size barely affects performance. Timings are in milliseconds.*

Rendering Step	**Scene: Heptagonal Toroid** Varying number of θ samples				
	128^2	256^2	512^2	1024^2	2048^2
Voxelize: From SM	0.2	0.4	0.8	1.5	2.9
Voxelize: CUDA	2.5	2.9	4.4	7.8	15
Voxelize: SSVox	1.0	1.0	1.0	1.2	2.2
Scan VSV	1.0	1.1	1.3	1.7	2.7

Table 3: *Varying the number of epipolar planes greatly affects performance for all of types of voxelization. As we scan along each plane, θ sampling also affects the parallel scan. Timings are in milliseconds.*

Resampling the shadow map does not slow significantly when varying shadow map resolution (see Table 2). The main cost is additional fill rate to render the map, as our resampling technique uses a fixed number of shadow map samples for any voxel buffer resolution, one sample per (α,θ) pair. The slight cost increase in Table 2 likely stems from poorer cache coherence for big shadow maps.

Increases to voxel buffer resolution significantly impact performance (see Table 3). Fill rate costs generally increase linearly for larger voxel buffers, though the screen-space and conservative voxelizers have fixed costs that mask this relationship (e.g., CUDA interop overhead and per-vertex transforms to epipolar space). Changes in other buffer dimensions affect performance in slightly different ways, outlined in Table 4.

Figure 7 shows we can render surface shadows with VSVs. Generally this does not make sense, especially when generating the VSV by resampling a shadow map. Even using the other creation methods, the voxel grid view-dependence means surface shadow aliasing varies with viewpoint.

Compared to other interactive shadowing techniques, voxelized shadow volumes use a discrete representation. This introduces aliasing not apparent in analytical shadow volumes techniques, such as Billeter et al. [2010]. Since we sample in similar epipolar spaces, our discretization artifacts resemble those of Chen et al. [2011] and Engelhardt and Dachsbacher [2010]. Techniques sampling coarsely in image space (e.g., Wyman and Ramsey [2008]) tend to avoid aliasing by blurring shadow details (see Figure 14), which we need not do. Our remaining aliasing mainly stems from poor sampling from elongated voxels near the epipole. This manifests as poor ϕ sampling for objects near the light, as in Figure 15. Unlike Chen [2011], we need not fall back to brute force ray marching in this region. Only for dense media (as in our images) will some discretization becomes visible, though in animations it is usually transient. We suspect bilateral filtering in either epipolar or screen space could address this.

Given sufficient sampling density in the epipolar domain, our voxelized shadow volume gives results equivalent to ray tracing (see Figure 16), but with significantly improved performance. As with

Scene	Poly Count	Individual Step Times (in milliseconds)							Total Frame (Voxelizing from shadow map)
		Shadow Map	G Buffer	Voxelize to Epipolar Space			Scan VSV	Compute L_s	
				From SM	CUDA	SSVox			
24-Cell Tesseract Animation	123 k	0.4	0.7	1.3	120	1.0	2.7	0.7	5.8 ms
Chain Link Fence + Lucy	682 k	1.1	1.4	2.0	31	1.9*	2.7	0.7	7.9 ms
Complex Models in Cornell Box	1700 k	2.6	2.1	1.9	10	3.3	2.7	0.7	9.8 ms
Fairy Forest Animation	175 k	0.8	3.0	3.1	150	3.9*	2.7	0.7	10.1 ms
Hand and Figure Animation	120 k	0.3	1.3	1.4	1.8	0.5*	2.7	0.7	6.3 ms
Heptagonal Toroid	573 k	0.9	0.9	2.9	15	2.2	2.7	0.7	7.9 ms
Indoor Garden	51 k	0.4	1.6	3.4	580	*n/a**	2.7	0.7	8.8 ms
Stanford Dragons	575 k	1.2	1.9	1.4	3.2	1.5	2.7	0.7	7.7 ms

Table 1: *Performance breakdown of individual algorithm steps for our scenes. All timings use an output resolution of* 1024^2*,* 2048^2 *dual paraboloid shadow maps, and a* $512 \times 2048 \times 512$ *epipolar voxel buffer (i.e.,* θ_i *is sampled for* $i \in [0..2048]$*). Steps include shadow map creation, G-buffer creation, voxelization into epipolar space, performing a parallel scan to compute the VSV, and a final step that computes scattering and outputs a final color. Timings are given for the three different voxelization techniques: shadow map resampling (*From SM*), a conservative voxelizer (*CUDA*), and screen-space voxelization (*SSVox*). Screen-space voxelization fails for non-watertight meshes; starred timings represent scenes with significant artifacts that may not represent true voxelization cost (the indoor garden scene fails catastrophically). The total frame time sums all steps, voxelizing via shadow map resampling, which performed consistently well in all scenes. Due to rounding, the total cost does not equal the sum of all steps.*

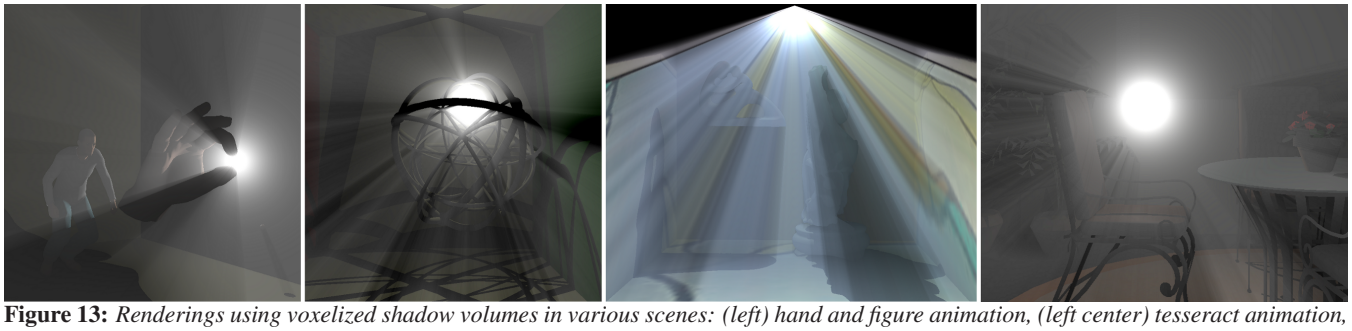

Figure 13: *Renderings using voxelized shadow volumes in various scenes: (left) hand and figure animation, (left center) tesseract animation, (right center) simple Buddha scene with anisotropic (i.e., textured) light, and (right) the indoor garden.*

Rendering Step	Varying Sampling Rate In		
	α	θ	ϕ
Voxelize: From SM	$O(1)^1$	$O(\theta)^2$	$O(\phi)^2$
Voxelize: CUDA	$O(\alpha)^2$	$O(\theta)^2$	$O(1)^3$
Voxelize: SSVox	$O(\alpha)^2$	$O(\theta)^2$	$O(\phi)^4$
Scan VSV	$O(\log \alpha)^5$	$O(\theta)^2$	$O(\phi)^4$

[1] No increase in shadow map samples. [2] From increased fill rate.
[3] Varies with cache coherence. [4] Extra memory traffic to framebuffer.
[5] Scan steps increase logarithmically along scan direction.

Table 4: *How performance varies with voxel buffer resolution.*

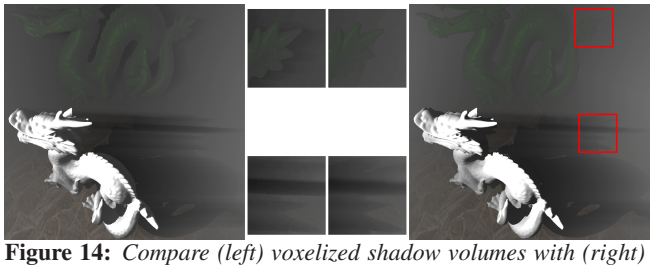

Figure 14: *Compare (left) voxelized shadow volumes with (right) Wyman and Ramsey [2008]. (Top) We handle all geometry for equivalent cost, not focusing computation on designated shadow casters. (Bottom) We do not blur in image space, allowing sharper crepuscular rays.*

other visibility techniques, we need not limit VSVs to computing visibility in participating media from isotropic lights (see Figure 13), as visibility computation is usually orthogonal to the L_{in} term in Equation 5. Times for the final render step in Table 1 use Dobashi's [2002] factored visibility. This works poorly for textured lights, so in those cases we coarsely step along each view ray accumulating light color, increasing L_s computation to 5.3 ms. An incremental computation along epipolar planes [Baran et al. 2010] would speed this step.

6 Conclusions and Future Work

We introduced *voxelized shadow volumes*, an algorithm based on a new discretization of epipolar space. We voxelize the scene into this space, and scan along light rays to efficiently compute a voxelized representation of a shadow volume. Querying this volume provides fast, cache coherent shadow samples throughout the scene. We demonstrate an application of VSVs to compute visibility for interactive shadows in single-scattering homogeneous participating media. While we see VSVs as more general, one could view them as a drop-in replacement for prior shadow query techniques (e.g., the 1D min-max mipmaps from Chen et al. [2011]).

We see many future extensions for VSVs, including filtering techniques in the epipolar voxel buffer, applications to other domains such as soft shadowing, adaptive sampling to improve performance and quality using fewer samples, and exploring efficient implementations with non-binary voxels (e.g., for deep shadow maps).

Acknowledgments

The author was supported by grants from DARPA (#HR0011-09-1-0027) and ARO (#W911NF-10-1-0338), with generous hard-

Figure 15: *Epipolar regions may have poor ϕ sampling, due to elongated voxels in this region. Angular aliasing occurs in regions with few occluded ϕ samples. Noticeable artifacts above occur going from $0 \rightarrow 1$ or $1 \rightarrow 2$ ϕ occlusions between adjacent epipolar planes.*

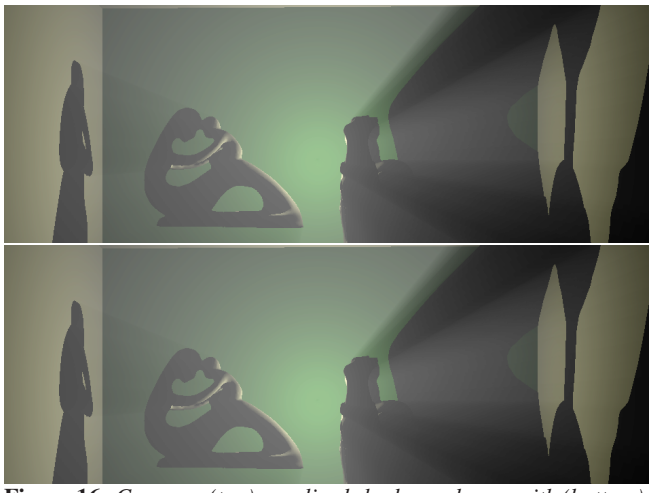

Figure 16: *Compare (top) voxelized shadow volumes with (bottom) an OptiX ray traced ground truth. VSVs run at 127 fps versus 0.3 fps for OptiX. The OptiX implementation uses ray marching, firing a shadow ray for the same 512 ϕ samples we use on each view ray.*

ware contributions from NVIDIA. Models come from Google's 3D Warehouse, the Stanford repository, Keenan Crane, AIM@Shape, and the Utah 3D Animation Repository. The author also thanks the anonymous reviewers and other people, too numerous to list, who listened to and gave feedback on these ideas.

References

BARAN, I., CHEN, J., RAGAN-KELLEY, J., DURAND, F., AND LEHTINEN, J. 2010. A hierarchical volumetric shadow algorithm for single scattering. *ACM Transactions on Graphics 29*, 178:1–178:10.

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

BIRI, V., ARQUES, D., AND MICHELIN, S. 2006. Real time rendering of atmospheric scattering and volumetric shadows. *Journal of WSCG 14*, 65–72.

BRABEC, S., ANNEN, T., AND SEIDEL, H.-P. 2002. Shadow mapping for hemispherical and omnidirectional light sources. In *Proceedings of Computer Graphics International*, 397–408.

CHEN, J., BARAN, I., DURAND, F., AND JAROSZ, W. 2011. Real-time volumetric shadows using 1d min-max mipmaps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 39–46.

DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2002. Interactive rendering of atmospheric scattering effects using graphics hardware. In *Graphics Hardware*, 99–107.

DONG, Z., CHEN, W., BAO, H., ZHANG, H., AND PENG, Q. 2004. Real-time voxelization for complex polygonal models. In *Proceedings of Pacific Graphics*, 43–50.

EISEMANN, E., AND DÉCORET, X. 2006. Fast scene voxelization and applications. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 71–78.

ENGELHARDT, T., AND DACHSBACHER, C. 2010. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 119–125.

HARRIS, M., SENGUPTA, S., AND OWENS, J. 2008. *GPU Gems 3*. Addison-Wesley, ch. Parallel Prefix Sum (Scan) with CUDA, 851–876.

HOFFMAN, N., AND PREETHAM, A. 2003. *Game Programming Methods*. Charles River Media, ch. Real-time light-atmosphere interactions for outdoor scenes, 337–352.

HUNT, W., AND MARK, W. 2008. Adaptive acceleration structures in perspective space. In *Proceeedings of the Symposium on Interactive Ray Tracing*, 11 –17.

LAINE, S., AND KARRAS, T. 2010. Efficient sparse voxel octrees. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 55–63.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *Proceedings of SIGGRAPH*, 385–392.

MAX, N. 1986. Atmospheric illumination and shadows. In *Proceedings of SIGGRAPH*, 117–124.

MCCOOL, M. 2000. Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics 19*, 1, 1–26.

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

PEGORARO, V., SCHOTT, M., AND PARKER, S. 2009. An analytical approach to single scattering for anisotropic media and light distributions. In *Proceedings of Graphics Interface*, 71–77.

SCHWARZ, M., AND SEIDEL, H.-P. 2010. Fast parallel surface and solid voxelization on gpus. *ACM Transactions on Graphics 29*, 179:1–179:10.

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

TEVS, A., IHRKE, I., AND SEIDEL, H.-P. 2008. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *Proceedings of the Symposium on Interactive 3D graphics and games*, 183–190.

WYMAN, C., AND RAMSEY, S. 2008. Interactive volumetric shadows in participating media with single-scattering. In *Proceedings of IEEE Symposium on Interactive Ray Tracing*, 87–92.