Imperfect Voxelized Shadow Volumes

Chris Wyman[*]
NVIDIA

Zeng Dai[†]
University of Iowa

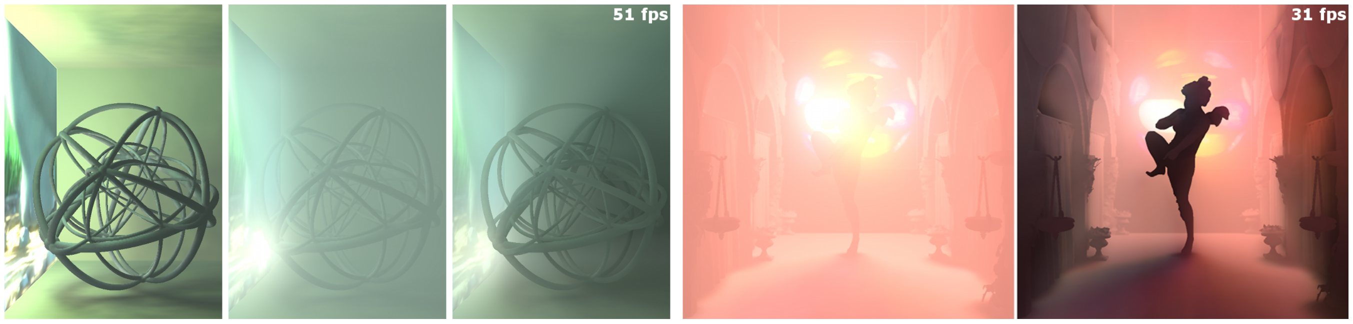

Figure 1: *(Left) A scene with no participating media, media with no shadows, and with imperfect voxelized shadow volumes. (Right) A more complex scene without and with volumetric shadows. All images also use imperfect voxelized shadow volumes for surface shadows.*

Abstract

Voxelized shadow volumes [Wyman 2011] provide a discretized view-dependent representation of shadow volumes, but are limited to point or directional lights. We extend them to allow dynamic volumetric visibility from area light sources using *imperfect shadow volumes*. We show a coarser visibility sampling suffices for area lights. Combining this coarser resolution with a parallel shadow volume construction enables interactive rendering of dynamic volumetric shadows from area lights in homogeneous single-scattering media, at under 4x the cost of hard volumetric shadows.

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

Keywords: shadows, area lights, participating media, voxelization

1 Introduction

Interactive shadow algorithms typically work in either image- or object-space (e.g., shadow maps [Williams 1978] or volumes [Crow 1977]). Generally, the cost-performance tradeoff favors shadow mapping, as constant-time shadow lookups make up for z-buffer memory overhead. But shadow maps were designed for surfaces; volumetric variants (e.g., [Dobashi et al. 2002]) require redundant and incoherent z-buffer lookups.

While shadow volumes analytically identify shadowed regions, cost depends on geometric complexity and robustness requires care [Everitt and Kilgard 2002]. However, lighting participating media with shadow volumes is straightforward [Biri et al. 2006].

Hybrid techniques seek the benefits of both image- and object-space shadows. McCool [2000] and Billeter [2010] robustly generate shadow volumes from a z-buffer, but do not address the fill rate required to render. Chan [2004] applies shadow volumes only near aliased shadow map discontinuities, but no longer provides volumetric visibility. Wyman [2008] reduces incoherent shadow map accesses but introduces a dependence on geometric complexity. More recently, voxelized shadow volumes (VSVs) [Wyman 2011] create a *discrete* shadow volume, maintaining both a constant-time query and volumetric shadow representation.

[*]e-mail:chris.wyman@acm.org
[†]e-mail:zeng-dai@uiowa.edu

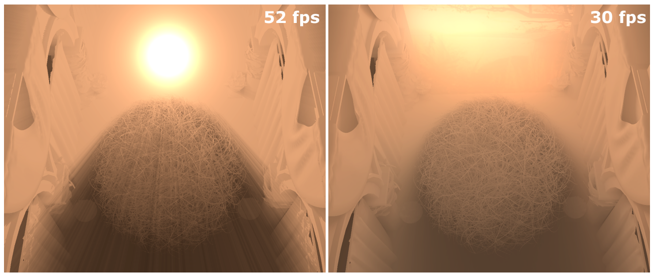

Figure 2: *Shadows from (left) a point and (right) an area light.*

Current interactive methods cannot shadow participating media from area lights, though many approximations exist for surfaces [Woo and Poulin 2012] and offline volumetric shadows are well understood [Kulla and Fajardo 2012]. Our imperfect voxelized shadow volumes allow dynamic soft shadows inside media, giving dramatically different results from hard shadows (see Figure 2).

We take a slow, brute-force approach (summing voxelized shadow volumes results at many samples on an area light) and identify its inefficiencies. This provides insights to help reduce overhead and amortize costs over samples: an appropriate volume resolution avoids oversampling and allows parallel VSV creation; a fast interpolation scheme improves quality; and imperfect VSVs reduces geometry sampling costs. These allow rendering shadows from 256 virtual point lights (VPLs) at under 4x the cost of a single, high quality voxelized shadow volume.

This paper has four main contributions:

- a novel soft shadow algorithm for participating media;
- an approach for parallel construction of hundreds of VSVs;
- outlining trade-offs in VSV resolution vs. quality;
- a fast VSV interpolation that reduces shadow banding.

2 Background

Substantial recent research has explored media rendering (e.g., [Novák et al. 2012]). While giving stunning results, many methods focus on quality over performance. Below we focus on work prioritizing interactivity, in addition to a brief theory review.

2.1 Light Transport in Media

Radiance along a ray through media depends on the scattering coefficient $\sigma_s(\mathbf{x})$, absorption coefficient $\sigma_a(\mathbf{x})$, extinction coefficient $\sigma(\mathbf{x}){=}\sigma_s(\mathbf{x}){+}\sigma_a(\mathbf{x})$, phase function $p(\vec{\omega}, \vec{\omega}')$, and field radiance $L_{in}(\mathbf{x}, \vec{\omega}')$:

$$L(\mathbf{x}, \vec{\omega}) = \int_0^d \sigma_s(u)\; E(u) \int_\Omega p(\vec{\omega}, \vec{\omega}') L_{in}(u, \vec{\omega}') d\vec{\omega}' du, \quad (1)$$

where $E(u) = \mathbf{e}^{-\int_0^u \sigma(t)dt}$ and d the distance to the nearest surface along ray $\vec{\omega}$. For homogeneous media σ_s, σ_a, and σ are constant. With a point light the inner integral disappears, replaced by $V(u, \vec{\omega}')$ to query light visibility in direction $\vec{\omega}'$. This leads to the standard airlight equation [Nishita et al. 1987]:

$$L(\mathbf{x}, \vec{\omega}) = \int_0^d \sigma_s \mathbf{e}^{-\sigma u} V(u, \vec{\omega}') p(\vec{\omega}, \vec{\omega}') L_{in}(u, \vec{\omega}') du. \quad (2)$$

In this paper we sample an area light as multiple VPLs, turning the second integral into a sum:

$$L(\mathbf{x}, \vec{\omega}) = \int_0^d \sigma_s \mathbf{e}^{-\sigma u} \left[\sum_{\ell_i \in A} V(u, \vec{\omega}_i) p(\vec{\omega}, \vec{\omega}_i) L_{in}(u, \vec{\omega}_i) \right] du, \; (3)$$

where A is the light's surface. Reversing integration and summation order allows us to apply Equation 2 once per VPL:

$$L(\mathbf{x}, \vec{\omega}) = \sum_{\ell_i \in A} \left[\int_0^d \sigma_s \mathbf{e}^{-\sigma u} V(u, \vec{\omega}_i) p(\vec{\omega}, \vec{\omega}_i) L_{in}(u, \vec{\omega}_i) du \right]. \; (4)$$

Equations 2 and 4 integrate quite smooth functions, allowing coarse SVD approximation [Chen et al. 2011] to replace the integral by the sum of a few terms. We use a somewhat cruder approximation suggested by Dobashi et al. [2002]:

$$L(\mathbf{x}, \vec{\omega}) \approx \sum_{\ell_i \in A} \left[\int_0^d V(u, \vec{\omega}_i) du \int_0^d \sigma_s \mathbf{e}^{-\sigma u} p(\vec{\omega}, \vec{\omega}_i) L_{in}(u, \ell_i) du \right],$$

which factors per-VPL visibility and scattering. The second integral can be computed analytically (e.g., Sun et al. [2005]), and since VSVs discretize visibility the first integral becomes a sum.

2.2 Analytic Scattering Computations

Solving the airlight integral can be costly, so many researchers have sought methods for interactive airlight evaluation. Sun et al. [2005] tabulate solutions for isotropic phase functions ($p{=}\frac{1}{4\pi}$). Hong et al. [2006] represent anisotropic phase functions via Legendre polynomial series. Pegoraro et al. [2009; 2009; 2010] analytically solve the airlight equation and generalize for anisotropic media and lights (e.g., spotlights). Unfortunately all these neglect visibility, leaving the scattering too bright in shadowed regions.

2.3 Shadows in Homogeneous Participating Media

Building on unshadowed airlight, numerous researchers have added visibility. Max [1986], James [2003], and Biri et al. [2006] split Equation 2 in separate intervals (for $V(u, \vec{\omega}')$ entirely 0 or 1) via shadow volumes, computing analytic airlight in illuminated intervals. They differ in interval computation: Max sorts offline, James orders via depth peeling, and Biri uses stenciled shadow volumes. Billeter et al. [2010] generate shadow volumes from a shadow map, separating rendering costs from geometric complexity.

Ray casting emits rays through each pixel, sampling light visibility along each. Either slicing via screen-aligned quads [Dobashi et al. 2002] or marching each pixel's ray [Lefebvre and Guy 2002] can generate these samples; both query shadow map visibility once per sample before summing their contributions. Without guaranteed sampling on shadow boundaries, only dense sampling avoids aliasing. But querying the shadow map every sample is costly.

To reduce visibility queries, Wyman and Ramsey [2008] coarsely bound shadowed regions with shadow volumes, but this introduces costs that vary with scene complexity. Toth and Umenhoffer [2009] apply interleaved sampling in image space. Engelhardt and Dachsbacher [2010] sample and interpolate along epipolar planes, where volumetric shadows only change at depth discontinuities. Chen et al. [2011] use min-max mipmaps to non-uniformly step along view rays and approximate the airlight integral with a coarse sum.

Voxelized shadow volumes [Wyman 2011] densely sample per-ray visibility via a cache-coherent data structure storing many visibility samples in each texel. We build on voxelized shadow volumes.

2.4 Heterogeneous Participating Media

While not our focus, similar techniques can apply to heterogeneous media. Kim and Neumann [2001] create a multi-layer light-space map, with opacity rather than depth, allowing approximate reconstruction of light transmittance. Jansen and Bavoil [2010] store opacity using Fourier coefficients and sample via per-pixel ray marching. Delalandre et al. [2011] improve transmittance sampling from such maps. Salvi et al. [2010] compute a piecewise constant opacity function for each shadow map texel. Unfortunately, these methods scale poorly with additional lights.

Some techniques [Zhou et al. 2007; Zhou et al. 2008] use radial basis functions to represent heterogeneous media, allowing preintegration of extinction and inclusion of complex lighting via spherical harmonic PRT. However, these require substantial precomputation.

2.5 Direct Illumination from Area Lights

Various research aims for interactive soft shadows [Woo and Poulin 2012], mostly split between object- and image-space methods. Penumbra wedges [Akenine-Möller and Assarsson 2002] consume significant fill rate, as do shadow volumes, though recent work uses wedges to shadow participating media at near-interactive rates [Forest and Segovia 2011].

Shadow mapping can generate soft shadows by blurring [Reeves et al. 1987], precomputing visibility [Ritschel et al. 2007], or generating many imperfect shadow maps (ISMs) [Ritschel et al. 2008]. These methods do not currently work in media, but we leverage imperfect shadow maps to generate many voxelized shadow volumes.

Voxel-based shadow queries work for hierarchical ray-based [Nichols et al. 2010] and cone-based lighting [Crassin et al. 2011], but neither illuminates participating media from area lights.

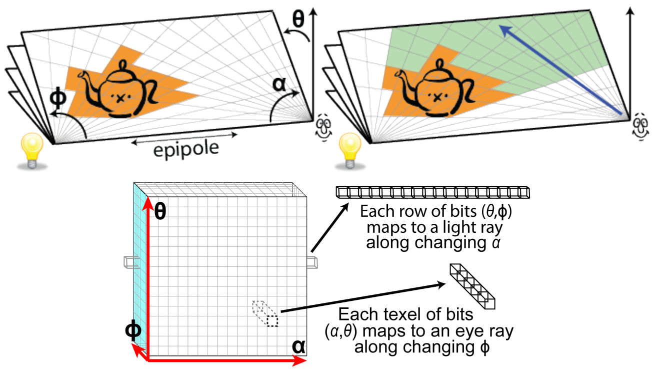
θ
ϕ
α
epipole
θ
ϕ
α
Each row of bits (θ,ϕ) maps to a light ray along changing α
Each texel of bits (α,θ) maps to an eye ray along changing ϕ

Figure 3: *Voxelized shadow volume review. (Top) Geometry is voxelized to an epipolar grid specified by angles (α, θ, ϕ). A prefix sum extrudes occlusion along the axis of constant ϕ. Each texel stores all binary visibility sampled along a ray of constant (α,θ), e.g., the blue ray. (Bottom) A 2D* `uvec4` *texture stores epipolar space, with (α, θ) corresponding to (x, y) and ϕ to individual texel bits. Eye rays map to a single texel, with each texel bit representing light visibility at a ray sample. Rectification resamples the shadow map for every (θ_i, ϕ_i), converts the depth to α_i, and sets voxel $(\alpha_i, \theta_i, \phi_i)$.*

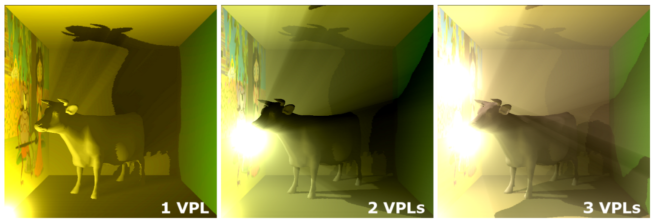

Figure 4: *A naive area lighting approach could render new VSVs for each VPL. But costs scale linearly, at roughly 7.2 ms for creation and use of each 512^2 VSV (up to 1850 ms with 256 VPLs).*

2.6 Voxelized Shadow Volume Review

Before introducing our work we review voxelized shadow volumes, a discrete shadow volume sampled in epipolar space. VSVs use brute force sampling, optimizing computations and lookups for GPUs rather than suggesting a faster algorithmic approximation. The volume is parameterized by epipolar angles, using axis-aligned light and eye rays.

VSVs use three passes (see Figure 3): voxelize geometry to epipolar space, extrude occluded voxels away from the light, and lookup all visibility samples along a view ray simultaneously. Many techniques can voxelize geometry; for our work, the fastest and most robust method resamples a shadow map to fill in the nearest occluder along every light ray (i.e., the nearest α for each (θ, ϕ)-pair).

3 Imperfect Voxelized Shadow Volumes

We initially implemented a brute force area lighting scheme, sampling VPLs on the light and using independent voxelized shadow volumes for each (see Figure 4). The costs for this naive approach increase linearly with VPL count. While one VSV runs interactively, eliminating banding requires at least 64 VPLs (costing over 450 ms per frame), making this naive approach infeasible.

3.1 Algorithm Overview

In this naive approach, the parallel scan clearly proves the key bottleneck (see results in Table 1); however, other stages also take considerable time. Our new approach speeds all stages by at least an order of magnitude, and is split into five steps:

1. Sample VPLs to a buffer
2. In parallel, render imperfect shadow maps for all VPLs
3. In parallel, rectify ISMs into a voxel grid for each VPL
4. In parallel, perform voxel-space prefix sum to create VSVs
5. To gather, index all VSVs and accumulate visibility

At a high level, this is a simple improvement to the original VSV algorithm. We describe efficient parallelization of steps 2 through 5 over hundreds of VPLs, avoiding the brute force serial implementation needed to individually accumulate contributions from each.

3.1.1 Step 1: Sample VPLs

Before computing visibility, we need to identify VPLs. This can occur many ways: precomputing samples, importance sampling, or regular or jittered sampling. In this paper, we sample (via a Halton sequence) a rectangular area light textured with a video, storing per-VPL position, color, and normal into a buffer. This easily extends to environment maps or other complex lights.

3.1.2 Step 2: Render Imperfect Shadow Maps

To create VSVs we need voxelized geometry in separate, per-VPL epipolar spaces. Currently, the fastest epipolar voxelization method resamples shadow maps. To generate shadow maps for each VPL we use imperfect shadow mapping [Ritschel et al. 2008], which efficiently render a multiview z-buffer in a single framebuffer.

We use the ISM algorithm unmodified: creating a point cloud representation of the scene, rendering a subset of the points to each shadow map, and a push-pull algorithm for hole filling. Some well-sampled scenes may not need hole filling; imperfect voxelized shadows degrade gracefully for poor samplings, which causes gradually increasing light leakage.

3.1.3 Step 3: Resample Imperfect Shadow Maps

After generating imperfect shadow maps, we resample in parallel (see Figure 5). Rectification cost depends on epipolar space θ and ϕ resolutions, as each (θ, ϕ)-pair generates one voxel (see Figure 3). As (θ, ϕ)-resolution decreases, pass overhead becomes significant. Combining all rectification passes into one over a buffer containing all epipolar spaces reduces overhead significantly. Resampling to one 128^3 epipolar grid takes 0.15 ms; amortizing overhead on 32, 64, or 256 epipolar spaces provides a speedup of 7x, 9x, or 10x, respectively. (I.e., rectification costs 0.020, 0.017, and 0.015 ms.)

3.1.4 Step 4: Parallel Scan to Create VSVs

After Section 3.1.3, our framebuffer now stores the geometry voxelized to multiple epipolar spaces. We now need a per-space parallel scan (with bitwise OR operator) to extrude occlusion away from each VPL. As above, separate per-VPL passes incur a big overhead.

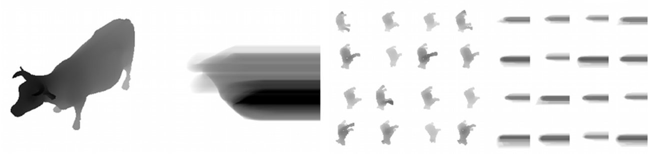

Figure 5: *VSVs convert each shadow map to a voxelized shadow volume (left). Imperfect shadow maps store many depth buffers in one framebuffer; imperfect voxelized shadow volumes turn each into a shadow volume in its own epipolar space (right).*

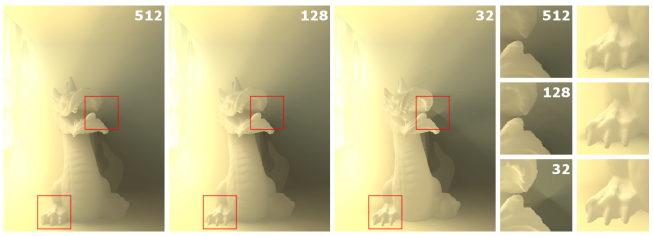

Figure 6: *Simultaneously varying the shadow map and α, θ, and ϕ VSV resolutions, all using 128 VPLs on the area light.*

We run the VSV prefix sum simultaneously over all epipolar spaces, allowing the GPU to extract additional parallelism.

Along each row in a single epipolar space, the scan runs across the entire image width. In a tiled imperfect voxelized shadow volume, we run a single segmented parallel scan along each row; the number of segments depends on the tiled width of each epipolar space (see Figure 5). A scan on one 128^3 epipolar space takes 0.41 ms. Amortizing over 32, 64, or 256 VPLs provides a speedup of 15x, 19x, or 23x. (I.e., scans require 0.028, 0.022, and 0.018 ms per space.)

3.1.5 Step 5: Accumulating Scattering from All VPLs

After Section 3.1.4, we have one buffer containing all our VSVs. Our final step sums per-VPL visibility and scattering in a final gather. As discussed in Section 2.1, each fragment needs to solve:

$$L(\mathbf{x},\vec{\omega}) \approx \sum_{\ell_i \in A} \left[\int_0^d V(u,\vec{\omega}_i)du \int_0^d \sigma_s \mathbf{e}^{-\sigma u} p(\vec{\omega},\vec{\omega}_i) L_{in}(u,\ell_i)du \right],$$

where $\int V(u,\vec{\omega}_i)du$ becomes a weighted average of visibility bits between $[0..d]$ in the per-VPL shadow volume texel (α,θ). We compute the second integral using an analytic airlight solution, such as Sun et al. [2005].

While straightforward, we provide shader pseudocode for clarity:

```
$\mathbf{F} \leftarrow$ FragmentPosition()
$\vec{N} \leftarrow$ FragmentNormal()
$d \leftarrow$ DistanceToVisibileFragment()
vec3 volColor = 0, surfColor = 0
for $i \in [0..\text{numVPLs}]$ do
    $\mathbf{c}_i$, $\vec{N}_i$, $\mathbf{L}_i \leftarrow$ LoadVPLColorNormalPosition($i$)
    vsvCoord$_i \leftarrow$ ComputeVSVCoordinate($i$,$\mathbf{L}_i$,$\mathbf{F}$)
    vis$_i \leftarrow$ ReadImperfectVSV(vsvCoord$_i$)
    surf$_i \leftarrow$ QueryVisibilityAtFragment(vis$_i$,$d$)
    surfColor += surf$_i$ * $\mathbf{c}_i$ * PhongColor($\mathbf{F}$,$\vec{N}$,$\vec{N}_i$,$\mathbf{L}_i$)
    vol$_i \leftarrow 1-$ [ CountBitsBeforeFrag(vis$_i$,$d$) / TotalBitsBeforeFrag($d$) ]
    volColor += vol$_i$ * $\mathbf{c}_i$ * AnalyticAirlight($d$,$\vec{N}_i$,$\mathbf{L}_i$)
end for
trans $\leftarrow$ ComputeMediaTransmittance($d$)
return ( trans*surfColor + $(1 -$ trans)*volColor )
```

Here, `CountBitsBeforeFrag` counts all shadowed bits in front of a fragment at distance d and `TotalBitsBeforeFrag` counts the total bits to a fragment at d. For clarity, this pseudocode uses a uniform average of visibility bits, but adding a weighted average is straightforward.

3.2 Speeding VSVs: Lower Resolution Sampling

Above we parallelized voxelized shadow volume creation, but a key question is appropriately sizing each VSV. Resolution affects performance for all steps from Section 3.1. Z-buffer size influences shadow map creation; epipolar sampling costs vary with θ and ϕ; α and θ counts impact the prefix sum; and ϕ sampling varies texture bandwidth when gathering. Ideally, we would select the minimal resolutions that avoids significant quality degradation after averaging the contributions of many VPLs.

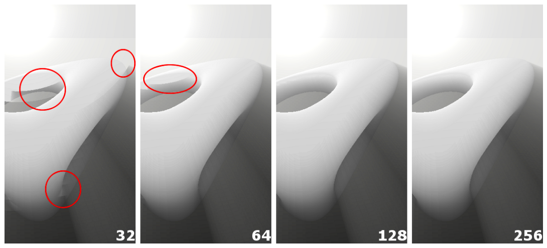

Figure 7: *Shadow detachment for a single VSV with varying α resolution. Media density and image contrast increased for visibility.*

Figure 6 shows quality at various resolutions with 128 VSVs. Over numerous scenes, two key artifact types appear with lower resolution: occluder shadow detachment and aliased shadow boundaries. Interestingly, these are the same artifacts introduced by shadow mapping. Using VSVs for surface shadows, both problems arise.

In media, detached shadows have little visual impact. Thin occluders cast faint shadows, barely visible even when correct. And only large detachment affects bigger objects, as back faces occlude errors from small detachments (see Figure 7). Averaging errors over many VPLs, shadows detach only at the coarsest resolutions.

As with all sampling methods, low resolutions introduce aliasing. With over 128 VPLs, averaging eliminates most aliasing if epipolar grids are uncorrelated. ISMs often need 1024 VPLs to converge [Ritschel et al. 2008], but media has less visual impact, allowing convergence with fewer uncorrelated samples. But our queries need to avoid sample correlation throughout the volume, not just on surfaces. Even intelligently sampled VPLs can have correlated samples in select regions (see Figure 6, inset), leading to strong banding. Interpolation of VSV samples is vital to avoid these artifacts.

Figure 8 shows a single VSV under varying resolutions. Good hard shadows need at least 512 samples on all axes. For area lights, sampling gross shadow structure is key, as averaging usually blurs aliasing from individual VSVs. To maximize speed, no dimension should be sampled too finely; final quality depends on the worst-sampled axis, so adding samples on other axes slows performance without improving quality. A resolution of 128 in all dimensions (shadow map size, α, θ, and ϕ) works well for imperfect VSVs.

3.3 VSV Interpolation

Reducing shadow aliasing improves visual quality. Adding VPLs or increasing resolution clearly achieves this, albeit at high cost. One cheaper alternative is interpolation. Unlike non-linear z-buffer depths, binary VSV visibility can be naively interpolated. But the binary payload makes prefiltering (e.g., mipmaps) impossible; instead interpolation must occur after the query.

Consider the binary epipolar grid (see Figure 9). Initially, one might consider interpolating along all three axes (α, θ, and ϕ). But by construction, visibility changes at most once along α (at the frontmost occluder); interpolating across such boundaries is incorrect. Also by construction, voxels of varying ϕ occupy multiple bits in a single VSV texel. Thus, ϕ interpolation requires no additional texture lookups, just additional bit-twiddling on existing texels.

This means instead of trilinear interpolation, VSVs only need two texel lookups. We do not interpolate along α, we read two adjacent texels along θ, and use bit-twiddling to interpolate along ϕ.

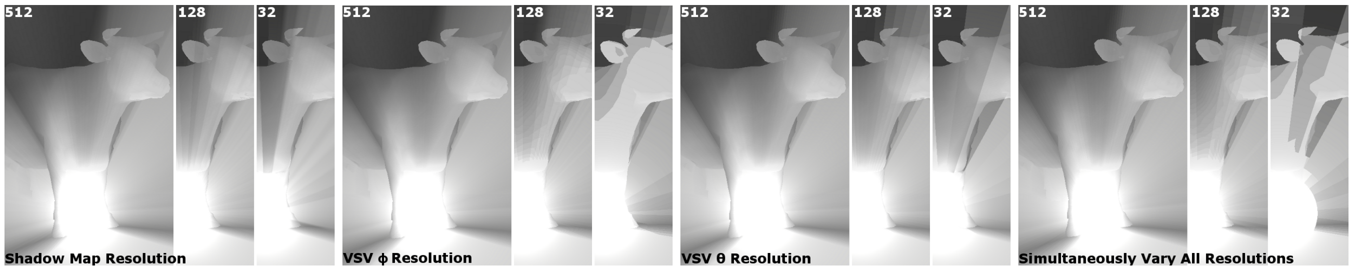

Figure 8: *Evaluating quality versus sampling rate for individual voxelized shadow volumes. (Left) Varying only shadow map resolution; (center left) only ϕ resolution; (center right) only θ resolution; and (right) simultaneously varying α,θ,ϕ, and shadow map resolutions.*

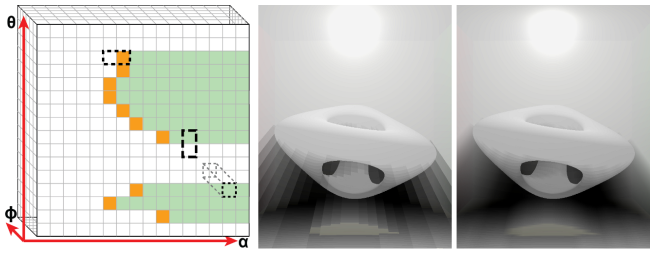

Figure 9: *(Left) Epipolar space has axes α, θ, and ϕ. α only varies at occluder surfaces (orange voxels); blurring here introduces light leakage. Voxels with varying ϕ reside in one texel, so VSV interpolation needs just two texel lookups (at adjacent θ values). (Center) A very coarse VSV without and (right) with shadow interpolation.*

But naive interpolation for surface shadows gives extremely poor performance. We view `uvec4` texels as streams of 128 individually accessible bits. Current GPUs are not designed for this usage pattern, resulting in convoluted ASM from simple GLSL. Appendix A provides our fast implementation and insights into these performance issues.

Unlike surface shadows, interpolating volume visibility is straightforward and naive implementations perform efficiently. Volumetric visibility along θ can usually be interpolated *after* accumulating scattering (i.e., in the last pseudocode line in Section 3.1.5). Interpolating visibility between samples along ϕ is often unnecessary; $\int V(u,\ell_i)du$ is the same independent of linear interpolation (except in the partial voxel from $\lfloor d \rfloor$ to d).

3.4 Gathering at Low Resolution and Upsampling

After significantly speeding shadow volume creation, the final scattering accumulation (described in Section 3.1.5) becomes the bottleneck at 80% of render time. While we saw a speedup over the naive approach due to common computation reuse and better use of texture resources, this proved fairly minor (only 2x).

Given that scattering in media changes smoothly, various common approaches reduce final gather costs. We tested both interleaved sampling and a low resolution gather. Our interleaved sampling accumulates from $\frac{1}{9}$ the VPLs per pixel (so every 3×3 screen-space window samples all VPLs) and a bilateral filter gives our final result. For our downsampled gather, we found accumulating at $\frac{1}{4}$ resolution followed by a bilateral upsample works well. Because VSV texture bandwidth is the major accumulation cost, sampling fewer pixels or using fewer VPLs per pixel provide near-linear speedups (e.g., 9x for interleaved and 4x for low resolution sampling).

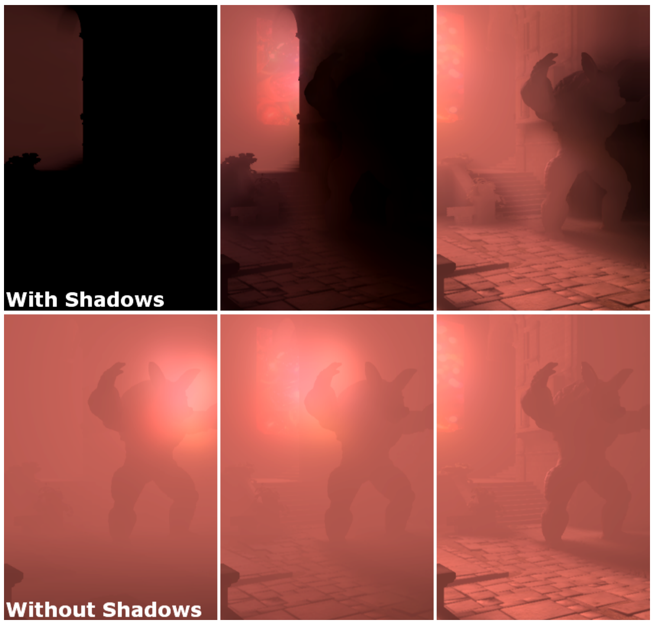

Figure 10: *Comparison of scattering with and without shadows in a sequence of images with a moving light; runs at 25.7 fps.*

4 Results

We implemented imperfect voxelized shadow volumes in OpenGL. All timings use a Core i7-3820 with a Quadro K5000. Unless otherwise specified, images are captured at 1024^2 with each VPL using a 128^2 shadow map and 128^3 voxelized shadow volume.

Table 1 compares performance of imperfect voxelized shadow volumes with the brute force application of per-VPL shadow volumes for the scene in Figure 4. Note the timings differ from those in Figure 4 due to differing VSV resolution. Using 256 VPLs, we achieve a 20x speedup for VSV creation. For the final gather, we achieve a 1.5-2x performance improvement, from reduced pass overhead and OpenGL state changes. Interleaved and low resolution sampling provide a further speedup for the final gather, shown in Table 2.

With careful implementation, all our steps scale linearly with increasing VPL counts. As all 256 epipolar spaces reside in a single 2048^2 framebuffer (i.e., 16×16 tiles of 128^2), the shadow map push-pull, rectification, and parallel scans should only execute on buffer regions containing valid VPLs. With hundreds of VPLs, the incremental cost of new lights is $\sim$0.1 ms in typical scenes.

Figures 1, 2, 6, 10, 11, 12, and 13 show imperfect voxelized shadow volumes cast from large area lights. The light acts like a large television, with color specified by a video texture. Figures 1 and 10 compare media with and without volumetric shadows; Figure 11 is

	Step Timings (in msec) and Speedup Ratios												Total Frame Cost		
	Shadow map			Voxelization			Parallel scan			Final render (full res)					
	Naive	IVSV	Speedup	Naive	IVSV	Speedup	Naive	IVSV	Speedup	Naive	IVSV	Speedup	Naive	IVSV	Speedup
1 VPL	0.44	1.66	**0.3x**	0.24	0.24	**1.0x**	0.47	0.47	**1.0x**	0.39	0.39	**1.0x**	1.54	2.76	**0.6x**
4 VPLs	1.22	1.23	**1.0x**	1.01	0.30	**3.4x**	1.99	0.48	**4.2x**	1.57	0.92	**1.7x**	5.79	2.93	**2.0x**
16 VPLs	4.31	1.12	**3.7x**	3.89	0.44	**8.8x**	7.73	0.62	**12.5x**	6.19	3.26	**1.9x**	22.12	5.44	**4.1x**
64 VPLs	16.73	2.00	**8.4x**	15.67	1.20	**13.1x**	30.94	1.55	**20.0x**	24.89	12.63	**2.0x**	88.23	17.38	**5.1x**
256 VPLs	66.25	3.58	**18.6x**	62.37	3.75	**16.6x**	123.73	4.79	**25.8x**	99.48	50.26	**2.0x**	351.83	62.38	**5.6x**

Table 1: *Performance for Figure 4 using naive, per-VPL VSVs and imperfect voxelized shadow volumes (both with* 128^3 *epipolar grids). We achieve speedups over an order of magnitude, except for our full-screen gather. Our shadow map performance initially underperforms; all VPL counts use a fixed point set to generate ISMs and timings include the push-pull. Performance initially improves for higher VPLs, as framebuffer contention decreases when points are scattered more sparsely through a larger set of shadow maps.*

	Hairball (Figure 2)	Angel (Figure 1)	City (Figure 10)
Shadow Map	9.7 ms **(30x)**	9.1 ms **(29x)**	16.5 ms **(26x)**
Voxelization	4.3 ms **(30x)**	4.3 ms **(30x)**	4.1 ms **(26x)**
Scan	4.8 ms **(26x)**	4.8 ms **(26x)**	4.8 ms **(26x)**
Full Res Gather	77.7 ms **(1.6x)**	77.7 ms **(1.6x)**	77.7 ms **(1.6x)**
3x3 Interleaved	12.4 ms (n/a)	11.8 ms (n/a)	11.8 ms (n/a)
$\frac{1}{4}$ Upsample	19.0 ms (n/a)	19.1 ms (n/a)	19.0 ms (n/a)
Total Frame (w/ Interleaved)	33.6 ms **(21x)**	32.5 ms **(20x)**	38.9 ms **(20x)**

Table 2: *Timings for various scenes using 256 VPLs, with speedups versus brute force in parenthesis. We did not implement interleaved or low-resolution gathers for our brute-force approach. Total frame times include shadow map creation, voxelization, scan,* 3×3 *interleaved gather, plus some overhead (e.g., G-buffer creation).*

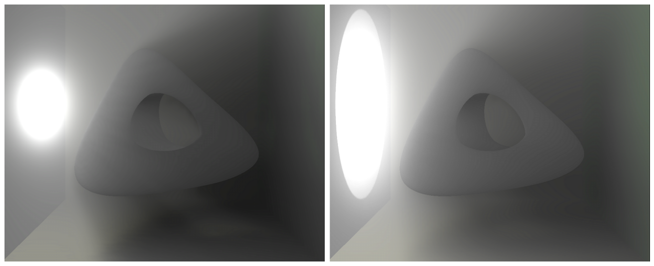
Figure 11: *Imperfect shadow volumes under varying sized lights.*

illuminated by a subset of the light (part of the light texture is black) and shows how our shadows behave with increasing light size. Figure 13 compares IVSVs with and without voxel interpolation.

Figure 12 compares our results with a ground truth ray tracing. We also timed this scene with varying resolution IVSVs to show performance scaling. For simplicity, our imperfect shadow volume implementation is hardcoded to use 128 samples in ϕ. When using $32^2 \times 128$, $64^2 \times 128$, 128^3, and $256^2 \times 128$ volumes, frame costs were 19.5, 19.7, 26.8, and 43.8 ms, respectively. Of this, around 15.8 ms was for the $\frac{1}{4}$ resolution gather pass.

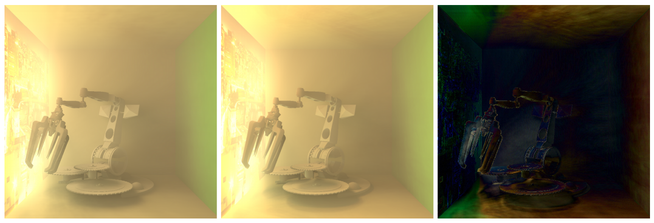
Figure 12: *(Left) Imperfect voxelized shadow volumes, (center) ground truth ray tracing, (right) a 4x difference image.*

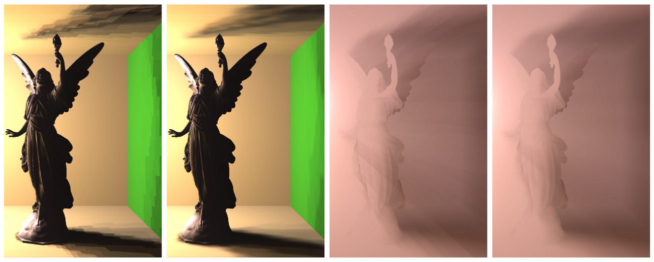
Figure 13: *Using 8 VPLs, shadows are quite aliased. Our interpolation for (left) surface and (right) volumetric shadows.*

5 Discussion: Robustness and Performance

As in analytic shadow volumes, voxelized shadow volumes use some tricks to achieve robustness. Dynamic range selection for all epipolar angles (α, θ, ϕ) is vital, especially for low resolution voxel grids. For offscreen lights, only a small subset of epipolar space is visible; shadow samples should focus in the visible subset of space.

Adding a bias to voxel indices reduces aliasing, similar to a shadow map bias. Shadowed samples should be slightly pushed away from the light and the eye. We found a α_{bias} of -2.25 voxels and ϕ_{bias} of -1 voxels to work well for most resolutions.

On some GPUs, the 128th bit of a `uvec4` is unaddressable. This causes light leakage in regions of epipolar space. Imperfect VSVs average away this artifact and in 128^3 epipolar grids this ϕ bit rarely impacts pixel color, so we ignore the issue. For larger grids or fewer VPLs, we suggest using only 127 bits per texel in a render target.

At the epipole, voxels become extremely elongated; only a single voxel contains the epipole in any epipolar plane, causing extreme undersampling. Fortunately, here a shadow query logically means, ”is anything between the eye and light?” Comparing the fragment position along the epipole with the light location answers this question more accurately than a VSV query.

Our imperfect shadow map code uses a simple point sampling of geometry, treating all samples the same. Using a point-LOD (e.g., [Hollander et al. 2011]) would improve quality by ensuring geometry close to the light appears in all shadow maps. Currently, we oversample geometry (slowing ISM creation) to avoid light leaks caused by missing occluders near the light.

6 Conclusion

We introduced *imperfect voxelized shadow volumes*, an algorithm combining voxelized shadow volumes and imperfect shadow maps that allows interactive volumetric visibility from area lights, enabling shadows inside homogeneous single-scattering participating media. Our key contributions include parallelizing voxelized shadow volume creation from multiple point lights to reduce pass overhead, exploring appropriate volume resolutions for area lights, and a fast VSV interpolation scheme to efficiently eliminate volumetric and surface banding.

There are numerous ways to improve imperfect voxelized shadow volumes. We see applications to dynamic heterogeneous media, perhaps using non-binary voxel payloads. Additionally new gathering techniques may speed scattering computations, reducing the current bottleneck.

Acknowledgments

The authors would like to thank Hang Dou, Ethan Kerzner, Yajie Yan, Peter-Pike Sloan, Pete Shirley, and Dave Luebke for feedback on the ideas, implementation, and paper drafts. Models come from the Stanford, AIM@Shape, and Utah 3D Animation repositories. We also thank the anonymous reviewers for their feedback.

References

AKENINE-MÖLLER, T., AND ASSARSSON, U. 2002. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Eurographics Rendering Workshop*, 309–318.

BILLETER, M., SINTORN, E., AND ASSARSSON, U. 2010. Real time volumetric shadows using polygonal light volumes. In *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.

CHAN, E., AND DURAND, F. 2004. An efficient hybrid shadow rendering algorithm. In *Eurographics Symposium on Rendering*, 185–196.

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

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum 30*, 7, 1921–1930.

CROW, F. 1977. Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH*, 242–248.

DELALANDRE, C., GAUTRON, P., MARVIE, J.-E., AND FRANÇOIS, G. 2011. Transmittance function mapping. In *Interactive 3D Graphics and Games*, 31–38.

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

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

EVERITT, C., AND KILGARD, M. 2002. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Tech. rep., NVIDIA, http://arxiv.org/abs/cs.GR/0301002.

FOREST, V., AND SEGOVIA, B., 2011. Object-based shadowed volumetric single-scattering for point and area lights. http://vaplv.free.fr/vsv.php.

HOLLANDER, M., RITSCHEL, T., EISEMANN, E., AND BOUBEKEUR, T. 2011. Manylods: Parallel many-view level-of-detail selection for real-time global illumination. *Computer Graphics Forum 30*, 4, 1233–1240.

HONG, R., HO, T.-C., CHUANG, J.-H., SHIU, R.-M., AND KUO, R. 2006. A real-time analytic lighting model for anisotropic scattering. In *Computer Graphics Workshop*.

JAMES, R. 2003. *Graphics Programming Methods*. Charles River Media, ch. True volumetric shadows, 353–366.

JANSEN, J., AND BAVOIL, L. 2010. Fourier opacity mapping. In *Interactive 3D Graphics and Games*, 165–172.

KIM, T.-Y., AND NEUMANN, U. 2001. Opacity shadow maps. In *Eurographics Rendering Workshop*, 177–182.

KULLA, C., AND FAJARDO, M. 2012. Importance sampling techniques for path tracing in participating media. *Computer Graphics Forum 31*, 4, 1519–1528.

LEFEBVRE, S., AND GUY, S., 2002. Volumetric lighting and shadowing. http://sylefeb.aracknea-core.net/cgshaders/vshd/.

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.

NICHOLS, G., PENMATSA, R., AND WYMAN, C. 2010. Interactive, multiresolution image-space rendering for dynamic area lighting. *Computer Graphics Forum 29*, 4, 1279–1288.

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.

NOVÁK, J., NOWROUZEZAHRAI, D., DACHSBACHER, C., AND JAROSZ, W. 2012. Progressive virtual beam lights. *Computer Graphics Forum 31*, 4.

PEGORARO, V., AND PARKER, S. G. 2009. An analytical solution to single scattering in homogeneous participating media. *Computer Graphics Forum 28*, 2, 329–335.

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

PEGORARO, V., SCHOTT, M., AND PARKER, S. G. 2010. A closed-form solution to single scattering for general phase functions and light distributions. *Computer Graphics Forum 29*, 4, 1365–1374.

REEVES, W., SALESIN, D., AND COOK, R. 1987. Rendering antialiased shadows with depth maps. In *Proceedings of SIGGRAPH*, 283–291.

RITSCHEL, T., GROSCH, T., KAUTZ, J., AND MUELLER, S. 2007. Interactive illumination with coherent shadow maps. In *Eurographics Symposium on Rendering*, 61–72.

RITSCHEL, T., GROSCH, T., KIM, M., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics 27*, 5, 1–8.

SALVI, M., VIDIMCE, K., LAURITZEN, A., AND LEFOHN, A. 2010. Adaptive volumetric shadow maps. *Computer Graphics Forum 29*, 4, 1289–1296.

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.

TOTH, B., AND UMENHOFFER, T. 2009. Real-time volumetric lighting in participating media. In *Eurographics (Short Papers)*, 57–60.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH*, 270–274.

WOO, A., AND POULIN, P. 2012. *Shadow Algorithms Data Miner*. A. K. Peters/CRC Press.

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

WYMAN, C. 2011. Voxelized shadow volumes. In *High Performance Graphics*, 33–40.

ZHOU, K., HOU, Q., GONG, M., SNYDER, J., GUO, B., AND SHUM, H.-Y. 2007. Fogshop: Real-time design and rendering of inhomogeneous, single-scattering media. In *Pacific Graphics*, 116 –125.

ZHOU, K., REN, Z., LIN, S., BAO, H., GUO, B., AND SHUM, H.-Y. 2008. Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics 27*, 3, 36:1–36:12.

A Fast Bit-Twiddling For VSV Interpolation

While GPUs now treat integers as first-class datatypes, each vector element is treated separately. VSVs use a `uvec4` as a 128-bit voxel stream and extract single bits based on a [0..127] index. This means component-wise GLSL functions (and underlying hardware instructions) like `bitfieldExtract` and `bitCount` are insufficient; additional logic is needed to select individual VSV voxels.

At first we did not consider this issue, but our initial interpolation code increased gather costs by a factor of three! Given our gather remains a bottleneck, this significantly reduced performance.

Three issues contributed to this cost: our voxelized shadow volume code [Wyman 2011] uses older GLSL shaders without `bitCount` and many float-to-int and int-to-float conversions; GPUs are optimized for floats, so integer performance is subpar; and dynamic vector-component selection is not allowed. Writing a fast interpolator balancing register pressures, datatype conversions, and component selection was actually quite challenging.

Listing 1 shows our best two individual VSV voxel queries. Version A selects the `uvec4` component containing the queried bit, zeros out other bits, and compares to 0. However, `voxels[x]` may compile inefficiently (for us, it alone generated 24 ASM instructions), so manual component selection (version B) performs significantly better.

Based on this efficient voxel query, we designed an efficient function to interpolate surface shadows from a VSV. Listing 2 shows this code. Much is similar, including the bit selection and dynamic component indexing. To avoid numerous type conversions when multiplying voxel values and their interpolation weights, we leverage the binary voxel values to use `mix` (to perform an equivalent selection) and then use a dot product to sum the weights.

```
// Clean version.  Compiles to 35 ASM instructions
bool isShadowed_A( uvec4 voxels, int phiBit )
{
  return ( voxels[phiBit>>5] & (1<<(phiBit&31)) ) != 0;
}

// Fast version.  Compiles to 20 ASM instructions
bool isShadowed_B( uvec4 voxels, int phiBit )
{
  uint comp = phiBit>>5, compBit = 1<<(phiBit&31);
  uvec2 val = (comp>1) ? voxels.zw : voxels.xy;
  val.x = (comp&1)==0 ? val.x : val.y;
  return (val.x & compBit) != 0;
}
```

Listing 1: *Two GLSL functions to query a voxel's binary VSV visibility. Input* `voxels` *is a texel with specific* (α, θ) *from the epipolar texture.* `phiBit` *contains* $\phi \in [0..127]$.

```
// Fast version.  Compiles to 56 ASM instructions
float litPercent( uvec4 vox0, uvec4 vox1,
                  int ϕBit, float wθ, float wϕ )
{
  // Data needed to select correct phiBits
  uint comp0 = ϕBit>>5,     bit0 = 1<<(ϕBit&31);
  uint comp1 = (ϕBit+1)>>5, bit1 = 1<<((ϕBit+1)&31);

  // Interpolation weights for 4 voxels
  float wθϕ = wθ * wϕ;
  vec4 weights = vec4(1.0 - wϕ - (wθ - wθϕ),
                    wθ - wθϕ, wϕ - wθϕ, wθϕ);

  // Select 2 bits from each of the input texels
  uvec4 vox, tmp;
  tmp = comp0>1 ? uvec4(vox0.zw,vox1.zw)
                : uvec4(vox0.xy,vox1.xy);
  vox.xy = ((comp0&1)==0?tmp.xz:tmp.yw) & uvec2(bit0);
  tmp = comp1>1 ? uvec4(vox0.zw,vox1.zw)
                : uvec4(vox0.xy,vox1.xy);
  vox.zw = ((comp1&1)==0?tmp.xz:tmp.yw) & uvec2(bit1);

  // Identify set bits (store as true in bvec4)
  bvec4 voxSet = greaterThan( vox, uvec4(0) );

  // Multiply w/weights (mix), sum all 4 (dot)
  return dot( mix(coefs, vec4(0), voxSet), vec4(1));
}
```

Listing 2: *Fast VSV interpolation for surface shadows. Inputs: two voxels with adjacent* θ *values,* $\phi \in$[0..127]*, and two interpolation weights along the* θ *and* ϕ *axes.*