Real-Time Volumetric Shadows using 1D Min-Max Mipmaps

Jiawen Chen[1] Ilya Baran[2] Frédo Durand[1] Wojciech Jarosz[2]

[1]MIT CSAIL [2]Disney Research Zürich

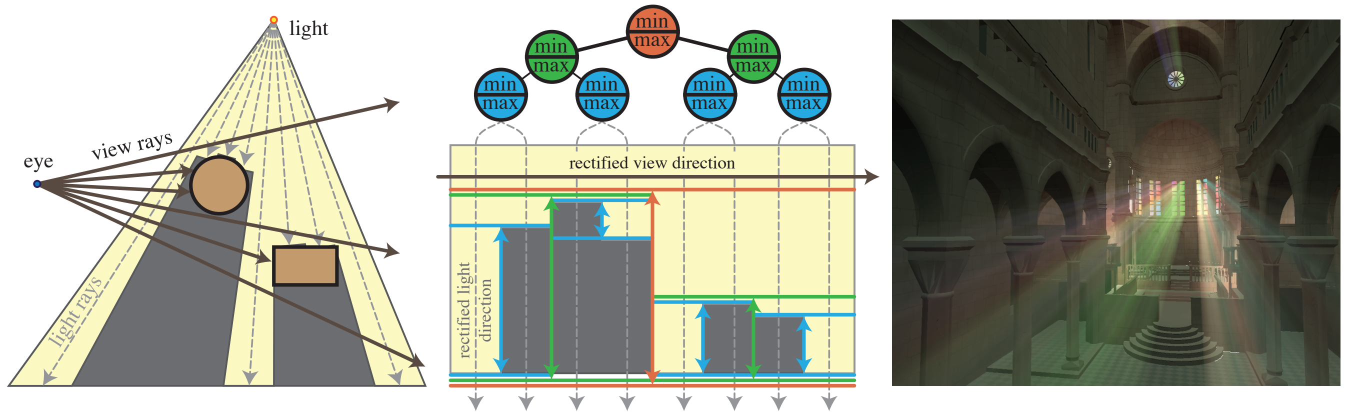


Figure 1: *To compute single scattering in scenes with occluders (left) we compute a depth image from the camera, and a shadow map from the light. After epipolar rectification, each row of the shadow map is a 1D heightfield. We optimize the computation of the scattering integral by using an efficient data structure (a 1D min-max mipmap, center) over this heightfield. This data structure helps compute the scattering integral for all camera rays in parallel. Our method can render complex high-quality scenes with textured lights (right) in real-time (55 FPS).*

Abstract

Light scattering in a participating medium is responsible for several important effects we see in the natural world. In the presence of occluders, computing single scattering requires integrating the illumination scattered towards the eye along the camera ray, modulated by the visibility towards the light at each point. Unfortunately, incorporating volumetric shadows into this integral, while maintaining real-time performance, remains challenging.

In this paper we present a new real-time algorithm for computing volumetric shadows in single-scattering media on the GPU. This computation requires evaluating the scattering integral over the intersections of camera rays with the shadow map, expressed as a 2D height field. We observe that by applying epipolar rectification to the shadow map, each camera ray only travels through a single row of the shadow map (an epipolar slice), which allows us to find the visible segments by considering only 1D height fields. At the core of our algorithm is the use of an acceleration structure (a 1D min-max mipmap) which allows us to quickly find the lit segments for all pixels in an epipolar slice in parallel. The simplicity of this data structure and its traversal allows for efficient implementation using only pixel shaders on the GPU.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shadowing

Keywords: volumetric scattering, global illumination

1 Introduction

In real world scenes, moisture or dust in the air often results in visible volumetric shadows and beams of light (Figure 1, right) known as “god rays” or “crepuscular rays.” Rendering these light scattering effects is often essential for producing compelling virtual scenes. Simulating all scattering events is prohibitively expensive, especially for real-time applications, and approximations are used instead. The “single-scattering” model [Blinn 1982] greatly simplifies rendering, while still producing realistic effects. In this model, a light ray travels from a source, and may get scattered into the eye at any point in the participating medium, while attenuating along the path. This simplification has allowed participating media with shadows to make its way into modern computer games. Like a lot of previous work, we further assume that the scattering medium is homogeneously distributed through the scene. This assumption could be relaxed somewhat, as discussed in Section 4.3.

The simplest method for rendering single scattering is ray marching. A ray is cast from the camera eye through each pixel and the scattering integral is approximated by marching along that ray and checking if each sample is lit or shadowed using a shadow map. To generate high-quality images, however, many samples are needed and several methods for accelerating this process have been published in the last couple of years. Our key insight is that we can apply a simple acceleration structure to intersect camera rays with the shadow map, which we treat as a height field. Furthermore, we use epipolar rectification of the shadow map to transform the 2D height field into a collection of independent 1D height fields, one for each epipolar slice. We also use singular value decomposition to approximate the smoothly varying terms of the scattering integral, avoiding an analytical solution and allowing us to support textured lights. Our main contribution is using a simple but fast acceleration structure (a 1D min-max mipmap, Figure 1, middle) to find the lit segments for all pixels in the image *in parallel*. This procedure is similar to the work by Baran et al. [2010]; however, our actual scattering integral computation, which is the key step of the algorithm, is both much simpler to implement and faster on a GPU.

2 Related Work

There have been several papers written on solving the single-scattering integral (semi)-analytically [Sun et al. 2005; Pegoraro and Parker 2009; Pegoraro et al. 2010], but they necessarily ignore shadowing, which is often required for realism and is the effect on which we concentrate. Other methods, such as volumetric photon mapping [Jensen and Christensen 1998; Jarosz et al. 2008] and line space gathering [Sun et al. 2010] compute solutions to more difficult scattering problems, such as volumetric caustics or multiple scattering, but even with GPU acceleration they are far from real-time on complex scenes.

Max [1986] described how to compute the single-scattering integral by finding the lit segments on each ray using shadow volumes intersected with epipolar slices. The integral on each lit segment is computed analytically. Epipolar sampling [Engelhardt and Dachsbacher 2010] speeds up ray marching by computing it only at depth discontinuities along image-space epipolar lines. The scattered radiance at all other points is interpolated along these lines, but this can cause temporally-varying artifacts, as discussed in prior work [Baran et al. 2010]. Wyman and Ramsey [2008] use shadow volumes to cull ray marching in unlit areas. Hu et al. [2010] recently presented an algorithm for interactive volumetric caustics, using Wyman and Ramsey's [2008] method for single scattering. Though this method works well for simple occluders, it becomes slow in the presence of complex visibility boundaries. Several methods [Dobashi et al. 2000; Dobashi et al. 2002] compute the scattering integral by constructing slices at different depths, rendering the scattering at these slices, and using alpha-blending to combine them. Imagire and colleagues [2007] use a hybrid approach that incorporates both slices and ray marching.

The algorithm of Billeter et al. [2010] is similar to that of Max [1986], but generates the shadow volume from the shadow map and uses the GPU rasterizer to compute the lit segments. This latter method is exact up to the shadow map resolution. It is very fast for low-resolution shadow maps, but slows down significantly as the shadow map resolution approaches 4096^2 and the number of vertices in the shadow volume overwhelms the pipeline. Unfortunately, the large and complex scenes we target require such high-resolution shadow maps to avoid aliasing. For high-resolution shadow maps, Billeter et al. perform online decimation of the shadow volume, but this is expensive and we did not observe a significant performance benefit from this in our experiments. Like Billeter et al., we also compute the lit segments, but while their outer loop is over the shadow volume elements, our outer loop is over pixels. This precludes the need for sending a large number of vertices to the GPU and allows us to handle high-resolution shadow maps with ease. Additionally, unlike their method, we support textured lights, and our method can be combined with epipolar sampling [Engelhardt and Dachsbacher 2010] for further acceleration, although we have not tested this.

The algorithm of Baran et al. [2010], (which we call "incremental integration"), uses epipolar rectification to reduce scattering integration to partial sums on a rectilinear grid and uses a partial sum tree to accelerate this computation. While this method has good worst-case upper bounds and is very fast on the CPU, it requires an incremental traversal of multiple camera rays in a particular order, making it difficult to utilize the full parallelism provided by a GPU. Implementing their method on a GPU also requires using a GPGPU API, such as CUDA or OpenCL. Unfortunately, as of this writing, these are not supported on all graphics cards, such as those found in current game consoles. In contrast, our method sacrifices worst-case guarantees, but allows all camera rays to be processed in parallel using only features found in DirectX 9 pixel shaders. We also do not require camera rectification, avoiding the need to process twice as many camera rays as pixels (due to the non-uniform sampling in polar coordinates) and reducing aliasing. Overall, we achieve a significant speedup at slightly better quality, even on high-complexity scenes, like a forest, which is a worst-case scenario for our algorithm.

Min-max mipmaps have previously been used for accelerating soft shadows [Guennebaud et al. 2006], global illumination [Nichols and Wyman 2009], as well as ray tracing geometry images [Carr et al. 2006] and height fields [Mastin et al. 1987; Musgrave et al. 1989; Tevs et al. 2008]. In our case, thanks to rectification, we only need 1D min-max mipmaps, whose traversal is simple and efficient. Also, unlike the other raytracing applications, we need all intersections between the height field and the ray, not only the first.

3 Method

Our technique uses the following high-level per-frame procedure:

1. Render a depth map from the camera and a shadow map from the light.
2. Perform epipolar rectification on the shadow map (Section 3.2).
3. Compute a low-rank approximation to all but the visibility terms in the scattering integral (Section 3.2).
4. For each row of the rectified shadow map, compute a 1D min-max mipmap (Section 3.3).
5. For each camera ray, traverse the min-max mipmap to find lit segments and accumulate the scattering integral.
6. If the epipole is on or near the screen, compute the scattering near the epipole using brute force ray marching.

This pipeline is similar to that of incremental integration [Baran et al. 2010]; our technical contribution is to eliminate the dynamic data structure (partial sum tree) used for integration, replacing it with a static min-max mipmap, which is simpler, allows all rays to be processed in parallel, and avoids the need for camera rectification.

In the rest of this section we define the problem, briefly describe the common elements shared with incremental integration, and describe in detail how we incorporate min-max mipmaps for integration.

3.1 Single Scattering Formulation

In the single-scattering model, the radiance scattered toward the eye is integrated along each camera ray, up to the first surface. At every point on this ray, if the light source is visible from that point, a certain fraction of that light gets scattered towards the eye. Light also gets attenuated as it travels from the light source to the scattering point and to the eye. This leads to the following equation for the radiance L scattered towards the eye (assumed to be the origin) over a camera ray whose direction is $\mathbf{v}$:

$$L(\mathbf{v}) = \int_0^d e^{-\sigma_t s}\, V(s\mathbf{v})\, \sigma_s\, \rho(\theta)\, L_{\text{in}}(s\mathbf{v})\, \mathrm{d}s, \tag{1}$$

where d is the distance to the first occluder along the ray, σ_s is the scattering coefficient, $\sigma_t = \sigma_s + \sigma_a$ is the extinction coefficient, $V(s\mathbf{v})$ is 1 if the point $s\mathbf{v}$ can see the light and 0 if it is in shadow, $L_{\text{in}}(s\mathbf{v})$ is the radiance incident to point $s\mathbf{v}$ assuming no occlusion, and $\rho(\theta)$ is the scattering phase function with θ being the angle between $\mathbf{v}$ and the light direction. For the simplest case of a uniform directional light source with no extinction along light rays, $L_{\text{in}}(s\mathbf{v})$ is a constant. For an isotropic point light source at $\mathbf{x}$,

$$L_{\text{in}}(s\mathbf{v}) = \frac{I e^{-\sigma_t d(s\mathbf{v})}}{d(s\mathbf{v})^2}, \quad d(s\mathbf{v}) = \|\mathbf{x} - s\mathbf{v}\|, \tag{2}$$

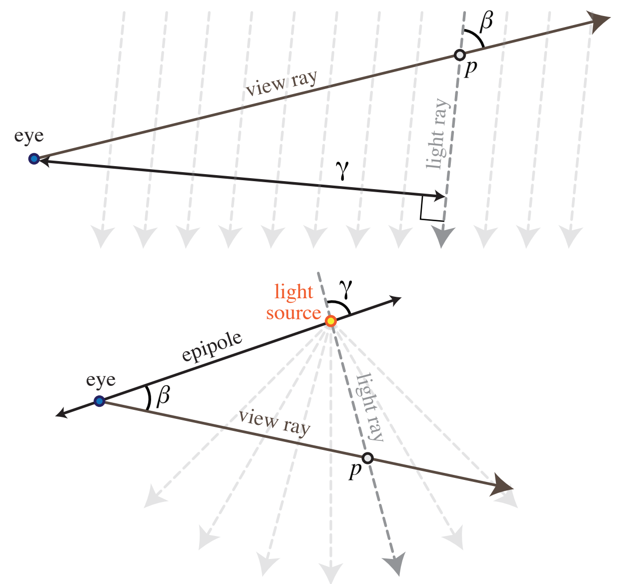


Figure 2: *Epipolar coordinates within an epipolar slice for a directional light (top) and a point light (bottom). The α coordinate determines the slice.*

where I is the light intensity. For a textured light source, the intensity I is a function of $s\mathbf{v}$ projected to light coordinates.

3.2 Rectification and Low-Rank Approximation

Epipolar rectification is an assignment of coordinates (α, β, γ) to every world space point p such that camera rays are indexed by (α, β) and light rays are indexed by (α, γ). The world space is partitioned into epipolar slices, planes that contain the eye and that are parallel to the light direction, and the α coordinate specifies p's slice. The coordinate β specifies the view ray within the slice. For directional lights this is the angle to the light direction, and for point lights this is the angle between the view ray and the direction from the eye to the light source. The γ coordinate specifies the light ray within the slice, measured as the distance to the eye for a directional light source or the angle to the eye for a point light source. We illustrate this in Figure 2. Each pixel of the shadow map corresponds to a point in world space. In the rectified shadow map (Figure 3), rows are indexed by α, columns by γ, and the element stored at (α, γ) is the β coordinate of the camera ray at which that light ray terminates. Because we do not need to process the camera rays in any particular order, we do not rectify the camera depth map.

Using a change of variables, the scattering integral (1) of an untextured light may be written in epipolar coordinates as:

$$L(\alpha,\beta) = \int_0^{D(\alpha,\beta)} e^{-\sigma_t s(\beta,\gamma)} V(\alpha,\beta,\gamma)\, \sigma_s\, \rho(\beta,\gamma)\, L_{\text{in}}(\beta,\gamma)\, \frac{\mathrm{d}s}{\mathrm{d}\gamma} \mathrm{d}\gamma, \quad (3)$$

where $D(\alpha, \beta)$ is the γ coordinate of the light ray at which the camera ray (α, β) is blocked. Except for the visibility component, the integrand only depends on β and γ, not α. Thus, we may write

$$L(\alpha,\beta) = \int_0^{D(\alpha,\beta)} V(\alpha,\beta,\gamma)\, \mathcal{I}(\beta,\gamma) \mathrm{d}\gamma, \quad (4)$$

where $\mathcal{I}$ has all of the other terms baked in. We would like to precompute $\int \mathcal{I}(\beta,\gamma)\mathrm{d}\gamma$, but that would require approximating the integral once for each β, which would be expensive. Instead, we note that all terms of $\mathcal{I}$ vary smoothly. We can therefore approximate $\mathcal{I}(\beta,\gamma)$ as $\sum_i^N \mathrm{B}_i(\beta)\Gamma_i(\gamma)$ for a small N. We compute this approximation by sampling $\mathcal{I}$ in a 64-by-64 grid, taking the SVD (on the CPU, as this needs to be done only once per frame), and using the singular vectors associated with the top N singular values (we use $N = 4$). We then obtain

$$L(\alpha,\beta) \approx \sum_i^N \left(\mathrm{B}_i(\beta) \int_0^{D(\alpha,\beta)} V(\alpha,\beta,\gamma)\Gamma_i(\gamma)\mathrm{d}\gamma \right). \quad (5)$$

Approximating the integral as a Riemann sum, and using the structure of the visibility function we obtain

$$L(\alpha,\beta) \approx \sum_i^N \left(\mathrm{B}_i(\beta) \sum_{\substack{\gamma < D(\alpha,\beta) \\ S[\alpha,\gamma] > \beta}} \Gamma_i(\gamma)\Delta\gamma \right), \quad (6)$$

where $S[\alpha, \gamma]$ is the rectified shadow map. This allows us to compute only N prefix sum tables, instead of one for each β.

3.3 Min-Max Mipmap Construction and Traversal

Up to this point, our algorithm is the same as incremental integration, except that we do not need to rectify the camera depth map. The difference is in how the actual integral, i.e., the inner sum of Equation (6) is evaluated. Incremental integration uses interdependence between camera rays within a slice, maintaining $\sum_\gamma V(\beta,\gamma)\Gamma_i(\gamma)\Delta\gamma$ in a partial sum tree for each slice. In contrast, we process each ray independently, which enables massive parallelism. For each ray, we find the segments (γ_-, γ_+) for which $S[\alpha,\gamma] > \beta$—in other words, the lit regions of that ray. We then use a table of prefix sums of Γ_i to compute the integral over the segments. Our speed is due to using a 1D min-max mipmap to accelerate finding the lit segments.

Each row of the rectified shadow map S represents a 1D heightfield in an epipolar slice and we want to quickly find the intersection of that heightfield with a camera ray (Figure 4, middle). We compute a complete binary tree on each row with every node of this tree storing the minimum and maximum values of S below that node (Figure 4, top). To compute these trees, we use $\log_2 d$ ping-pong passes each of which computes a level from the level below (d is the number of light rays—the resolution of γ). The two buffers are then coalesced into a single one with each row laid out level by level (Ahnentafel indexing), starting with the root at index 1.

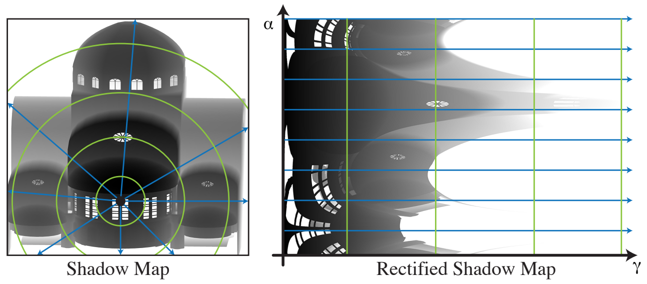


Figure 3: *We perform rectification on the shadow map (left). This turns epipolar slices (radial lines emanating from the epipole, left) into rows of our rectified shadow map (right). For this directional light, the radial distance from the epipole is the coordinate γ which specifies a column within each slice. The value of each pixel in the rectified shadow map gives the β coordinate of the occluder.*

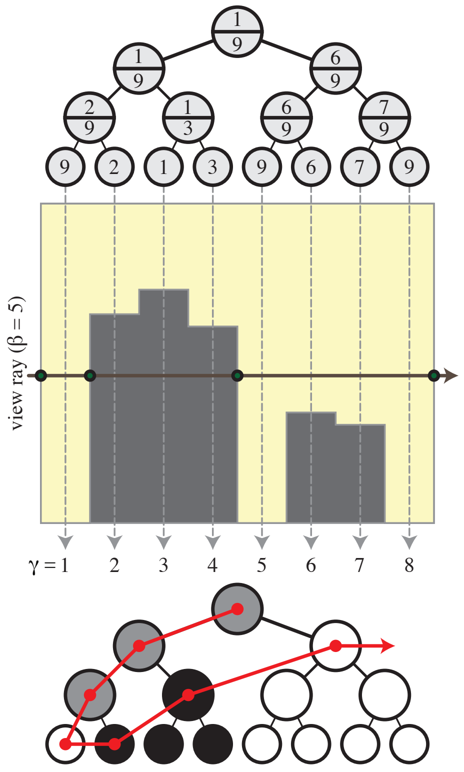


Figure 4: *The min-max mipmap (top) corresponding to a row of the rectified shadow map in the scene from Figure 1. To compute the scattering integral for any view ray in this epipolar slice (middle), we traverse the tree by thresholding the min-max mipmap with the β value of the view ray (bottom). Nodes entirely in shadow are colored black, nodes entirely lit are colored white, and nodes containing visibility boundaries are grey.*

After the tree is constructed, for each camera ray we traverse it using a recursive algorithm (Figure 4, bottom), which we efficiently implement in a pixel shader without recursion (see pseudocode in Figure 5). We have a camera ray at "height" β and we start from the root of the tree. When we process a tree node, if β is less than the minimum, that node is completely lit and we add the integral over the range of that node to the output. If β is greater than the maximum, the node is completely shadowed and we skip it. Otherwise, we recursively process the two child nodes. To avoid the recursion, when we are done with a node, instead of popping the call stack, we move to the next node, which is either the node at the next index location or its ancestor that is a right child.

3.4 Textured Lights

To simulate an effect like stained-glass windows, it is useful to treat the light as having a texture. Similarly to the shadow map, we rectify the light texture map, to obtain $T(\alpha, \gamma)$, which gets multiplied into the integrand. Because this function may not be smoothly varying, we do not bake it into $\mathcal{I}$, leaving it as a separate term. Equation (6) becomes:

$$L(\alpha,\gamma) \approx \sum_{i}^{N} \left(\mathrm{B}_i(\beta) \sum_{\substack{\gamma < D(\alpha,\beta) \\ S[\alpha,\gamma] > \beta}} \Gamma_i(\gamma) T(\alpha,\gamma) \Delta\gamma \right) . \quad (7)$$

```
proc INTEGRATE-RAY($x$, $y$)
 1 float $depth$ = $D[x, y]$
 2 float $(\alpha, \beta, \gamma_1)$ = TO-EPIPOLAR$(x, y, depth)$
 3 int $node$ = 1   // start at the root
 4 int2 $(\gamma_-, \gamma_+)$ = RANGE$(node)$
 5 float4 $out$ = (0,0,0,0)
 6 do
 7   if $min[node, \alpha] \le \beta < max[node, \alpha]$
 8     $node$ = $node * 2$   // recurse to left child
 9   else   // fully lit or fully shadowed
10     if $\beta < min[node, \alpha]$   // fully lit node
11       $out = out + \Gamma_{\mathrm{sum}}[\min(\gamma_1, \gamma_+)] - \Gamma_{\mathrm{sum}}[\gamma_-]$
12     end if
13     $node$ = $node + 1$   // advance to next node
14     while $node$ is even   // while node is a left-child
15       $node$ = $node / 2$   // go to parent
16     end while
17   end if
18   $(\gamma_-, \gamma_+)$ = RANGE$(node)$
19 while $node \ne 1$ && $\gamma_1 > \gamma_-$   // while ray extends into node
20 return dot(B[$\beta$], $out$)
```

Figure 5: *Pseudocode for integration using a min-max mipmap. By the time this function is called, $min[node, \alpha]$ and $max[node, \alpha]$ contain the min-max mipmap, and Γ_{sum} stores the prefix-sums of Γ. The function* RANGE *returns the range of γ coordinates that are below the given tree node.*

Instead of precomputing prefix sums of $\Gamma_i(\gamma)$, we need to precompute prefix sums of $\Gamma_i(\gamma)T(\alpha, \gamma)$, which is a lot more work because it needs to be done per α. We do this using $O(\log_2 t)$ ping-pong passes, where t is the γ-resolution of the rectified light texture.

3.5 Implementation Details

Our implementation includes minor optimizations over the method presented above. To avoid a potentially expensive while loop in the shader for going up the tree, we only go up at most one level, replacing the **while** on line 14 with an **if**. This optimization has also been used for heightfield rendering [Tevs et al. 2008].

Our method often computes the integral over a single lit segment of a ray using multiple tree nodes. To avoid adding and subtracting Γ_{sum} terms that just cancel out, we delay the update to the integral (line 11 in the pseudo-code) until the next unlit tree node we encounter or the end of the traversal. This optimization is especially useful for colored textured light sources because we need to access 24 floats (three color channels by four singular vectors for the start and end) instead of eight.

In our textured light source tests, we use a light texture whose resolution is lower than the shadow map (512×512 in our examples). To achieve good performance, we do both the rectification and the prefix sum computation at the lower resolution and access the resulting texture (line 11) using hardware linear interpolation. The error this introduces is not noticeable.

4 Results and Evaluation

We implemented our method in DirectX 11 although we do not use features beyond those in DirectX 9. All our tests were done on an Intel Core i7 960 (3.2 GHz) with an NVIDIA GeForce 480 GTX GPU at 1280×960 resolution. We use four scenes for our tests: a church scene (SIBENIK), a church with a textured light (SIBENIKTEX), a forest scene (TREES), and an open mountain scene (TERRAIN). The number of epipolar slices used in our

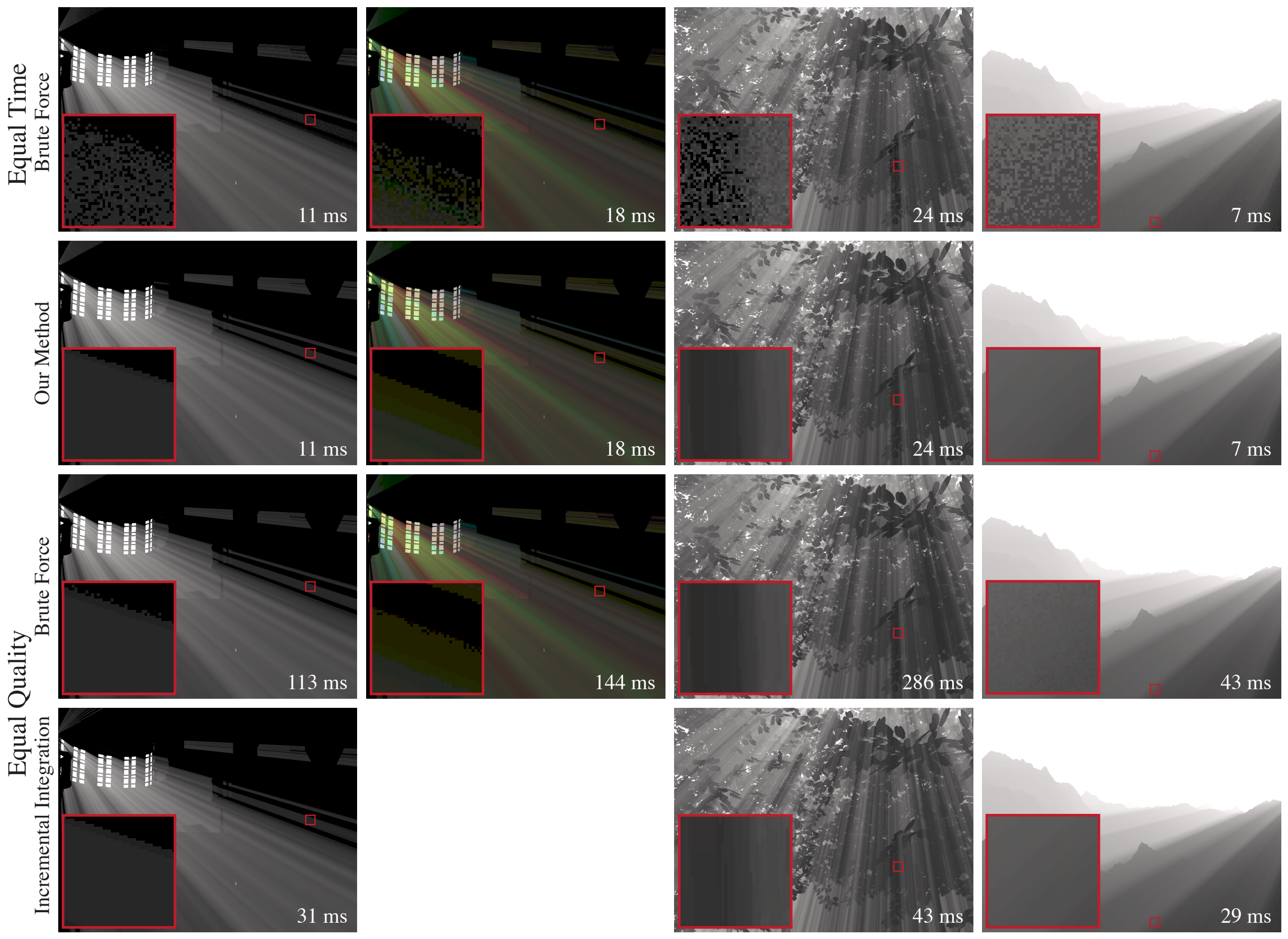


Figure 6: *A comparison of our method to brute force ray marching and incremental integration. The GPU implementation of incremental integration does not support colored textured light sources and is therefore omitted. The reported times are for the scattering computation only, excluding the time to render the shadow map, the depth map, and to compute direct lighting.*

algorithm and incremental integration is scene-dependent, sufficient to guarantee that each camera ray is within half a pixel of a slice [Baran et al. 2010]. We used an isotropic phase function $\rho = 1/4\pi$. In timing the scattering computation, we do not include the time necessary to render the camera depth map or the unrectified shadow map, as these operations are part of a standard rendering loop.

4.1 Performance

Table 1 shows how long various stages of the algorithm take for our scenes with a 4K×4K shadow map. Figure 6 shows a comparison against brute force ray marching at equal time and equal quality. Markus Billeter and his colleagues kindly provided us with an implementation of their method [2010], which we slightly sped up (roughly by 10%) by using a triangle strip instead of individual triangles to render the light volume. However, this implementation does not support directional lights and to compare with their method, we used a far-away spotlight in our scenes, as shown in Figure 7. Table 2 shows our performance and that of Billeter et al. as the shadow map resolution varies from 1K to 4K. Our method is faster (except at 1K shadow-map resolution on TERRAIN) and scales better to high-resolution shadow maps. Note that for complex scenes, even a 2K shadow map resolution is insufficient and leads to temporal aliasing, as shown in our video. Our video also

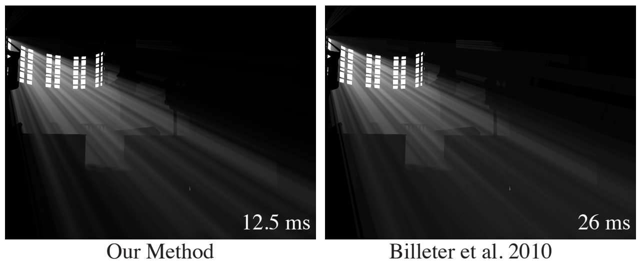


Figure 7: *Single scattering in* SIBENIK *using a shadow map resolution of* 4096×4096 *rendered using our method (left) and the method from Billeter et al. [2010].*

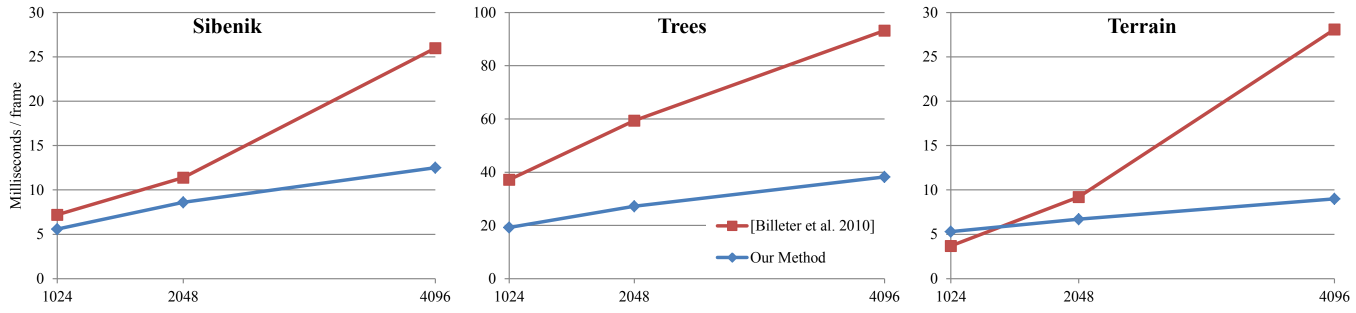
has a comparison to epipolar sampling [Engelhardt and Dachsbacher 2010], demonstrating that our method is significantly faster for similar quality.

4.2 Complexity Comparison

It is instructive to understand the number of operations our method performs, compared to recent work. Let s be the number of slices, p be the number of pixels, and d be the number of depth samples (the resolution of the γ variable, equal to the shadow map resolution in our experiments). Let l be the number of contiguous lit

Scene	Shadow Map & Direct Lighting	Shadow Map Rectification	Light Texture Precomputation	Mipmap Construction	Integration	Brute Force near Epipole	**Total Scattering (Our Method)**	Total Scattering [Baran et al. 2010]	Total Scattering (Brute Force)
		Our Method							
SIBENIK	3.6	1.3	-	2.5	6.8	0.3	**11**	31 (2.8×)	113 (10×)
SIBENIKTEX	3.7	1.4	1.2	2.5	12.2	0.3	**18**	-	144 (8×)
TREES	7.3	1.3	-	2.6	19.8	-	**24**	43 (1.8×)	286 (12×)
TERRAIN	11.2	1.4	-	2.5	2.6	-	**7**	29 (4.1×)	43 (6×)

Table 1: *This table shows the breakdown of the timing of our method among various stages for a single frame, as well as the timing of incremental integration and brute force ray marching at equal quality. The total times are for scattering only and do not include the time for rendering the shadow map, the camera depth map, and direct lighting, as these operations are part of the standard pipeline and necessary for all of the methods. When comparing to incremental integration, we also omit its (roughly 10 ms) CUDA/Direct3D interop overhead in the numbers above. The shadow map resolution is* 4096×4096. *All times are in milliseconds.*



	SIBENIK			TREES			TERRAIN		
	1K	2K	4K	1K	2K	4K	1K	2K	4K
Our method	5.6	8.6	12.5	19.2	27.2	38.2	5.3	6.7	9.0
[Billeter et al. 2010]	7.2	11.4	26.0	37.2	59.4	93.2	3.7	9.2	28.1
Our speedup	1.3×	1.3×	2.1×	1.9×	2.2×	2.4×	0.7×	1.4×	3.1×

Table 2: *Timings for computing scattering using our method and the method of Billeter et al. [2010] on versions of our scenes with a spotlight instead of a directional light. The timings for each scene are for three shadow map resolutions and are given in milliseconds. For the terrain scene at 4K resolution, we used the adaptive version of Billeter et al.'s method because it was* 1.8*ms faster than the non-adaptive version.*

segments—so for a scene with no objects, $l = p$, and for a scene like TREES, $l \gg p$. Let l' be the number of contiguous lit segments up to the first blocker (i.e., whose integral is actually relevant)—so $p \leq l' \leq l$. Our method performs $O(l' \log d)$ operations, of which precisely $2l'$ are accesses to Γ. The method of Billeter et al. [2010] performs $\Theta(l)$ operations, of which $2l$ are computations of the analytic scattering model. They avoid the $\log d$ tree overhead, but have to process segments that do not contribute to the scattering integral to avoid z-fighting when rendering the light volume. Having to process invisible segments and large numbers of polygons for high-resolution shadow maps offsets their advantage from not having to use trees to find light-shadow transitions. For incremental integration, the complexity is $\Theta((p+sd)\log d)$, but while $p+sd$ is almost always much smaller than l', that algorithm has a large working set and relatively poor parallelism. This makes incremental integration slower on the GPU even on scenes with very large l', like TREES.

4.3 Discussion and Limitations

Like several other methods for speeding up volumetric shadows, we assume a homogeneous isotropic medium. The isotropy assumption allows us to compute the SVD only once per frame: for an anisotropic medium, $\mathcal{I}$ becomes a function of α. It is a smooth function of α, so it may be possible to compute the SVD at a few values of α and interpolate between the results, but we have not tested this. A similar method may work for a very smoothly varying nonhomogeneous medium. Note that an anisotropic medium is not the same thing as an anisotropic phase function, which is only a function of β and γ and which our method supports (although we have not verified this experimentally).

Reducing aliasing, both spatial and temporal, is a challenge for all methods based on shadow maps. Aliasing in the shadow map leads to aliasing in the scattering integral. If in the next frame, a polygon edge is rasterized differently, that may cause a sudden jump in the calculated inscatter that manifests itself as a temporal artifact. Our video demonstrates this problem. Using a high-resolution shadow map is necessary to keep the aliasing to a minimum. In addition, our method introduces a little extra aliasing when it rectifies the shadow map and when a camera ray is "quantized" to a specific epipolar slice. This aliasing is relatively minor, but can be seen on close examination. The aliasing in incremental integration is strictly greater than in our method: additional aliasing is introduced by the camera rectification and unrectification. Incremental integration also supports reducing the rectified shadow map resolution and antialiasing in the spirit of deep shadow maps [Lokovic and Veach 2000]. Because our method scales much better to higher resolution shadow maps, we have not had to do this.

Unlike in the method of Billeter et al., our method's rectified shadow map depends on the view, and not only on the scene. Their method is therefore much better suited than ours for a static scene with a fixed light: it will produce no temporal aliasing because the shadow map is fixed and the number of triangles in the shadow volume can be greatly reduced as a precomputation. Using the SVD and precomputing texture prefix sums can also replace the analytic formula in Billeter et al.'s method to enable it to support textured lights. We have not explored the effect this would have on their performance. Conversely, for an untextured light, instead of using an SVD, we could use a (semi-)analytical model in our method to compute the integrals over lit segments of the camera rays.

5 Conclusions

We have presented a real-time algorithm for rendering volumetric shadows in single-scattering media. Overall, we achieve a significant speedup or better quality compared to the state of the art. Our key insight was to apply a simple acceleration structure to intersect camera rays with the shadow map, which we treated as a height field. We furthermore used epipolar rectification of the shadow map to reduce the problem of intersecting with a 2D height field to intersecting with 1D height fields. Our main contribution was using a 1D min-max mipmap to find the lit segments for all pixels in the image. This data structure sacrifices the worst-case guarantees provided by Baran et al. [2010], but its simplicity allows us to better exploit the parallel processing capabilities of the GPU by processing all camera rays simultaneously. We showed that our technique scales well to large shadow map resolutions and, due to good use of the GPU, even to scenarios with highly complex visibility functions. The resulting algorithm is simple to implement and only requires features available on current game consoles.

Acknowledgements

We would like to thank Markus Billeter for providing us with their code to use for comparisons. We would also like to thank Thomas Engelhardt and Carsten Dachsbacher for providing us with their prototype. Thanks to Derek Nowrouzezahrai for helpful feedback. Jaakko Lehtinen contributed support code and made TERRAIN and TREES. SIBENIK was made by Marko Dabrovic.

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*, 5. to appear.

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

BLINN, J. F. 1982. Light reflection functions for simulation of clouds and dusty surfaces. 21–29.

CARR, N. A., HOBEROCK, J., CRANE, K., AND HART, J. C. 2006. Fast gpu ray tracing of dynamic meshes using geometry images. In *Graphics Interface 2006*, 203–209.

DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2000. Interactive rendering method for displaying shafts of light. In *Proc. Pacific Graphics*, IEEE Computer Society, Washington, DC, USA, 31.

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

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

GUENNEBAUD, G., BARTHE, L., AND PAULIN, M. 2006. Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering (EGSR), Nicosia, Cyprus, 26/06/2006-28/06/2006*, Eurographics, http://www.eg.org/, 227–234.

HU, W., DONG, Z., IHRKE, I., GROSCH, T., YUAN, G., AND SEIDEL, H.-P. 2010. Interactive volume caustics in single-scattering media. In *I3D '10: Proceedings of the 2010 symposium on Interactive 3D graphics and games*, ACM, 109–117.

IMAGIRE, T., JOHAN, H., TAMURA, N., AND NISHITA, T. 2007. Anti-aliased and real-time rendering of scenes with light scattering effects. *The Visual Computer 23*, 9–11 (Sept.), 935–944.

JAROSZ, W., ZWICKER, M., AND JENSEN, H. W. 2008. The beam radiance estimate for volumetric photon mapping. *Computer Graphics Forum 27*, 2 (Apr.), 557–566.

JENSEN, H. W., AND CHRISTENSEN, P. H. 1998. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proc. SIGGRAPH 98*, 311–320.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 385–392.

MASTIN, G. A., WATTERBERG, P. A., AND MAREDA, J. F. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics & Applications 7*, 3 (Mar.), 16–23.

MAX, N. L. 1986. Atmospheric illumination and shadows. In *Computer Graphics (Proc. SIGGRAPH '86)*, ACM, New York, NY, USA, 117–124.

MUSGRAVE, F. K., KOLB, C. E., AND MACE, R. S. 1989. The synthesis and rendering of eroded fractal terrains. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, 41–50.

NICHOLS, G., AND WYMAN, C. 2009. Multiresolution splatting for indirect illumination. In *ACM Symposium on Interactive 3D Graphics and Games*, ACM, 83–90.

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

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.

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.

SUN, X., ZHOU, K., LIN, S., AND GUO, B. 2010. Line space gathering for single scattering in large scenes. *ACM Trans. Graph. 29*, 4.

TEVS, A., IHRKE, I., AND SEIDEL, H.-P. 2008. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *Symposium on Interactive 3D Graphics and Games (i3D'08)*, 183–190.

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