High Performance Graphics (2010)
M. Doggett, S. Laine, and W. Hunt (Editors)

Real Time Volumetric Shadows using Polygonal Light Volumes

Markus Billeter[1], Erik Sintorn[1] and Ulf Assarsson[1]

[1]Chalmers University of Technology, Gothenburg, Sweden

Abstract

This paper presents a more efficient way of computing single scattering effects in homogeneous participating media for real-time purposes than the currently popular ray-marching based algorithms. These effects include halos around light sources, volumetric shadows and crepuscular rays. By displacing the vertices of a base mesh with the depths from a standard shadow map, we construct a polygonal mesh that encloses the volume of space that is directly illuminated by a light source. Using this volume we can calculate the airlight contribution for each pixel by considering only points along the eye-ray where shadow-transitions occur. Unlike previous ray-marching methods, our method calculates the exact airlight contribution, with respect to the shadow map resolution, at real time frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing and texture

1. Introduction

Light scattering in air and other participating media is a key component to generate realistic images for many real-world scenes. Aside from providing a depth cue to the viewer, many commonly seen phenomena such as glows around light sources, volumetric shadows and crepuscular rays (also known as “god-rays”) are obtained correctly only by considering how light scatters and is absorbed in the surrounding media, before reaching the eye.

Simulating true multiple scattering of light in a participating medium can be done with e.g. path tracing or photon mapping, but for real-time rendering, neither method can handle scenes of any complexity. A simpler model, which only considers single scattering in a homogeneous participating medium, is often used, and for optically thin media (like, for instance, air) this is still a very good approximation [SRNN05]. In the single scattering model, we consider only how the light leaving a point in space is attenuated due to absorption or out scattering along the eye-ray. At the same time, intensity increases due to photons directly emitted from the light source that scatter towards the eye along the same ray. These effects can be formulated as a comparatively simple integral [NMN87]. While the integral has no simple analytic solution, several methods suitable for real-time rendering have been suggested that approximate it well [SRNN05, PP09].

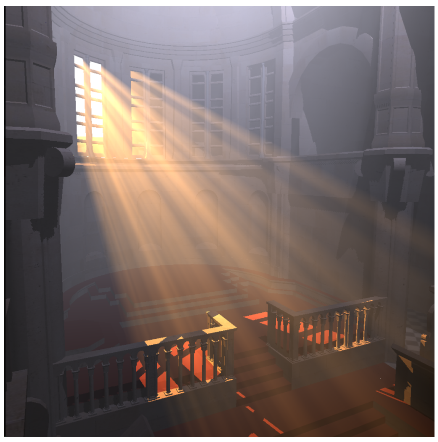


Figure 1: *The Sibenik cathedral lit by a strong yellow light from outside the window and a large white light from above. Both light sources cast volumetric shadows constructed from* 1024×1024 *shadow maps. The image is also rendered at* 1024×1024 *and runs at* ~ 40 *FPS.*

The single scattering model captures convincingly the effects of fog and halos around light sources. Light that is scattered towards the observer, making the participating medium visible, is often referred to as *airlight*. Parts of the participating medium may be occluded along the eye ray, which must be taken into account to capture volumetric shadowing effects. In Section 3, we present a method to correctly calculate the airlight contribution, given unordered locations of transitions between lit and unlit regions along an eye-ray. Section 4 deals with finding these transitions efficiently.

The main contribution of this paper is the realization that a shadow map generated for a point light actually defines a volume enclosing the directly illuminated space. A mesh surrounding this space can be constructed and rendered in real-time, which allows calculation of the true amount of airlight for each pixel (limited by the resolution of the shadow map), without the need for ray-marching. This produces better quality images at significantly higher frame rates.

Furthermore, in Section 4.1, we suggest a scheme for adaptively tessellating the enclosing meshes, allowing us to render very high quality volumetric shadows at interactive frame-rates.

2. Previous Work

We will attempt to list the publications most relevant to understanding this paper, but make no attempt to cover the complete huge body of work related to scattering in participating media.

Single scattering in a participating medium was treated by Nishita et. al. [NMN87], who introduced the airlight integral. Our algorithm relies on the ability to solve the integral efficiently, and several papers discuss this problem. Sun et. al. [SRNN05] find a form of the integral that depends only on dimensionless parameters and use this to look up a value in a scene-independent precomputed 2D texture. Pegoraro et. al. [PP09] instead present a simplified form of the integral, which depends on a one-dimensional exponential integral, of which the solution can be approximated using a Taylor series expansion.

In an up-following paper [PSP09], that idea is extended to support anisotropic media and light sources and a GPU based implementation is presented. In another related paper, Zhou et. al. [ZHG*07] handle inhomogeneous media by approximating the density as a sum of Gaussian functions.

Our implementation, presented in this paper, relies on a method very similar to that of Sun et. al. [SRNN05], but any method of computing airlight could potentially be used with our volumetric shadowing scheme.

Shadows for single scattering in participating media have been thoroughly researched as well. One approach is to ray-march through the medium, either by drawing alpha blended planes [DYN02, IJTN07] or by explicitly looping in a fragment shader. Among the latter is the work by Toth and Umenhoffer [TU09], who suggest that a few samples are taken for each pixel and nearby pixels may borrow results from each other. In [ED10], the ray-marching is done instead along the epipolar lines from the light source, and the final airlight contribution is found for each pixel by interpolating between these samples.

Another approach is to use the shadow volumes cast by occluders to find the points along the eye-ray where airlight contributions must be evaluated. One such method is the one presented by Venceslas et. al. [VDS06], where the shadow planes are sorted back to front before rendering. Similarly, James [Jam03] orders the shadow planes through depth peeling.

Wyman et. al. [WR08] present a hybrid of these two approaches. Shadow Volumes are used here to limit the range in which ray-marching is required, significantly improving performance for some scenes. In their Future Works section, a method much like the one we present here, is suggested, but not implemented or evaluated.

Finally, one method that falls into neither category is presented by Mitchell [Mit07]; this method operates entirely in screen space. One limitation of this approach is that the light source must be visible.

The first paper suggesting the extrusion of shadow volumes from shadow maps was [McC00]. That paper observes that such a volume has no nesting or overlapping, which allows generating shadows using a single parity bit unlike traditional shadow volumes where shadow-polygons must be counted in a stencil buffer. This same property is what allows us to dispose of ray-marching as explained in Section 4.

3. Single scattering using Light Volumes

Our goal is to compute the amount of light, from a point source, that is scattered towards the observer in a homogeneous participating medium, taking into account occlusion of the light source along the eye-ray. Figure 2 shows an example scene with a single light source, an observer and two objects.

The view ray marked in Figure 2 passes through two distinct illuminated regions. Only these regions contribute to the total airlight observed along the view ray. Occluded regions will, under the single-scattering approximation, not contribute to the airlight; the medium in those regions receives no direct light that may be scattered.

The airlight contribution from a line segment, d_0 to d_1, along the view ray, is given by the integral [NMN87, SRNN05]

$$L_a(d_0, d_1) = \int_{d_0}^{d_1} \beta\, k(\alpha)\, \frac{I_0\, e^{-\beta d(x)}}{d(x)^2}\, e^{-\beta x}\, dx. \qquad (1)$$

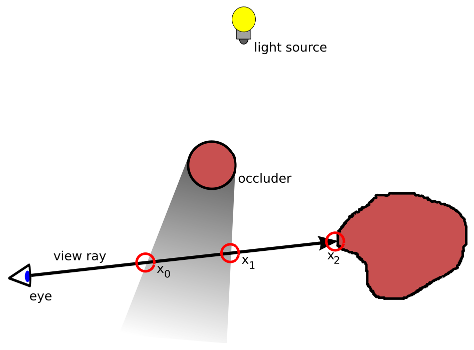


Figure 2: *Example scene with a light source, a viewer and two objects. A single view ray is considered, starting at the eye, $x = 0$, and passing through an unlit region between $x = x_0$ and $x = x_1$, and finally ending at the second object, $x = x_2$. The ray thus passes through two lit regions, each contributing to the airlight independently.*

Using this integral, we can express the airlight contribution in Figure 2 as

$$L_a^{total} = L_a(0, x_0) + L_a(x_1, x_2).$$

More generally, to compute the airlight contribution, we need to evaluate Equation 1 for each lit region along the view ray and sum the results. Finding these lit regions efficiently is tricky. Instead, using additivity of integration on intervals, Equation 1 can be rewritten as

$$L_a(d_0, d_1) = L_a(0, d_1) - L_a(0, d_0). \qquad (2)$$

The example in Figure 2 becomes with Equation 2

$$L_a^{total} = L_a(0, x_0) - L_a(0, x_1) + L_a(0, x_2),$$

i.e. it is sufficient to identify the boundaries between lit and unlit regions. The order in which the boundaries are processed does not matter, and the boundaries no longer need to be grouped into intervals along the view ray.

Generalized to an arbitrary number of lit regions, the airlight contribution, L_a^{total}, along a view ray is

$$L_a^{total} = \sum_n s_n L_a(0, x_n), \qquad (3)$$

where s_n equals one if the n-th boundary at x_n represents a transition from a lit region to an unlit one. Otherwise s_n equals negative one. An example is provided in Figure 3a.

The sum in Equation 3 requires that there is no overlap between regions. Overlapping regions will cause problems with false contributions that incorrectly remove too much airlight, as demonstrated in Figure 3b.

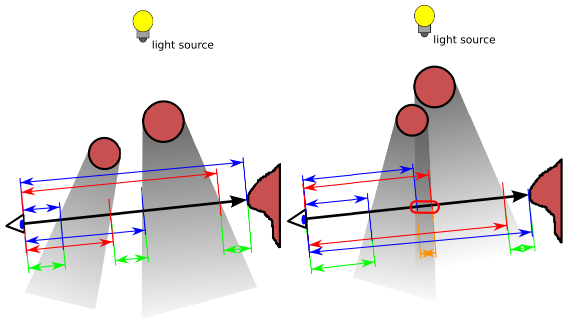
It is possible to find the lit regions using a shadow volume algorithm, where silhouette edges of meshes are extruded to form the boundaries of lit regions. However, this generally results in several overlapping regions. It is possible to eliminate these overlapping regions by sorting the shadow volume's polygons, as demonstrated by Venceslas et. al. [VDS06] and James [Jam03]. However, since the polygons can intersect with each other, the sorting must be performed per pixel for accurate results.

Instead we use shadow maps to reconstruct a light volume with no overlapping regions. This light volume encloses the parts of the participating medium that are directly illuminated. We will refer to these parts of space as the *directly illuminated volume*.

4. Extruding Directly Illuminated Volumes from Shadow Maps

Consider a polygonal mesh that represents the directly illuminated volume, as shown in Figure 4d. If we have such a mesh, calculating the airlight contribution for each pixel is greatly simplified: we can render the mesh from the point of view of the camera, effectively running a fragment shader for each eye-ray intersection with the boundary of the directly illuminated volume. Each intersection represents a term in Equation 3, i.e. each invocation of the fragment shader calculates one term of the sum. The sign of the contribution is determined by the facing of the polygon: if the polygon is back-facing, we are entering an unlit region, and set s_n to one. Else, if the polygon is front-facing, we are entering a lit region where s_n equals negative one.

We use a GPU implementation of McCool's shadow volume extrusion [McC00] to generate the directly illuminated volumes from a shadow map. McCool observes that such a



(a) **(b)**

Figure 3: *Example scenes with two distinct unlit regions (a), and two overlapping unlit regions (b). Transitions between lit and unlit regions along the view ray are considered. When entering an unlit region, a positive airlight contribution is registered (marked blue in the figure). When entering a lit region, a negative airlight contribution is added (marked in red). The total airlight along the view ray is marked in green. In (b), the overlap gives rise to a false negative airlight contribution, marked orange, which demonstrates why using plain shadow volumes could lead to an incorrect result.*

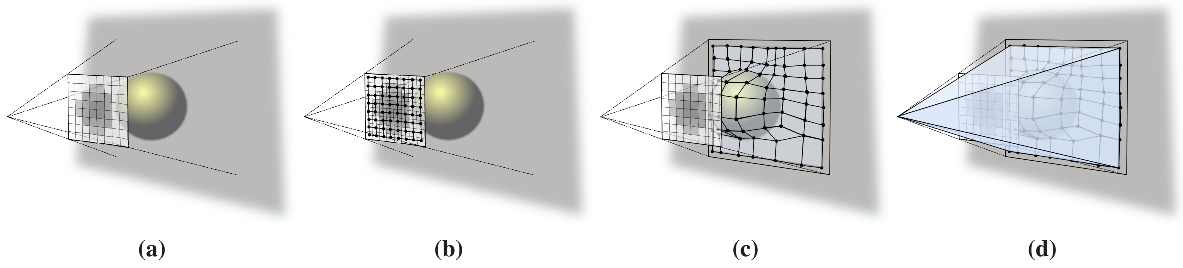


Figure 4: *Creation of the polygonal mesh representing the directly illuminated volume. In (a) a shadow map is rendered, then (b) a pre-generated rectangular mesh is rendered displaced by the depth values in the shadow map (c). Finally, in (d), the border edges are connected to the light sources origin in order to construct a closed volume.*

volume, unlike traditional shadow volumes generated from the scene geometry, has no nesting or overlapping.

The directly illuminated volume is constructed by using a base geometry, e.g. a grid, where the shadow map is used to displace each of the vertices (see Figures 4a through 4c). For an omnidirectional light source, six such displaced grids form the complete light volume. For a directional light source, the borders of the shadow map are connected to the position to the light source in order to form a closed light volume (Figure 4d).

The resulting mesh is clearly an approximation to the true volume's surface, as only a discrete set of samples are taken in the shadow map and distances between samples will be linearly interpolated. Generating the volume from a too-low resolution shadow map will lead to visible aliasing artifacts, especially for moving light sources. However, in many cases, as demonstrated in Section 6, due to the smooth nature of airlight, fairly coarse approximations to the true surface can still produce visually pleasing volumetric shadowing effects.

4.1. Adaptive Tessellation

High resolution shadow maps will produce a large amount of geometry, e.g. a shadow map with a resolution of 1024×1024 will generate a mesh with more than two million triangles.

In some cases it is possible to simply reduce the shadow map resolution. However, this is not ideal in all cases. Many scenes produce shadow maps with sharp edges between fairly large flat regions.

We can adaptively tessellate the generated mesh to provide higher resolution in regions with a high variance, and use lower resolution for the remaining parts. Our tessellation algorithm relies on edge detection performed with a Laplacian filter. From the original shadow map, we create an edge map which in turn is used to build a simple tree, similar to a mip map hierarchy. Each texel in the hierarchy represents a quad in one possible tessellation level and contains information on: (a) Whether the quad needs further subdivision (given some threshold) and (b) Which of the quad's neighbors are further subdivided. The information about neighbors is used to ensure that the final tessellation level between connected quads never differs by more than one level. This avoids T-junctions.

Adaptive tessellation starts with a coarse grid of quads, which corresponds to a specific level in the edge map hierarchy. Each quad checks if the corresponding texel in the edge map indicates that the quad should be further tessellated. If further tessellation is required, four new quads are emitted for further processing. If no further tessellation is required, a number of triangles are emitted instead; these triangles are part of the final mesh representing the light volume.

The configuration of emitted triangles depends on the neighborhood. If all neighbors have the same tessellation level, a quad consisting of two triangles is created. If a neighbor has a higher tessellation level, the common edge has to be subdivided. In the worst case, all four edges need to be subdivided. There are 16 distinct configurations possible. We identify the configuration and emit a matching triangle-mesh in order to avoid T-junctions. At most, six triangles are emitted for a single quad.

The current implementation uses multiple passes of transform feedback in conjunction with geometry shaders. Each tessellation level requires two passes. In the first pass, quads that require further tessellation are filtered. The second pass emits the final triangles, as described above.

5. Algorithm

As explained in Section 4, the light volume is rendered in a separate pass, to add airlight to the image. Naturally, there will be no airlight contribution from segments of the eye ray that lie beyond the first intersection with the scene (that is, at depths greater than that stored in the depth buffer) and care must be taken to handle this correctly. It is therefore tempting to render the volume with depth-testing enabled (but not updating depth). However, as the shadow map is inexact, the light volume polygons that in reality should lie exactly on

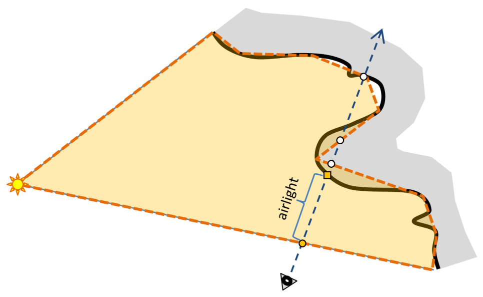


Figure 5: *2D Illustration of a light volume and its intersections with the eye-ray. When the volume is rendered, fragments representing intersections beyond the depth buffer will be generated (white circles on the eye-ray). These are clamped to the depth buffer value (orange square) and processed as usual.*

the lit surfaces will in practice have depths that are sometimes beyond and sometimes in front of the depth buffer (see Figure 5) and so artifacts similar to the well known "surface acne" would appear.

A simple solution is to bias the extrusion of the light volume so that the volume is pushed closer to the viewer. In many simple cases (e.g. light cast on flat surfaces) this would work and would allow us to render the volumes with depth-testing enabled, but it requires careful scene-dependent tweaking. Since the airlight contribution is found by adding and subtracting potentially very large values, insufficient biasing leads to very unpleasant artifacts.

Instead, in our implementation we simply disable depth-testing but present the depth buffer as a texture to the fragment-shader. When a fragment of a light-volume polygon has a depth that is larger than that stored in the depth buffer, it is simply clamped to the value from the depth buffer and then treated as usual. This solution introduces no visible artifacts, but does require us to process parts of the light volume that do not contribute to the airlight for the pixel.

To summarize, the algorithm consists of the following steps:

1. Render shadow map (or shadow-cube for omnidirectional light sources).
2. Render scene as seen from the observer with diffuse lighting, attenuating incoming and outgoing light due to absorption and scattering in media. Hard shadows are calculated using a standard shadow map.
3. Construct the geometry representing the directly illuminated volume, with or without adaptive tessellation.
4. Render that geometry with depth testing disabled and additive blending enabled. The fragment shader evaluates one term of Equation 3 with x_n clamped to the z-buffer depth.

Table 1: *Frame rates for adaptively versus statically tessellated light volumes. Frame rates include time required by rendering of the view and shadow maps. These times are not included in the detailed breakdowns. Detailed times are shown for Laplacian edge detection, building of the edge hierarchy, the adaptive tessellation passes and the final rendering of the light volume. The current implementation of the tessellation algorithm becomes viable first at high shadow map resolutions. The Sponza view (66k tris) is shown in Figure 8a; the Yeah Right view (188k tris) in Figure 8c.*

		Static Tessellation	*Adaptive Tessellation*
Sponza	sm = 1024^2	86 FPS	64 FPS
	Laplacian	-	0.66 *ms*
	build edge map	-	0.68 *ms*
	tessellate	-	3.11 *ms*
	render volume	9.0 *ms*	7.51 *ms*
	sm = 2048^2	35 FPS	35 FPS
	sm = 4096^2	10 FPS	18 FPS
	Laplacian	-	10.11 *ms*
	build edge map	-	9.14 *ms*
	tessellate	-	4.13 *ms*
	render volume	93.9 *ms*	17.75 *ms*
Yeah Right	sm = 1024^2	32 FPS	29 FPS
	Laplacian	-	0.74 *ms*
	build edge map	-	0.68 *ms*
	tessellate	-	2.59 *ms*
	render volume	9.0 *ms*	12.27 *ms*
	sm = 2048^2	21 FPS	21 FPS
	sm = 4096^2	8 FPS	12 FPS
	Laplacian	-	10.19 *ms*
	build edge map	-	9.11 *ms*
	tessellate	-	15.54 *ms*
	render volume	96.35 *ms*	27.54 *ms*

6. Results

We have implemented three different light volumes, one for an omni light source and two for directed point light sources (with and without adaptive tessellation). All tests were performed on an NVIDIA GTX280 GPU.

Other than the standard rendering of the scene for viewing and for the shadow map, the performance is largely independent of the number of triangles in the scene. Instead, frame rates depend mostly on the shadow map resolution. Figure 6 shows a simple scene rendered three times with varying shadow map resolutions. Frame rates vary from about 230 FPS ($32 \times 32 \times 6$ shadow map), to $\sim$ 165 FPS ($128 \times 128 \times 6$), and finally reach $\sim$ 25 FPS for a $512 \times 512 \times 6$ shadow map.

Ray-marching based methods are currently the only reasonable alternative solution for real time algorithms capable of producing high quality results for participating media. In Figure 7, we compare our algorithm to ray-marching. The ray-marching is performed in a fragment shader, which evaluates a constant number of samples, starting at the furthest depth. An airlight contribution is only calculated when a transition between a lit and an unlit region is detected. Note that the ray marching implementation is not fully optimized, i.e. performance could be improved.

For the view in Figure 7, approximately 400 samples are required for results with equivalent quality to our algorithm; 400 samples per pixel would be quite slow on any current hardware and ray-marching implementation. Our algorithm calculates the exact solution, with respect to the shadow map, at about 96 FPS.

Also note that this scene would benefit very little from the optimizations suggested by Wyman et. al. [WR08], as the closest and furthest shadow planes span the entire depth range visible in the image. Generally, our method is faster for equal quality images. Unlike Wyman et. al. [WR08] and Engelhardt and Dachsbacher [ED10], our method will always be exact with respect to the shadow map.

Table 1 displays frame rates for several shadow map resolutions with and without adaptive tessellation for the views shown in Figures 8a and 8c. An example of an adaptively tessellated mesh can be seen in Figure 8b. The current adaptive tessellation algorithm, based on geometry shaders, has a large overhead. Therefore, benefits are first observed for large shadow maps.

If the scene contains a large amount of very irregular geometry, e.g. Figure 8c, the adaptive tessellation also becomes less beneficial.

7. Future Work

Adaptive Tessellation Our current implementation of the tessellation algorithm, based on geometry shaders and transform feedback, has quite large overheads. It is possible to decrease these overheads somewhat with newer API and hardware functionality, e.g. the recently announced `GL_ARB_transform_feedback3`.

It would also be interesting to experiment with the new shaders specifically targeting tessellation.

Generalized Light Sources Several papers discuss more general environments, for instance textured light sources [PP09] and non-uniform and anisotropic participating media [ZHG*07]. Our method should be adaptable to non-uniform or anisotropic participating media, as long as Equation 1 can be solved efficiently for these configurations.

Textured light sources might be more problematic, as the textures must be sampled at regular intervals. At that point, ray-marching algorithms become a natural choice of solution.

References

[DYN02] Dobashi Y., Yamamoto T., Nishita T.: Interactive rendering of atmospheric scattering effects using graphics hardware. In HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 99–107.

[ED10] Engelhardt T., Dachsbacher C.: Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games (New York, NY, USA, 2010), ACM, pp. 119–125.

[IJTN07] Imagire T., Johan H., Tamura N., Nishita T.: Anti-aliased and real-time rendering of scenes with light scattering effects. Vis. Comput. 23, 9 (2007), 935–944.

[Jam03] James R.: True volumetric shadows. In Graphics programming methods (Rockland, MA, USA, 2003), Charles River Media, Inc., pp. 353–366.

[McC00] McCool M. D.: Shadow volume reconstruction from depth maps. ACM Trans. Graph. 19, 1 (2000), 1–26.

[Mit07] Mitchell K.: Volumetric light scattering as a post process. In GPU Gems 3 (2007), Addison-Wesley, pp. 275–285.

[NMN87] Nishita T., Miyawaki Y., Nakamae E.: A shading model for atmospheric scattering considering luminous intensity distribution of light sources. Proceedings of SIGGRAPH (1987), 303–310.

[PP09] Pegoraro V., Parker S. G.: An Analytical Solution to Single Scattering in Homogeneous Participating Media. Computer Graphics Forum 28, 2 (april 2009), 329–335.

[PSP09] Pegoraro V., Schott M., Parker S.: An analytical approach to single scattering for anisotropic media and light distributions. Proceedings of Graphics Interface 2009 (2009), 71–77.

[SRNN05] Sun B., Ramamoorthi R., Narasimhan S., Nayar S.: A practical analytic single scattering model for real time rendering. In ACM SIGGRAPH 2005 Papers (2005), ACM, p. 1049.

[TU09] Toth B., Umenhoffer T.: Real-time volumetric lighting in participating media. EUROGRAPHICS Short Papers (2009).

[VDS06] Venceslas B., Didier A., Sylvain M.: Real time rendering of atmospheric scattering and volumetric shadows. Journal Of WSCG (2006).

[WR08] Wyman C., Ramsey S.: Interactive volumetric shadows in participating media with single-scattering. IEEE Symposium on Interactive Ray Tracing, 2008. RT (2008).

[ZHG*07] Zhou K., Hou Q., Gong M., Snyder J., Guo B., Shum H.-Y.: Fogshop: Real-time design and rendering of inhomogeneous, single-scattering media. In PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (Washington, DC, USA, 2007), IEEE Computer Society, pp. 116–125.

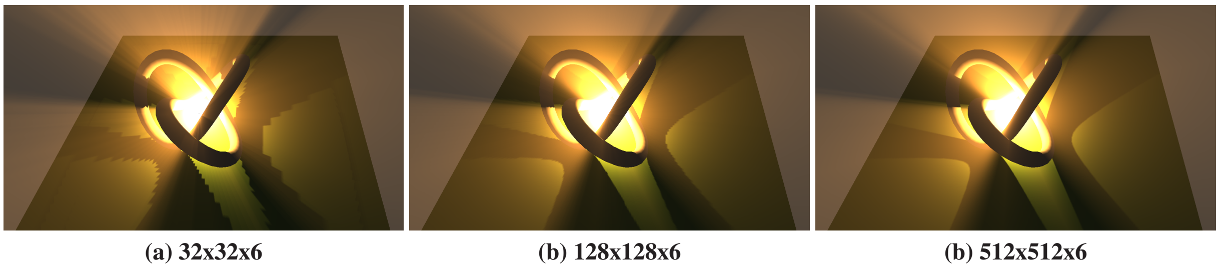


Figure 6: *A simple scene with an omni light source. Shadow map resolutions (for each cube face) are* 32×32 *(a),* 128×128 *(b) and* 512×512 *for (c). At the lowest resolution,* 32×32*, aliasing is clearly observed in both the airlight and the cast shadows. In (b), the airlight is already very smooth, while the cast shadows still display some aliasing. In (c), the airlight is mostly indistinguishable from (b). Frame rates are, from (a) to (c),* ~ 230 *FPS,* ~ 165 *FPS and* ~ 25 *FPS.*

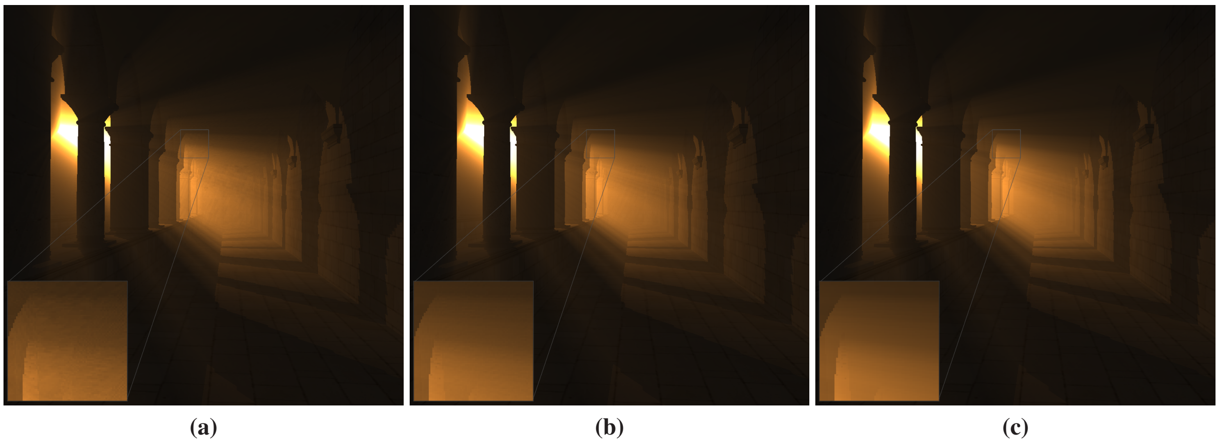


Figure 7: *A foggy Sponza hallway. From left to right: Ray-marching with* 150 *samples uniformly distributed along the view-rays (*25 *FPS), Ray-marching with* 400 *samples (*10 *FPS) and our method (*96 *FPS). The images were rendered at a resolution of* 1024×1024*, and the shadow map was set to* 512×512*.*

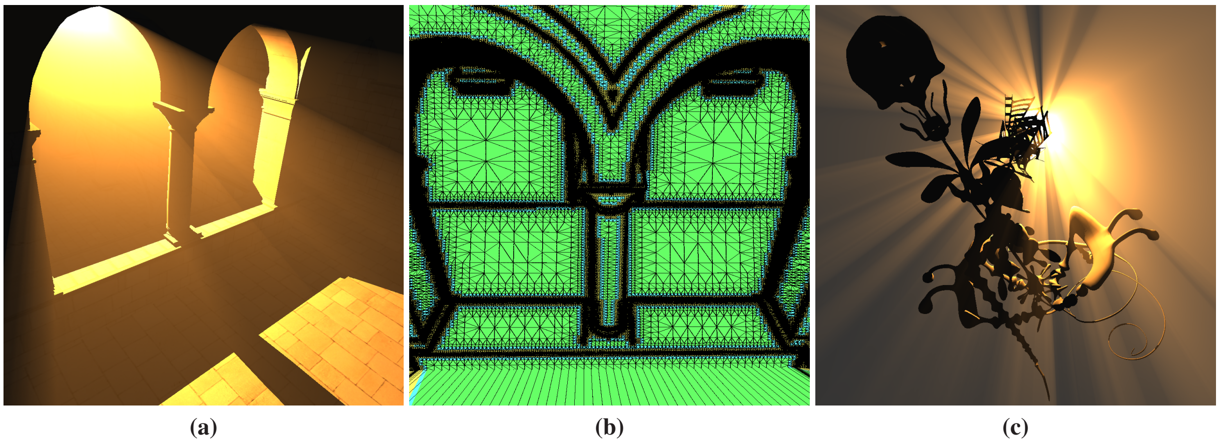


Figure 8: *Figures (a) and (c): views demonstrating the tessellated light volumes (at* 18 *and* 12 *FPS, respectively). The shadow map size is* 4096×4096*, which would yield over 32M triangles. The middle image (b) shows a cutout of the tessellated light volume from (a). Flat surfaces use a quite low tessellation level, whereas edges are successfully located and tessellated much more aggressively.*