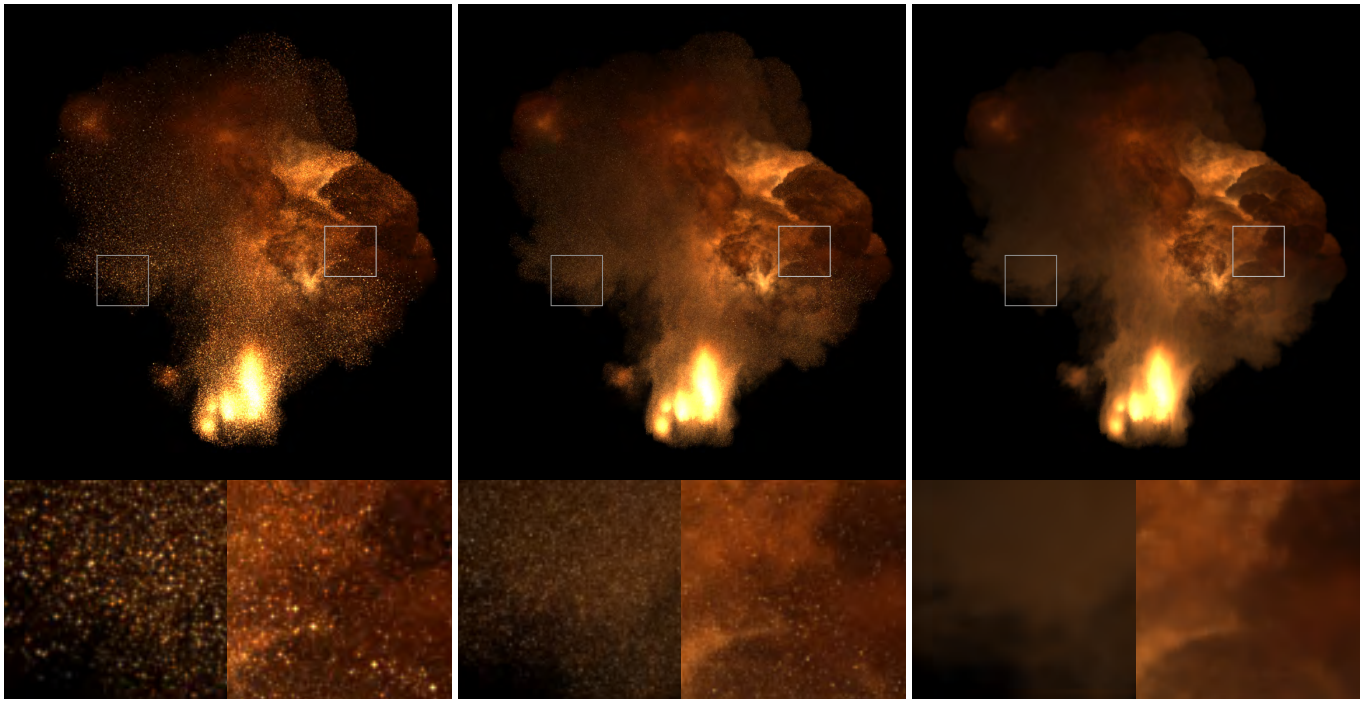
Lighting Grid Hierarchy for Self-illuminating Explosions

CAN YUKSEL, DreamWorks Animation
CEM YUKSEL, University of Utah



Path Tracing
21 minutes

Path Tracing
13 hours

Lighting Grid Hierarchy (Ours)
20 minutes

Fig. 1. An example frame from an explosion simulation rendered with multiple scattering using path tracing and our lighting grid hierarchy (including precomputation time). Notice that the path tracing result with render time equal to our method includes an excessive amount of noise, which is not fully eliminated after almost 40 times longer rendering using path tracing.

Rendering explosions with self-illumination is a challenging problem. Explosions contain animated volumetric light sources immersed in animated smoke that cast volumetric shadows, which play an essential role and are expensive to compute. We propose an efficient solution that redefines this problem as rendering with many animated lights by converting the volumetric lighting data into a large number of point lights. Focusing on temporal coherency to avoid flickering in animations, we introduce *lighting grid hierarchy* for approximating the volumetric illumination at different resolutions. Using this structure we can efficiently approximate the lighting at any point inside or outside of the explosion volume as a mixture of lighting contributions from all levels of the hierarchy. As a result, we are able to capture high-frequency details of local illumination, as well as the potentially strong impact of distant illumination. Most importantly, this hierarchical structure allows us to efficiently precompute volumetric shadows, which substantially accelerates the lighting computation. Finally, we provide a scalable approach for computing the multiple scattering of light within the smoke volume using our lighting grid hierarchy. Temporal coherency is achieved by relying on continuous formulations at all stages of the lighting approximation. We show that our method is efficient and effective approximating the self-illumination of explosions with visually indistinguishable results, as compared to path tracing. We also show that our method can be applied to other problems involving a large number of (animated) point lights.

CCS Concepts: • **Computing methodologies** → **Rendering**;

Additional Key Words and Phrases: Explosion rendering, many-lights, virtual point lights, participating media, translucent shadows, multiple scattering

ACM Reference format:
Can Yuksel and Cem Yuksel. 2017. Lighting Grid Hierarchy for Self-illuminating Explosions. *ACM Trans. Graph.* 36, 4, Article 110 (July 2017), 10 pages. https://doi.org/10.1145/3072959.3073604

1 INTRODUCTION

Explosions are frequently featured in visual entertainment. While there are effective methods for simulating explosions in computer graphics [Feldman et al. 2003; Kawada and Kanai 2011; Kwatra et al. 2010; Selle et al. 2005], rendering them with self-illumination has been challenging. Unfortunately, unbiased stochastic sampling methods based on path tracing fall short, since the high-frequency details of the volumetric illumination often require a large number of samples to converge to a low-noise solution. Considering the fact that explosions are typically represented as volume data and ray tracing high-resolution volumes can be expensive, path tracing leads to completely impractical render times (Figure 1).

Our approach is converting this expensive volumetric lighting problem into illumination with many animated point lights. Rendering images using a large number of light sources has obvious difficulties regarding computational performance. Nonetheless, prior work on this many-lights problem includes some elegant scalable solutions that can provide sub-linear performance using hierarchical clustering of light sources [Walter et al. 2006, 2005] or approximate computations of the lighting matrix [Hašan et al. 2007; Ou and Pellacini 2011]. However, when it comes to animation, especially with animated light sources, these methods introduce substantial amount of temporal flickering [Hašan et al. 2008], which limits their applications. Being able to handle animated lights without temporal flickering is a crucial challenge for efficiently rendering explosions with self-illumination.

In this paper we introduce a scalable solution for many animated lights using a hierarchical grid structure. This structure contains levels of grids with successively lower resolutions, where each grid vertex is treated as a light source that approximates the point lights around it. Our *lighting grid hierarchy* is carefully constructed to avoid flickering in animation, without having to process consecutive frames together. Unlike typical hierarchial structures that can be represented using trees, our lighting grid hierarchy forms a more connected graph, where each node can have multiple parents. During lighting computation we combine the contributions of multiple levels using overlapping *blending functions*. Thus, we avoid binary decisions in clustering and lighting computation, which we identify as a primary source of temporal flickering in hierarchical lighting solutions.

Another major source of temporal flickering with approximate lighting solutions is shadows. This presents a particularly important challenge for explosion rendering, where the light emitting voxels of the simulation data are often surrounded by an animated smoke layer with varying density. Therefore, we pair our lighting solution with a new volumetric shadow mapping approach designed for our grid structure. Our volumetric shadow mapping approach not only avoids flickering by carefully utilizing pre-filtered smoke data, but also provides more than an order of magnitude speedup as compared to using volume tracing for shadow computation.

We further extend our scalable lighting approach to incorporate a computationally efficient solution for multiple scattering of light within the smoke volume. As a result, we can render self-illuminating explosions and their smoke with self-shadows and multiple scattering involving millions of animated lights within several minutes in a production renderer with no noticeable noise or temporal flickering. Such an example is shown in Figure 1, along with comparison images generated using path tracing.

2 RELATED WORK

The many-lights problem has a long history in computer graphics [Dachsbacher et al. 2014]. Earlier methods relied on ordering lights based on their contributions [Ward 1994], stochastic sampling [Shirley et al. 1996], light clustering using octrees [Paquette et al. 1998], or selecting lights using precomputed visibility culling [Fernandez et al. 2002]. The many-lights problem received considerable attention after the instant radiosity method [Keller 1997] converted the global illumination problem to direct illumination from many virtual point lights (VPLs). Instant global illumination [Wald et al. 2002] extended this idea and further accelerated in scenes with high occlusion using a path tracing preprocess for light selection [Wald et al. 2003]. Virtual spherical lights [Hašan et al. 2009] were introduced for eliminating the singularities of VPLs and improving the illumination with glossy surfaces.

A more general solution to the many-lights problem was provided using Lightcuts [Walter et al. 2005], a method that builds a binary tree from the lights and determines the part of the tree that should be evaluated during shading, based on a highly conservative error bound for achieving sub-linear performance. This was later extended to high dimensional integrations involving volume scattering, depth of field, and motion blur [Walter et al. 2006]. Extensions of the Lightcuts approach include a progressive GPU-friendly variant [Davidovič et al. 2012], bidirectional Lightcuts [Walter et al. 2012] for improving the weighting scheme to support a wider range of materials, and an out-of-core GPU implementation [Wang et al. 2013] for rendering large scenes.

An alternative formulation of the many-lights problem was provided using an approximate evaluation of the lighting matrix [Hašan et al. 2007], where rows and columns correspond to the points to be shaded and light sources, respectively. Separating the illumination into local and global components allowed handling glossy surfaces [Davidovič et al. 2010]. The lighting matrix approximation was further accelerated by introducing cuts to adaptively evaluate a fraction of the lights [Ou and Pellacini 2011] and using a reduced matrix that approximates the entire lighting matrix [Huo et al. 2015]. These approaches are not ideal for rendering volume data, as it involves orders of magnitude more points to be shaded.

Many-lights methods were also used for rendering participating media, where most methods concentrate on eliminating singularities of VPLs using a path tracing step [Kollig and Keller 2006; Raab et al. 2008], introducing approximate bias compensation [Engelhardt et al. 2012], or generating rays [Frederickx et al. 2015; Huo et al. 2016; Novák et al. 2012] or beams [Novák et al. 2012] instead of VPLs.

On the other hand, the temporal coherence issue received less attention. Hašan et al. [2008] recognized the flickering in animated sequences using prior methods and attempted to reduce flickering by computing the lighting tensor that includes all frames of the animation. Dong et al. [2009] used a k-means clustering method that utilizes the clustering results of the previous frame for introducing temporal coherence. Nonetheless, these approaches had only limited success.

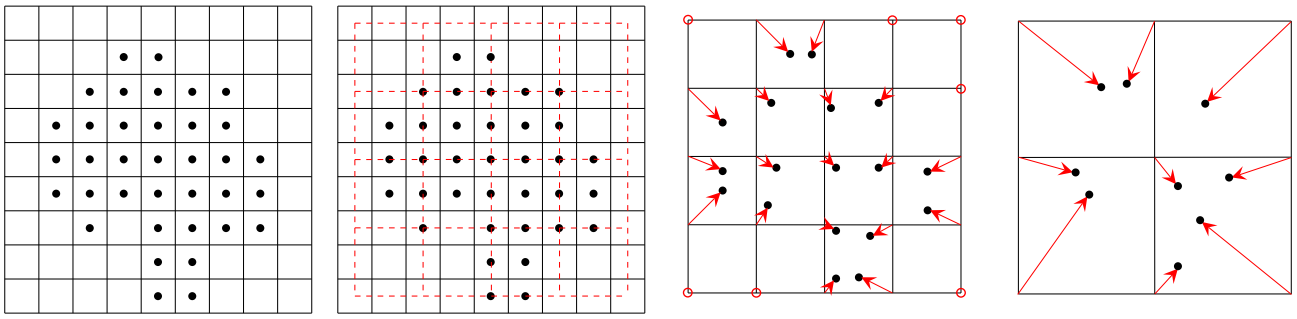


Fig. 2. Our lighting grid hierarchy for explosion rendering. We begin with the explosion simulation grid and generate point lights (shown as black dots) in voxels with high temperature values. We place the highest resolution (level 1) lighting grid, such that vertices of the grid are aligned with voxel centers. For each vertex of the lighting grid at any level, we keep the illumination center, shown as black dots along with offset arrows from their grid vertices.

3 LIGHTING GRID HIERARCHY

Our lighting grid hierarchy is a collection of grids with different resolutions. All of these grids correspond to the same volume in space. Given a set of point light sources $\mathbb{S}_0$ as input, each grid in the hierarchy approximates the entire set $\mathbb{S}_0$ on its own, such that each grid vertex is treated as a light source that represents a portion of the light emission in $\mathbb{S}_0$ around it. Hence, each level of the grid hierarchy allows us to access the entire lighting information at a different sampling resolution.

We adjust the grid resolutions such that the grid at level ℓ has half the resolution (in all dimensions) as compared to the grid at level $\ell - 1$. Given a user defined spacing of the grid vertices h_1 at level 1, the spacing for other levels is calculated using $h_\ell = 2^{\ell-1} h_1$. In this notation $\ell = 0$ corresponds to $\mathbb{S}_0$, and $h_0 = h_1/2$.

Note that in general $\mathbb{S}_0$ can be any collection of point lights. For rendering explosions, however, we generate a point light at the center of each simulation voxel that has a temperature high enough to cause light emission. Therefore, for rendering explosions we take h_0 as the simulation resolution and place the level 1 grid with half a voxel offset, as shown in Figure 2. Note that in general the resolution of level 1 grid does not have to be tied to a simulation resolution and it can be specified independently. Each vertex of the lighting grid at level ℓ corresponds to a light in $\mathbb{S}_\ell$, and it stores the total light intensity $\mathbf{I}_\ell$ and the illumination center $\mathbf{p}_\ell$ (shown as offset vectors in Figure 2) for a subset of lights in $\mathbb{S}_0$ that it represents.

3.1 Building the Hierarchy

Determining which lights in $\mathbb{S}_0$ are represented by which lights in $\mathbb{S}_\ell$ has utmost importance. A naïve choice would be clustering the lights based on their distances to the grid vertices. However, clustering uses binary decisions, so it leads to flickering. We circumvent this problem by allowing each light in $\mathbb{S}_0$ to have multiple representations in $\mathbb{S}_\ell$. For each light source $j \in \mathbb{S}_0$, we consider the corresponding grid cell at level ℓ, the volume of which encapsules the position of the light $\mathbf{p}_{0,j}$. We distribute the intensity of each light $\mathbf{I}_{0,j}$ to the vertices of the corresponding grid cell using trilinear weights $w_{i,j}$ of grid vertices i of level ℓ. Hence, the total intensity $\mathbf{I}_{\ell,i}$ of the grid vertex i at level ℓ is

$$\mathbf{I}_{\ell,i} = \sum_{j \in \mathbb{S}_0} w_{i,j}\, \mathbf{I}_{0,j}\ , \tag{1}$$

such that $w_{i,j}$ is zero for any light j that is not in one of the eight grid cells ($2 \times 2 \times 2$ block) around the grid vertex i at level ℓ. Note that this formulation distributes the intensity of each light to multiple lights at level ℓ.

For better representation of the underlying light sources, we store an illumination center per grid vertex, which is treated as the light position, using

$$\mathbf{p}_{\ell,i} = \frac{\sum_{j \in \mathbb{S}_0} v_{i,j}\, \mathbf{p}_{0,j}}{\sum_{j \in \mathbb{S}_0} v_{i,j}}\ , \tag{2}$$

where $v_{i,j} = w_{i,j} \left\|\mathbf{I}_{0,j}\right\|$ is the contribution of light j and $\left\|\mathbf{I}_{0,j}\right\|$ denotes the luminance component of the light intensity. This formulation places the illumination center to the weighted average of the light positions $\mathbf{p}_{0,j}$ that contribute to the light i at level ℓ. Note that the illumination center of a vertex can fall anywhere within the eight cells ($2 \times 2 \times 2$ grid block) around the grid vertex.

Using this formulation we can generate the illumination grid for all levels up to a user-defined maximum level ℓ_{max}. Note that the input light positions $\mathbf{p}_0$ of our explosion simulation are not animated (and placed at voxel centers), but the light intensities $\mathbf{I}_0$ change at each frame, causing the illumination centers $\mathbf{p}_\ell$ at higher (coarser) levels to move accordingly. Most importantly, this simple formulation avoids random decisions used for clustering [Walter et al. 2005] that lead to flickering, since each randomly generated clustering leads to a different illumination approximation for the entire image.

It is also possible to compute intensity $\mathbf{I}_{\ell,i}$ and position $\mathbf{p}_{\ell,i}$ values for level ℓ directly from the lights in the lower (finer) grid levels $\mathbb{S}_{\ell-1}$, instead of the input lights $\mathbb{S}_0$. While this accelerates the build operation, it is not exactly equivalent to using $\mathbb{S}_0$ directly and introduces some smoothing in lighting estimation.

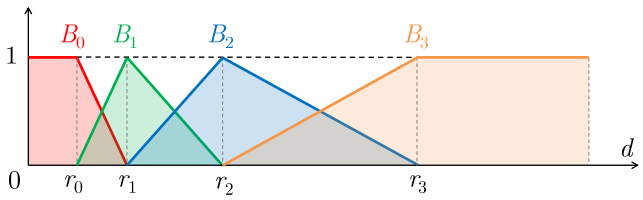


Fig. 3. Blending functions for levels 0 through 3 ($\ell_{max} = 3$).

3.2 Estimating Lighting

We use our lighting grid hierarchy for efficiently computing the lighting at a given point $\mathbf{x}$ in space by querying $\mathbb{S}_\ell$ at all levels ℓ for accessing the lighting information with different resolutions. We use different grid levels for approximating the incoming illumination from different distances. For achieving temporal coherency, we avoid using sharp thresholds that would determine which distance would correspond to which resolution. Instead, we blend the illumination contributions of different resolutions, such that the illumination contribution of a light i at level ℓ on point $\mathbf{x}$ is calculated using

$$\hat{\mathbf{I}}_{\ell,i}(\mathbf{x}) = V(\mathbf{x}, \mathbf{p}_{\ell,i})\, g(d)\, B_\ell(d)\, \mathbf{I}_{\ell,i}\,, \tag{3}$$

where $d = ||\mathbf{x} - \mathbf{p}_{\ell,i}||$ is the distance of $\mathbf{x}$ to the illumination center $\mathbf{p}_{\ell,i}$, V is the visibility function, g is the light fall-off function, and B_ℓ is the *blending function* that determines the influence region of level ℓ. Let $r_\ell = \alpha\, h_\ell$ be the influence radius of level ℓ, determined by a user-defined parameter α. Our blending functions linearly increase from zero at $r_{\ell-1}$ to one at r_ℓ (except for the first level), and linearly decrease down to zero at $r_{\ell+1}$ (except for the last level), as shown in Figure 3. Notice that the blending functions form a partition of unity and that the influence regions of neighboring levels overlap to avoid temporal flickering.

Using these blending functions, for a given shading point $\mathbf{x}$, we only need to query the lights in $\mathbb{S}_\ell$ within $2r_\ell$ radius for each level ℓ. The influences of the nearby light sources come from the lower (finer) levels and the higher (coarser) levels are used for distant illumination. This formulation allows us to achieve sub-linear performance in shading with many lights. The user defined parameter α controls the accuracy and the performance of the illumination computation. Larger values of α increase the influence region of lower (finer) levels, thereby increasing both the number of lights included in shading and the accuracy.

4 SHADOWS

Typically, most of the lighting computation for each light is devoted to shadow generation. Even though our method substantially reduces the number of lights considered during each shading operation (just like other sub-linear lighting approximations), volumetric shadows still dominate the rendering computations. Unfortunately, volume tracing even for a small fraction of the light sources leads to impractical render times. Therefore, we introduce a simple shadow precomputation method that exploits properties of our lighting grid hierarchy for efficiently computing shadows during rendering.

Prior many-lights methods for shadows include reusing shadows from previous frames [Laine et al. 2007] and computing shadows approximately [Ritschel et al. 2008]. Instead, we compute a volumetric cube-map shadow for each light at each level, completely independently at each frame. Obviously, computing full-resolution shadow maps for all light sources in $\mathbb{S}_0$ alone would be memory intensive. However, since the influence regions of our light sources in lower (finer) levels are limited, the shadow maps at these levels can be low-resolution. Let h be the size of a voxel that stores the smoke volume data (in our examples $h = h_0$). The sufficient cube-map resolution should have texels of size h at r_ℓ distance from the light, where the blending function B_ℓ reaches its apex. Such a cube-map would have $24(r_\ell/h)^2$ texels. Therefore, for lower (finer) levels of the hierarchy, a relatively small cube-map is sufficient for capturing the full shadow detail of the smoke volume data at the apex of the blending function. These cube-map texels can store a variant of deep shadow maps [Gautron et al. 2012; Lokovic and Veach 2000] for encoding the volumetric shadow data or simply a few extinction values at different distances. Note that this approach is not applicable to prior many-lights methods, since they do not limit the influence regions of lights.

Since r_ℓ doubles at successive levels (i.e. $r_\ell = 2\, r_{\ell-1}$), the sufficient cube-map resolution quadruples at higher (coarser) levels. On the other hand, the number of light sources at higher (coarser) levels decreases by a factor between 4 (for light sources scattered over surfaces) to 8 (for light sources scattered volumetrically). Therefore, the total number of shadow map texels for all lights in a level $\mathbb{S}_\ell$ is typically smaller for higher (coarser) levels.

Note that the explosion data may include substantial high-frequency changes in smoke density. Therefore, if the lights in $\mathbb{S}_\ell$ are treated as point lights for computing shadows, even the slightest change in the illumination centers $\mathbf{p}_\ell$ can cause a drastic change in volumetric shadows, which can lead to flickering in successive frames. This is due to the fact that shadows from a point light source in $\mathbb{S}_\ell$ may not accurately approximate the shadows from many individual lights in $\mathbb{S}_0$ that are scattered around it. For this reason, it is important to treat the lights at higher (coarser) levels ($\ell > 0$) as volumetric light sources, rather than point lights.

It is possible to assume that the size of the lights at level ℓ is proportional to the grid separation of that level h_ℓ. However, this can lead to substantial overestimation of the light size (especially for higher levels) and produce excessively smoothed shadows. Instead, we can perform stochastic sampling by picking a random position around $\mathbf{p}_{\ell,i}$ from a desired distribution with a variance that matches the variance $\sigma^2_{\ell,i}$ of the light positions represented by the grid light i using

$$\sigma^2_{\ell,i} = \frac{\sum_{j\in\mathbb{S}_0} v_{i,j}\, \mathbf{p}^2_{0,j}}{\sum_{j\in\mathbb{S}_0} v_{i,j}} - \mathbf{p}^2_{\ell,i} \tag{4}$$

in each direction. Yet, precomputing the shadows using stochastic sampling may require a large number of samples to converge, especially for higher (coarser) levels.

Therefore, we simplify this computation by prefiltering the smoke density for shadow tracing. Let $s_{\ell,i}$ represent the estimated size of a light at level ℓ as the weighted average of the corresponding light distances, such that

$$s_{\ell,i} = \frac{\sum_{j\in\mathbb{S}_0} v_{i,j}\, \left\|\mathbf{p}_{0,j} - \mathbf{p}_{\ell,i}\right\|}{\sum_{j\in\mathbb{S}_0} v_{i,j}}\,. \tag{5}$$

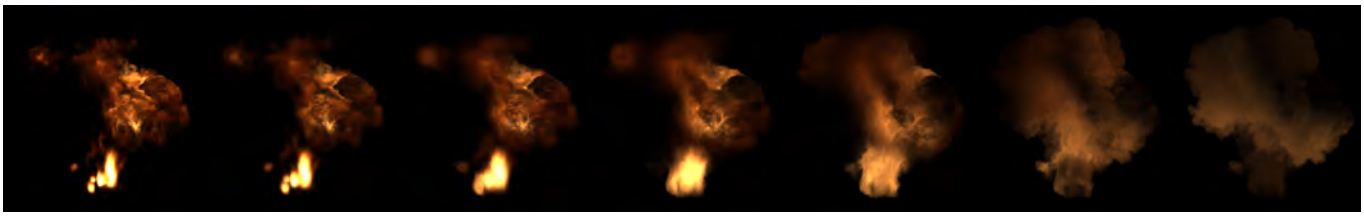


Fig. 4. The lighting contributions of different levels for an example frame taken from the explosion simulation shown in Figure 1. Notice that lower (finer) levels contain high-frequency local illumination and higher (coarser) levels account for distant illumination.

To compute the shadow map of a light $i \in \mathbb{S}_\ell$, we start tracing a ray from the illumination center $\mathbf{p}_{\ell,i}$ towards each shadow texel direction up to the maximum distance $2r_\ell$ (the last level is not bounded). At each sampling step during volume tracing we compute the filtered density value using a pyramidal convolution filter of size δ, such that

$$\delta(d) = \left(1 - \frac{d}{r_\ell}\right) s_{\ell,i} + \frac{d}{r_\ell} h_0 \ , \tag{6}$$

where d is the distance from the illumination center for $d \leq r_\ell$. When $d > r_\ell$, we use $\delta(d) = 0$. This formulation linearly reduces the filter size starting from $s_{\ell,i}$ down to 0 at distance r_ℓ from the light. Note that the pyramidal convolution filter coincides with the trilinear weights we use while computing the light data for the grids.

For efficiently approximating the filtered density values at any point, we can pre-filter the density field. This way, we can efficiently approximate the shadow from volumetric light sources and avoid shadow-related flickering in the final result. What makes this possible is the limited influence region of each light in the lighting grid hierarchy, and it provides more than an order of magnitude reduction in the total render time.

5 MULTIPLE SCATTERING

The lighting computation we discussed so far only includes direct illumination with volumetric shadows. The smoke density also causes multiple scattering of light within the explosion volume. For simplicity, we assume that the scattering function for the smoke component is completely isotropic. This is also the scattering function we use for rendering smoke in all our examples. In this case, our scalable lighting approach provides a mechanism for computing the multiple scattering component as well.

Let lights $\mathbb{S}^0_\ell$ represent the original lights we discussed in previous sections, providing direct illumination. We can generate a new set of virtual point lights $\mathbb{S}^1_\ell$ in the explosion volume for representing light that goes through one scattering event. Hence, illumination computation using $\mathbb{S}^0_\ell \bigcup \mathbb{S}^1_\ell$ would include both single scattering (from direct illumination) and two scattering events.

$\mathbb{S}^1_0$ at level 0 can be easily generated by computing the illumination at the center of each grid voxel of level 0 that has non-zero smoke density using $\mathbb{S}^0_\ell$ and multiplying the incoming illumination with the scattering function. The higher (coarser) levels $\mathbb{S}^1_\ell$ for $\ell > 0$ can be generated in the same way as explained in Section 3. Once we have all levels of $\mathbb{S}^1_\ell$, we can use it for computing $\mathbb{S}^2_0$. For producing multiple scattering data involving up to n scattering events, we generate lights for all scattering depths up to $\mathbb{S}^{n-1}_0$.

The problem is that each scattering event we incorporate adds a new set of lights to our system with a new lighting grid hierarchy. Nonetheless, all lights at level 0 of all scattering depths are placed exactly at voxel centers of the simulation grid. Therefore, we can easily merge the lights of a voxel that correspond to different scattering depths into a single light that incorporates all computed scattering depths.

6 IMPLEMENTATION AND RESULTS

We implemented our lighting grid hierarchy inside the SideFX Houdini software using its VEX language. All explosion images are generated with the SideFX Mantra renderer using its ray tracer for the volume data. The renderer uses ray marching for each primary camera ray sample and calls our shader at each step through the volume if enough smoke density is encountered, until the ray gets fully occluded. Therefore, each primary camera ray sample corresponds to multiple shading operations. The performance results are generated on a computer with dual Intel Xeon E5-2690 v2 CPUs (20 cores total) and 96GB RAM.

Our implementation keeps the generated lights in separate kd-trees for quickly retrieving the lights within r_ℓ radius of a given shading point. We remove the lights weaker than a small threshold from the lower (finer) levels of the hierarchy for additional optimization, but their cumulative contributions are included in higher (coarser) levels. We store the volumetric shadows in cube-map textures as accumulated densities recorded at four predefined distances from the light position. The density values in-between these positions are calculated using linear interpolation. Therefore, the resolution of the light extinction due to volumetric shadow is substantially lower at higher (coarser) levels in our implementation. We use $\alpha = 2$ for all examples in this paper, unless otherwise specified. We use inverse squared fall-off and clamp the minimum light distance to h_0 to avoid the singularity of point lights.

6.1 Explosion Rendering

Figure 1 shows a comparison of our method to path tracing. The path tracer also uses ray marching and it accumulates light emission from the volume at every step until the rays are absorbed. Since we use an isotropic scattering function for smoke, no importance sampling is used. Notice that our method can faithfully reproduce

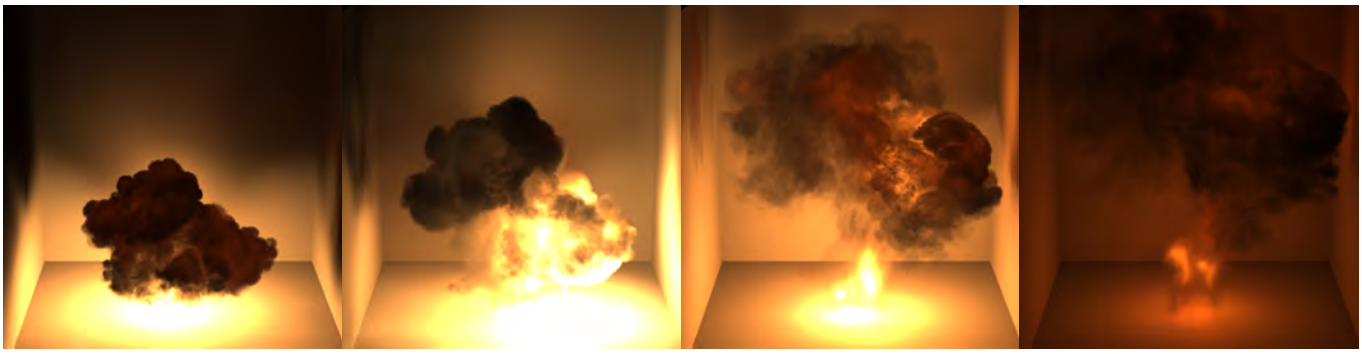
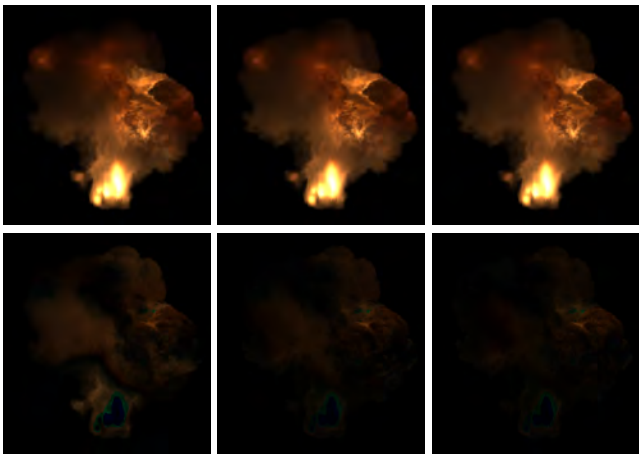


Fig. 5. Frames from the same explosion simulation as Figure 1 in a box that is illuminated using our method.



$\alpha = 1$ $\alpha = 2$ $\alpha = 3$

Fig. 6. The impact of α on single scattering accuracy. The bottom row shows the differences (×2) to the path tracing reference (using clamped low-dynamic-range images).

Single Scattering Multiple Scattering Luminance Diff. ×2

Fig. 7. The impact of multiple scattering for self-illumination is mostly visible near bright areas as a subtle glow.

the small-scale details of the illumination as well as the large-scale overall illumination of the explosion. Furthermore, our method can produce a noise-free image within several minutes (including the precomputation time), while path tracing requires more than an order of magnitude longer time to converge to a low-noise result. Note that this is a particularly challenging frame of this explosion simulation for our method. In other frames, where the illumination is not as widely distributed within the explosion volume, our method becomes substantially more efficient (i.e. produces fewer lights and uses less memory for shadow maps); therefore, it renders even faster, while path tracing produces even more noise for such frames.

The illumination contributions of all levels with our method for the frame in Figure 1 are presented in Figure 4. Notice that the lower (finer) levels contain high-frequency local illumination, and distant illumination is produced by the higher (coarser) levels. The majority of the error in our system comes from these higher (coarser) levels, mainly because our implementation keeps only 4 volumetric shadow values per deep shadow map texel.

Figures 5 and 8 show frames from two explosion simulations. Note that our method can also be used for illuminating and casting shadows onto other objects both inside and outside of the explosion volume. Especially when illuminating objects outside of the explosion volume, our method relies on the lights at higher (coarser) levels; therefore, the lighting computation is very efficient, particularly for distant illumination. In these examples rendering the box surrounding the explosions took only a few seconds per frame, though the box is relatively close to the explosions.

The impact of our α parameter is shown in Figure 6. Note that larger α values reduce the error in our lighting approximation. However, going from $\alpha = 2$ to $\alpha = 3$ more than doubles the precomputation and render time for this frame.

The effect of multiple scattering is displayed in Figure 7. This example uses four levels of scattered light. Thus, the final light reaching the camera includes up to five scattering events. Notice that multiple scattering produces a low-frequency "glow" as a function of both illumination and smoke density. Due to the scattering properties of the smoke medium, the total light intensity drops significantly with each scattering event. Therefore, the fourth scattering depth barely contributes to the total illumination. Even though the impact of multiple scattering can be substantial, the low-frequency difference can be hard to notice in side-by-side comparisons.

Our supplementary video demonstrates that our method avoids temporal flickering. On the other hand, simpler variants that build the hierarchy by clustering the lights (instead of distributing illumination to multiple parents at higher levels), use thresholds for

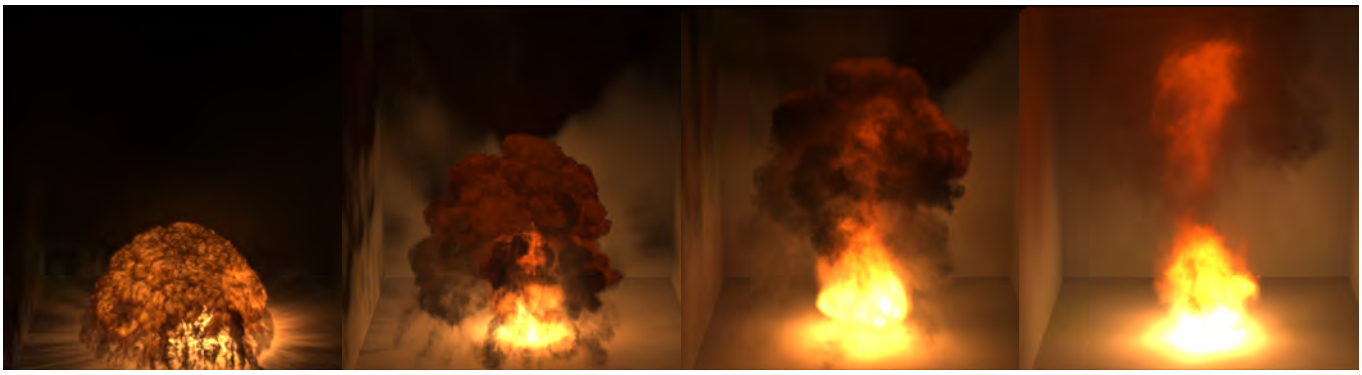
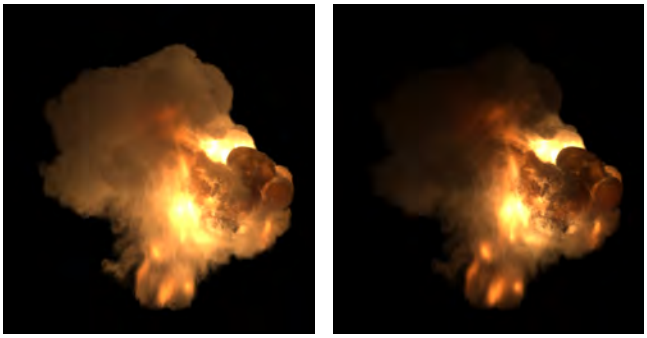


Fig. 8. Frames from another explosion simulation rendered using our lighting grid hierarchy with single scattering.



Grid Vertex Positions ($\mathbf{q}_\ell$) Illumination Centers ($\mathbf{p}_\ell$)

Fig. 9. An example frame showing the importance of placing lights at the illumination centers ($\mathbf{p}_\ell$). When lights are placed at the grid vertex positions ($\mathbf{q}_\ell$), illumination inside the explosion volume escapes the smoke layer around it and illuminates the explosion from outside, so it produces over-exposed illumination and incorrect self-shadows.

determining the effective distances of levels (instead of using blending functions), or generate shadows by treating lights at higher (coarser) levels as point lights (instead of using pre-filtered smoke data) lead to substantial amount of flickering. The importance of using illumination centers ($\mathbf{p}_\ell$) for lighting, as opposed to original grid vertex positions ($\mathbf{q}_\ell$), is shown in Figure 9. When the illumination centers are not considered, not only is the total illumination not properly reproduced, but also the lights at higher (coarser) levels can incorrectly illuminate the explosion from outside of the smoke volume. This is because the grid points that are outside of the smoke volume (but close enough to receive light intensity) escape the occlusion of the smoke layer surrounding the internal illumination. Therefore, using original grid vertex positions as light positions leads to highly over-exposed illumination, as shown in Figure 9.

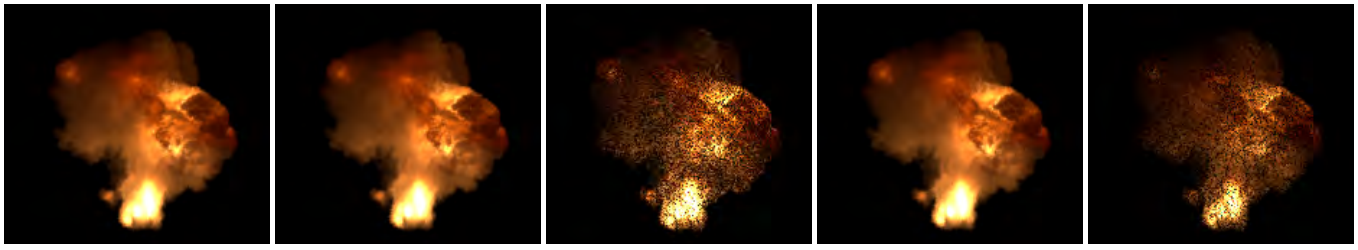
6.2 Comparisons to Lightcuts

Our approach is conceptually similar to the Lightcuts method [Walter et al. 2005], which uses a binary tree for clustering lights. However, the stochastic binary decisions used for building the light tree produce a different tree every time the algorithm is used, which is a source of flickering. The example in Figure 10 shows one frame of an explosion simulation rendered multiple times using Lightcuts with a 2% error threshold (as recommended). Notice that the difference (even without animation) is strong enough to cause visible flickering (Figure 10c). One could eliminate the stochastic decisions in light tree construction (by always picking the representative light with stronger intensity) (Figure 10d), but *deterministic* construction introduces bias (Figure 10e) and cannot guarantee temporal stability with animated illuminations.

The lighting computation of Lightcuts is based on a conservative error-bound that limits the maximum possible illumination that could come from a subtree. However, since hundreds of such subtrees are involved in lighting computation, the total error is not bounded by the error threshold and the temporal flickering can be substantial (though the total error estimation can be improved [Nabata et al. 2016]).

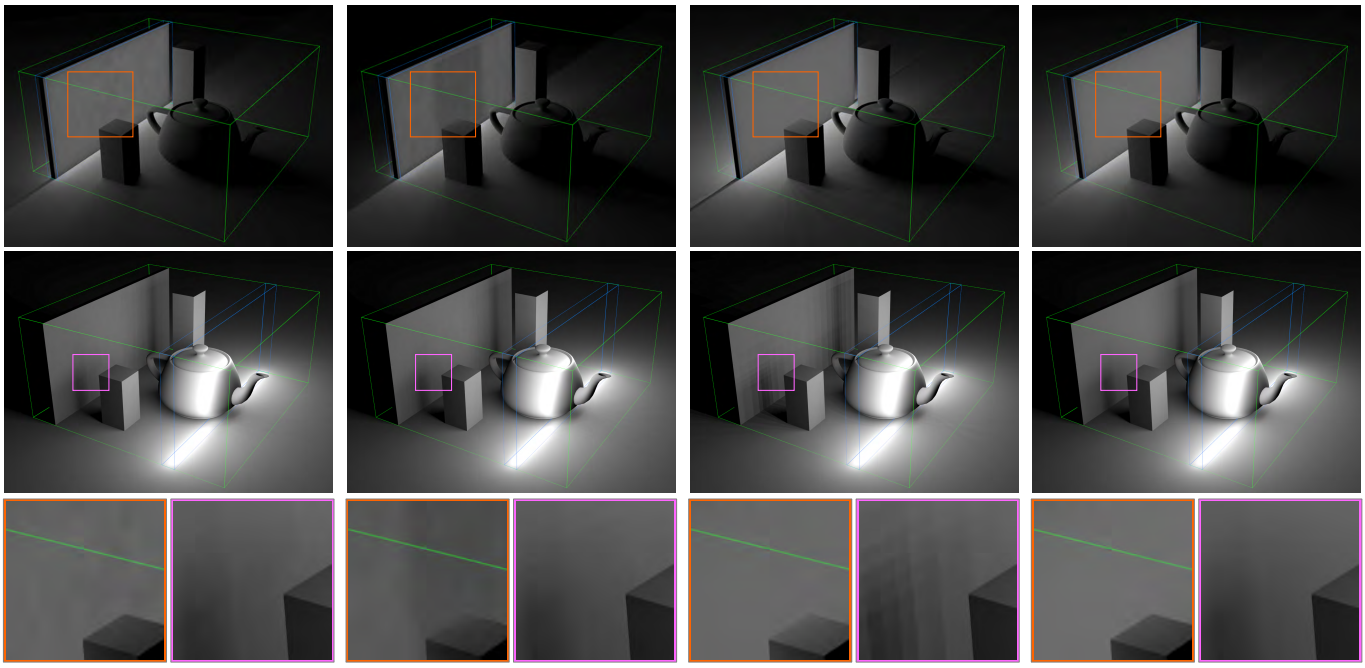
Most importantly, since Lightcuts and its variants do not limit the influence radii of lights, our shadow precomputation method is not applicable and the vast amount of high-resolution volume tracing for shadows leads to completely impractical render times. In fact, in our tests Lightcuts resulted in much slower render times than path tracing. The images in Figure 10 are generated using 100 times fewer primary camera rays than the images in Figure 1, but they rendered in about 2 hours, which is over an order of magnitude slower than our path tracing reference and *several orders of magnitude slower* than our method. This is because our path tracer directly uses the volume data (instead of generating many point lights), so it can accumulate light during ray marching, while the many-lights solution using Lightcuts merely accumulates shadows (via ray marching).

In comparison, our method builds a consistent structure every time (avoiding flickering) and allows efficient shadow precomputation (by limiting the light influence radii) for significantly accelerated rendering. However, if we skip shadow precomputation with our method and use ray marching instead, our method slows down to a similar impractical rendering speed (varies based on the α parameter) as Lightcuts. Therefore, the performance advantage of our method is directly tied to its ability to precompute volumetric shadows.



(a) Lightcuts (b) Lightcuts (rerun) (c) Difference (×4) (d) Deterministic Lightcuts (e) Deterministic bias (×4)

Fig. 10. One frame of an explosion rendered twice using Lightcuts producing (a-b) two different results, (c) their difference (×4), (d) Lightcuts with deterministic light tree construction, and (e) the bias introduced by deterministic tree construction shown as the difference (×4) between (d) and the average of 128 results with Lightcuts.



(a) Lightcuts (b) Deterministic Lightcuts (c) **Ours** (point light shadows) (d) **Ours** (volume light shadows)

Fig. 11. Opaque polygonal geometry inside the light volume (shown as green box), where only a narrow segment of the volume contains non-zero illumination (shown as blue box), computed using (a) Lightcuts, (b) Lightcuts with deterministic tree construction, (c) our method with point light shadows, and (d) our method with volume light shadows (9 shadow samples per light per pixel).

6.3 Opaque Shadows

We use ray traced shadows for handling opaque polygonal objects within or outside of the explosion volume, such as the example in Figure 11. Therefore, our method for handling opaque geometry does not benefit from the performance boost of precomputed shadows. However, since polygonal shadows are typically faster to compute as compared to ray marching through volume data, this is an acceptable solution.

For handling opaque polygonal geometry Lightcuts can produce faster results by sampling fewer lights than our method (depending on the parameters). On the other hand, Lightcuts suffers from temporal flickering and it can produce artifacts in soft shadows (Figure 11a). Modifying the light tree construction of Lightcuts with deterministic decisions (as described above) does not help with flickering or soft shadows, and the bias it introduces can lead to incorrect illumination, such as the middle of the wall in Figure 11b top row.

Using our method it is possible to precompute shadows for polygonal geometry as well. However, treating the lights at the higher (coarser) levels as points leads to objectionable hard shadows instead of soft shadows (Figure 11c), and it can produce incorrect results for distant illumination if the illumination centers happen to fall inside opaque geometry. Instead, we treat them as volume lights and generate shadows via Monte Carlo sampling using Equation 4 to produce the soft shadows in Figure 11d. While this produces high-quality shadows and alleviates light bleeding problems, like other scalable many-lights solutions, light bleeding cannot be entirely avoided.

Fig. 12. A scene illuminated by 227K animated balls. Both the balls and the letters are translucent volumes. The balls illuminate each other and cast shadows.

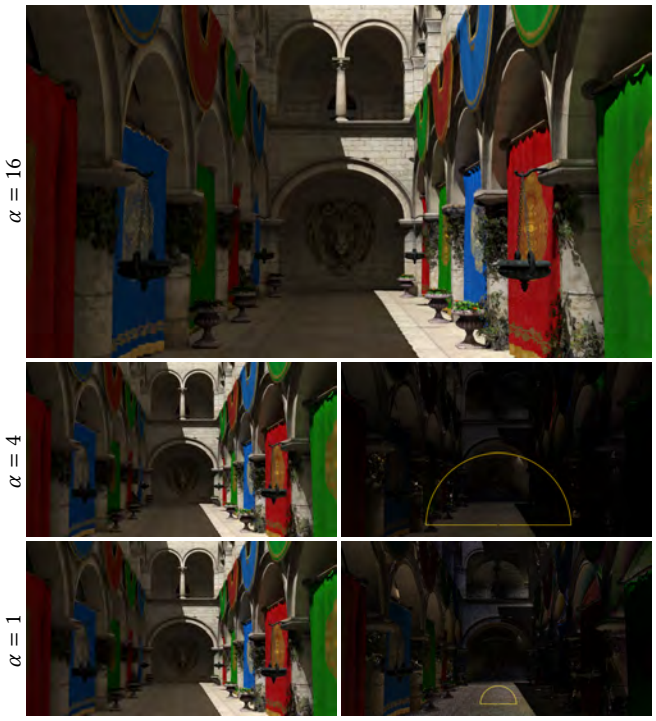


Fig. 13. Indirect illumination from 365K VPLs computed using our lighting grid hierarchy with α values 16, 4, and 1, sampling 43K, 4K, and 361 lights on average, respectively. Hemispheres with $2r_0$ radius that correspond to $\alpha = 1$ and $\alpha = 4$ are shown on the difference images, indicating the influence regions of lights in $\mathbb{S}_0$.

6.4 General Many-lights Problems

Besides explosions, lighting grid hierarchy can also be used for efficiently computing the illumination from a large number of animated point lights. Figure 12 shows such an example, where the scene is illuminated by animated balls. The letters and the balls are translucent and represented as volume data, so they illuminate each other and cast translucent shadows.

In fact, our method is oblivious to how the lights are generated. In Figure 13 we show a scene rendered using VPLs for indirect illumination, computed using our method with three different α parameter values: 1, 4, and 16. In this case, the lowest (finest) level grid separation (h_1) is automatically computed, starting with the bounding box of the VPLs with a desirable coarse resolution. Lower (finer) levels are generated until the number of grid vertices with non-zero illumination exceeds the number of VPLs. As the α value increases, our method samples more VPLs per shaded point, since the radius values used for the blending functions increase with α. Thus, when α is large enough, all VPLs are contained in the influence region of the lowest (finest) level, and our method converges to ground truth via (effectively) brute-force computation. Smaller α values permit approximating distant illumination using higher (coarser) levels, causing low-frequency error, as can be see in the difference images in Figure 13.

6.5 Performance

Table 1 includes the peak values for the number of light sources at all levels and the total shadow map sizes, along with the average computation times per frame for the explosion examples in Figures 5 and 8, and the animated balls in Figure 12. Notice that the number of lights rapidly decreases with increasing levels. The fastest component of our method is building the lighting grid hierarchy.

Table 1. Performance Results

	Figure 5	Figure 8	Figure 12
Voxel Count	~19M	~19M	~32M
Max. $\mathbb{S}_0$ light count	* 1,405K	700K	227K
Max. $\mathbb{S}_1$ light count	* 233K	114K	240K
Max. $\mathbb{S}_2$ light count	* 36K	19K	59K
Max. $\mathbb{S}_3$ light count	* 5.6K	3.2K	11K
Max. $\mathbb{S}_4$ light count	* 932	652	1.7K
Max. $\mathbb{S}_5$ light count	* 178	142	316
Max. $\mathbb{S}_6$ light count	* 46	46	–
Max. Total Shadow Size	905 MB	448 MB	683 MB
Max. $\mathbb{S}_0$ Shadow Size	426 MB	195 MB	67 MB
Max. $\mathbb{S}_1$ Shadow Size	202 MB	109 MB	185 MB
Max. $\mathbb{S}_2$ Shadow Size	111 MB	58 MB	178 MB
Max. $\mathbb{S}_3$ Shadow Size	68 MB	38 MB	124 MB
Max. $\mathbb{S}_4$ Shadow Size	45 MB	23 MB	78 MB
Max. $\mathbb{S}_5$ Shadow Size	34 MB	15 MB	51 MB
Max. $\mathbb{S}_6$ Shadow Size	19 MB	10 MB	–
Avrg. Precomputation	* 273 sec.	35 sec.	20 sec.
Avrg. Render Time	320 sec.	150 sec.	30 sec.

* Includes multiple scattering lights/computation.

Shadow generation takes most of the precomputation time, but it provides substantial savings during final rendering. Each depth level of the multiple scattering computation involves light generation by computing lighting through the entire simulation volume using the previously computed hierarchy, generating the hierarchy, and precomputing shadows.

7 CONCLUSION

We have introduced a scalable solution for computing illumination with many animated lights designed for rendering explosions. We have explained how we could avoid temporal flickering and how we could precompute volumetric shadows for providing orders of magnitude speed up. Furthermore, we have described an efficient method for incorporating multiple scattering. Our results show that our method is effective in rendering explosions with self-illumination and it can also be used for other many-lights problems.

However, our method has its limitations. First of all, it relies on a user-defined parameter α for determining its accuracy. While this may be a convenient knob for adjusting the performance and accuracy trade-off, we provide no formulation for bounding the error of our lighting approximation. Yet, even with relatively small α values our method generates consistent and noise-free results. Moreover, our multiple scattering computation only works for isotropic scattering. Even with single scattering, glossy materials may require relatively large α values. Furthermore, unlike path tracing solutions that can quickly produce a noisy preview, our method has a fixed cost and it directly produces the final result.

ACKNOWLEDGEMENTS

We thank David Hill for his help with preliminary tests, Ron Henderson and Andrew Pearce for their support for this work, and DreamWorks Animation for facilitating this project. We also thank Peter Shirley for his helpful feedback and Konstantin Shkurko for his comments. Finally, we thank anonymous reviewers for their valuable suggestions, particularly regarding extensive comparisons with Lightcuts.

REFERENCES

Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. 2014. Scalable Realistic Rendering with Many-Light Methods. *Computer Graphics Forum* 33, 1 (2014), 88–104.

Tomáš Davidovič, Iliyan Georgiev, and Philipp Slusallek. 2012. Progressive Lightcuts for GPU. In *ACM SIGGRAPH '12 Talks*. Article 1.

Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, Philipp Slusallek, and Kavita Bala. 2010. Combining Global and Local Virtual Lights for Detailed Glossy Illumination. *ACM Trans. Graph. (Proceedings of SIGGRAPH '10)* 29, 6, Article 143 (2010), 8 pages.

Zhao Dong, Thorsten Grosch, Tobias Ritschel, Jan Kautz, and Hans-Peter Seidel. 2009. Real-time Indirect Illumination with Clustered Visibility. In *Vision, Modeling, and Visualizaion Workshop 2009*. 187–196.

Thomas Engelhardt, Jan Novák, ThorstenW. Schmidt, and Carsten Dachsbacher. 2012. Approximate Bias Compensation for Rendering Scenes with Heterogeneous Participating Media. *Computer Graphics Forum* 31, 7 (2012), 2145–2154.

Bryan E. Feldman, James F. O'Brien, and Okan Arikan. 2003. Animating Suspended Particle Explosions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '03)* 22, 3 (2003), 708–715.

Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. 2002. Local Illumination Environments for Direct Lighting Acceleration. In *Proceedings of the 13th Eurographics Workshop on Rendering*. 7–14.

Roald Frederickx, Pieterjan Bartels, and Philip Dutré. 2015. Adaptive LightSlice for Virtual Ray Lights. In *Eurographics 2015 Short Papers*. 61–64.

Pascal Gautron, Cyril Delalandre, Jean-Eudes Marvie, and Pascal Lecocq. 2012. Volume-aware Extinction Mapping. In *ACM SIGGRAPH '12 Talks*. Article 31, 31 pages.

Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. 2009. Virtual Spherical Lights for Many-light Rendering of Glossy Scenes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia '09)* 28, 5, Article 143 (2009), 6 pages.

Miloš Hašan, Fabio Pellacini, and Kavita Bala. 2007. Matrix Row-column Sampling for the Many-light Problem. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '07)* 26, 3, Article 26 (2007), 10 pages.

Miloš Hašan, Edgar Velázquez-Armendariz, Fabio Pellacini, and Kavita Bala. 2008. Tensor Clustering for Rendering Many-light Animations. In *Proceedings of Eurographics Workshop on Rendering*. 1105–1114.

Yuchi Huo, Rui Wang, Tianlei Hu, Wei Hua, and Hujun Bao. 2016. Adaptive Matrix Column Sampling and Completion for Rendering Participating Media. *ACM Transactions on Graphics* 35, 6, Article 167 (2016), 11 pages.

Kosuke Nabata, Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. 2016. An Error Estimation Framework for Many-Light Rendering. *Computer Graphics Forum* 35, 7 (2016), 431–439.

Yuchi Huo, Rui Wang, Shihao Jin, Xinguo Liu, and Hujun Bao. 2015. A Matrix Sampling-and-recovery Approach for Many-lights Rendering. *ACM Transactions on Graphics* 34, 6, Article 210 (2015), 12 pages.

Genichi Kawada and Takashi Kanai. 2011. Procedural Fluid Modeling of Explosion Phenomena Based on Physical Properties. In *Proceedings of Symposium on Computer Animation*. 167–176.

Alexander Keller. 1997. Instant Radiosity. In *Proceedings of ACM SIGGRAPH '97*. 49–56.

Thomas Kollig and Alexander Keller. 2006. *Illumination in the Presence of Weak Singularities*. 245–257.

Nipun Kwatra, Jón T. Grétarsson, and Ronald Fedkiw. 2010. Practical Animation of Compressible Flow for Shock Waves and Related Phenomena. In *Proceedings of Symposium on Computer Animation*. 207–215.

Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. 2007. Incremental Instant Radiosity for Real-time Indirect Illumination. In *Proceedings of Eurographics Workshop Rendering*. 277–286.

Tom Lokovic and Eric Veach. 2000. Deep Shadow Maps. In *Proceedings of ACM SIGGRAPH '00*. 385–392.

Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. 2012. Progressive Virtual Beam Lights. *Computer Graphics Forum* 31, 4 (2012), 1407–1413.

Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. 2012. Virtual Ray Lights for Rendering Scenes with Participating Media. *ACM Transactions on Graphics* 31, 4, Article 60 (2012), 11 pages.

Jiawei Ou and Fabio Pellacini. 2011. LightSlice: Matrix Slice Sampling for the Many-lights Problem. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia '11)* 30, 6, Article 179 (2011), 8 pages.

Eric Paquette, Pierre Poulin, and George Drettakis. 1998. A Light Hierarchy for Fast Rendering of Scenes with Many Lights. *Computer Graphics Forum* 17, 3 (1998), 63–74.

Matthias Raab, Daniel Seibert, and Alexander Keller. 2008. *Monte Carlo and QuasiMonte Carlo Methods 2006*. Chapter Unbiased Global Illumination with Participating Media, 591605.

Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. 2008. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia '08)* 27, 5, Article 129 (2008), 8 pages.

Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. 2005. A Vortex Particle Method for Smoke, Water and Explosions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '05)* 24, 3 (2005), 910–914.

Peter Shirley, Changyaw Wang, and Kurt Zimmerman. 1996. Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics* 15, 1 (1996), 1–36.

Ingo Wald, Carsten Benthin, and Philipp Slusallek. 2003. Interactive Global Illumination in Complex and Highly Occluded Environments. In *Proceedings of Eurographics Workshop on Rendering*. 74–81.

Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. 2002. Interactive Global Illumination Using Fast Ray Tracing. In *Proceedings of Eurographics Workshop Rendering*. 15–24.

Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. 2006. Multidimensional Lightcuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06)* 25, 3 (2006), 1081–1088.

Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. 2005. Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '05)* 24, 3 (2005), 1098–1107.

Bruce Walter, Pramook Khungurn, and Kavita Bala. 2012. Bidirectional Lightcuts. *ACM Transactions on Graphics* 31, 4, Article 59 (2012), 11 pages.

Rui Wang, Yuchi Huo, Yazhen Yuan, Kun Zhou, Wei Hua, and Hujun Bao. 2013. GPU-based Out-of-core Many-lights Rendering. *ACM Transactions on Graphics* 32, 6, Article 210 (2013), 10 pages.

GregoryJ. Ward. 1994. Adaptive Shadow Testing for Ray Tracing. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*. 11–20.