LightGraph: Efficient Multiple Scattering in Participating Media using Shortest Path Finding

Alexandre Sirois-Vigneux
alexandre.sirois-vigneux@umontreal.ca
Université de Montréal

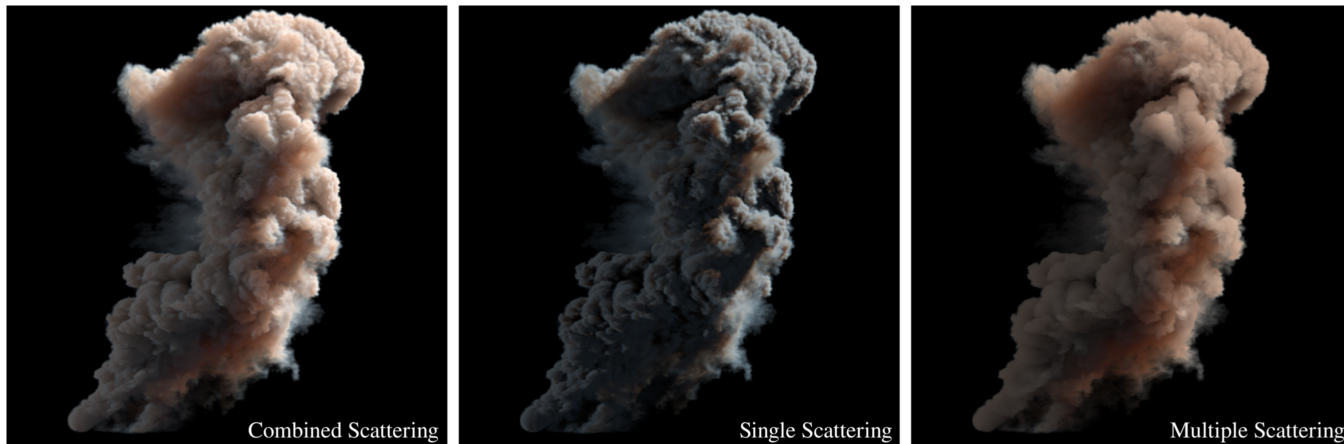

Figure 1: Render of multiple scattering using the technique presented in this paper. This shows from left to right: the final render, the single scattering contribution and the multiple scattering contribution.

ABSTRACT

We present an efficient way of estimating multiple scattering in discrete high resolution heterogeneous participating media. Our approach is based on a stochastically generated graph which estimates how light propagates through the volume using shortest path finding. This new method provides a way of achieving high quality photorealistic multiple scattering effect at a fraction of the computational cost of commonly used techniques in visual effects nowadays. The goal is not to be physically accurate nor is it to run in real-time, but to be a fast and reliable solution to allow quick turnarounds from a practical standpoint. Our method produce results that looks physically correct, work consistently across multiple cases containing animation and provide a minimal number of adjustable parameters while maximizing flexibility.

KEYWORDS

global illumination, participating media, multiple scattering, shortest path finding, precomputation, openvdb

1 INTRODUCTION

Computing multiple scattering in nonhomogeneous participating media has always been a challenging problem in computer graphics. The first successful approaches relied on Monte Carlo path tracing with the work of Kajiya and Von Herzen [6], but those suffered from very expensive render times. From that point, many papers were published over the years trying to improve upon this technique using bidirectional path tracing with Lafortune and Willems [9] or metropolis path tracing with Pauly et al. [13]. Those solutions, although physically correct and unbiased, still converge at rates that might not be practical in a production setting. Other approaches have tried to approximate the complex phenomenon with classical diffusion approximation (CDA) with the work from Stam [17] while others have been relying on density estimation like Jensen with photon mapping [4]. A fair number of researchers have also tried to achieve real-time visualization of multiple scattering such as the work from Kun Zhou et al. [21]. Those real-time techniques tend to impose too many constraints in regards to the volume resolution and animation, preventing them from having any practicle application in visual effects.

The current trend in visual effects production still gravitates around path tracers like Arnold because of how straight forward and predictable those system are. Large render farms owned by big studios have made it possible for those expensive techniques to be utilized in practice for the past few years. The industry has lean toward simpler techniques that use more machine hours and fewer man hours claiming that was the most effective tradeoff production-wise. The visual effects industry is now rapidly changing to a model where tasks that use to take months are now expected within weeks. In this new reality, clients reviews are often done twice a day and studios are expected to address notes in a few hours which makes a 32 hours render completely impractical regardless of the sheer amount of parallel farm power that even the biggest visual effects houses might have.

As a way to compare results and ultimately track progress in the field, most researchers evaluate the quality of their work by comparing it to the ground truth. Although this makes sense in the context of synthesizing the world around us in a physically accurate way, it turns out that none of this really matters when it comes down to creating the next Avenger film where the metric of choice is: "Does it look cool?" which is a lot more subjective and inaccurate but still precisely captures the main concerns of visual effects studios. Very often, actual footage shot in camera will be altered to better suit the creative vision of the director, which again stresses the importance of not using physical accuracy as our only way of evaluating the value of a method.

We propose a solution to estimate the effect of multiple scattering using shortest path finding (SPF). Single scattering events are handled with traditional ray marching while multiple scattering events are approximated by performing density estimation on a point cloud that corresponds to the vertices of a graph. The idea is to stochastically define multiple graphs with a few hundreds vertices in the volume, then ray march the edges to evaluate their transmittance according to the user provided density grid and use those transmittance value as weights to find the least occluded path between each pair of vertices with shortest path finding. The process can be repeated multiple times in parallel and rasterized down to a volumetric grid. The multiple scattering grid can then be treated as an emission component by the ray marcher at render time which makes it very efficient. Even though this technique is intrinsically biased, results empirically demonstrate that the technique could still offer a practical solution in visual effects thanks to its performance advantage over the alternative techniques currently used in production.

2 PREVIOUS WORK

B. Wang and N. Holzschuch present a technique to precompute the multiple scattering events, assuming an infinite medium, and store it in two 4D tables [19]. Their technique can be used with multiple rendering algorithms and is able to speed up convergence significantly. Unfortunately, it only applies to homogeneous participating media which makes it unusable to render things like clouds, smoke or any simulated fluids.

Szirmay-Kalos et al. propose a real-time method to compute multiple scattering in nonhomogeneous participating media having general phase functions [18]. Their technique is based on a particle system assuming that the volume is static while the light and camera can still be animated. They store the radiance interaction between particles in what they call an illumination network. Their technique allow the illumination to be computed in real-time, but the result is quite low resolution and could not be used in a visual effects or animation. Also, the fact that this technique does not support animated volumes restricts its usage considerably.

D. Koerner et al. propose a method called flux-limited diffusion for the rendering of multiple scattering effects in participating media [8]. Their technique offers an improvement over popular methods based on diffusion approximation especially for transparent regions which lead to a better match of the correct light transport. Their technique is applicable to heterogeneous media, where opaque material is embedded within transparent regions. Although this technique should generalize to more complex lighting configuration, we don't know how it would behave under environment lighting sampled from an high-dynamic-range (HDR) map.

Kallweit et al. present a technique for efficiently synthesizing images of atmospheric clouds using a combination of Monte Carlo integration and neural networks [7]. Instead of simulating all light transport during rendering, they pre-learn the spatial and directional distribution of radiant flux from multiple cloud examples rendered with Monte Carlo simulations. Their solution efficiently predicts the radiance at any point in the cloud and can therefore synthesize images of clouds that are nearly indistinguishable from the reference solution in a very short amount of time. One limitation of their system is that it's highly specialized toward cloud rendering and does not directly generalize to arbitrary heterogeneous participating media, limiting its application.

Jensen et al. present an extension of photon mapping for computing global illumination in scenes with participating media [4]. The method is based on bidirectional Monte Carlo ray tracing and uses photon maps to increase efficiency and reduce noise. Effects such as multiple scattering are easily reproduced by this technique. The large number of point cloud lookups executed as the volume is ray marched makes the render time suboptimal even when using an acceleration structure such as a balanced Kd-Tree. It is also not clear how well this technique would handle emissive volume like fire.

3 LIGHT TRANSPORT IN PARTICIPATING MEDIA

Note that the technique presented in this paper makes several assumptions in regard to the theory presented in this section. We are interested in calculating the incoming radiance from a given direction ω. If we consider a photon travelling through a volume towards the camera along ω, this photon can either be absorbed by the participating media or scattered in a new direction meaning that the ray will never hit the camera sensor. Those phenomena are called *absorption* and *out-scattering* and are denoted by their respective coefficients σ_a and σ_s. The scattering coefficient σ_s can also be wavelength dependant which leads to color shifts as the light scatters through the media. Photons travelling in an arbitrary direction could also be scattered in the direction ω and therefore land on the sensor, this is called *in-scattering* also denoted by σ_s. The probability distribution for scattering events is called a phase function [20] denoted by $p(x, \omega_i, \omega)$ where x is a point in the volume and ω_i is the direction of incoming radiance. The last component that needs to be considered is the emission of light that occurs in volumes such as fire which is denoted by L_e. The change in radiance at point x in direction ω can be expressed as the differential equation

$$\begin{aligned}\frac{\partial}{\partial x}L(x,\omega) &= L_i(x,\omega) - L_o(x,\omega)\\ &= \sigma_a L_e(x,\omega) + \sigma_s L_i(x,\omega) - \sigma_s L(x,\omega) - \sigma_a L(x,\omega)\end{aligned} \tag{1}$$

where L_i is the incoming radiance at point x and L_o is the resulting radiance at point x. We will be referring to the *out-scattering* and *absorption* together as the *extinction* coefficient denoted by σ_e such that

$$\sigma_e = \sigma_a + \sigma_s. \tag{2}$$

Another value that needs to be introduced is the albedo of the media

$$\rho = \frac{\sigma_s}{\sigma_e} \tag{3}$$

which defines how dark or bright the media is. In order to solve for the transmittance of an infinite homogeneous media over a distance t we can solve the following ordinary differential equation

$$dL(t) = -\sigma_e L(t)dt. \tag{4}$$

By re-arranging the terms and integrating both sides we get

$$\frac{L(t)}{L(0)} = e^{-\sigma_e t}. \tag{5}$$

This relationship is called Beer's law where transmittance is denoted by

$$T(x + \omega t, x) = e^{-\sigma_e t}. \tag{6}$$

This law still applies to the general case of heterogeneous participating media which is the one we are interested in. This yields the following formulation

$$T(x', x) = e^{\int_{x'}^{x} \sigma_e(t)dt} \tag{7}$$

where $x' = x + \omega t$. The incoming radiance $L_i(x, \omega)$ can be expressed as the integral

$$L_i(x, \omega) = \int_{\Omega} p(x, \omega', \omega)L(x, \omega')d\omega' \tag{8}$$

where Ω is the unit sphere centred in x and the irradiance from direction ω' is weighted by the normalize phase function p. If we use equation 2 and 8 inside equation 1 we get the integro-differential equation [4]

$$\begin{aligned}\frac{\partial}{\partial x}L(x, \omega) &= \sigma_a(x)L_e(x, \omega) \\ &+ \sigma_s(x) \int_{\Omega} p(x, \omega', \omega)L(x, \omega')d\omega' \\ &- \sigma_e(x)L(x, \omega).\end{aligned} \tag{9}$$

Integrating both sides of the equation along a straight path from x_0 to x (in direction ω) gives the following integral equation

$$\begin{aligned}L(x, \omega) = &\int_{x_0}^{x} T(x', x)\sigma_a(x')L_e(x', \omega)dx' \\ &+ \int_{x_0}^{x} T(x', x)\sigma_s(x') \int_{\Omega} p(x', \omega', \omega)L(x', \omega')d\omega' dx' \\ &+ T(x_0, x)L(x_0, \omega)\end{aligned} \tag{10}$$

which further simplifies to

$$L(x, \omega) = \int_{x_0}^{x} T(x', x) \left(L_e(x') + \frac{1}{4\pi} \int_{\Omega} L(x', \omega')d\omega' \right) dx' \tag{11}$$

if we drop the boundary condition term and assume that all participating media should be treated as isotropic, which is what we will do in this paper. Multiple scattering can be calculated using equation 11 by recursive Monte Carlo integration where sampling alternates between distances and directions, gradually building a path toward the boundary and evaluating the radiance [7]. For the sake of completeness, the phase functions $p(x, \omega', \omega)$ gives the probability that an incident photon is deflected by an angle θ between ω' and ω. A convenient model for the phase function is the Henyey-Greenstein function defined as

$$p(\theta) = \frac{1 - g^2}{(1 + g^2 - 2gcos\theta)^{3/2}} \tag{12}$$

where the dimensionless parameter $g < 1$ models the anisotropy of the scattering [2]. In visual effects production, most volumes are treated as isotropic even if some of them like clouds have a very particular phase function that should not be ignored when pursuing photorealistic images.

4 METHOD OVERVIEW

Our method tries to leverage the low-frequency nature of multiple scattering in participating media by approximating the solution with very few samples that are then rasterized into a sparse low-resolution grid to ensure fast lookup at render time. Multiple scattering is therefore precomputed and stored in memory before the render starts. Looking at equation 11, this multiple scattering grid (MSG) will be sampled similarly to the emission in the L_e term.

Figure 2: Render of bunny_cloud.vdb from https://www.openvdb.org/download/ using an environment light with a HDR map of a sunset. Using our method, multiple scattering is not more expensive to compute with this configuration than with pure directional lighting.

One particularity of our method is that we use SPF in order to estimate how much light is travelling between two points in the media instead of relying of random walks. This adds bias to our

solution, but also significantly speeds up its convergence as most paths have a meaningful contribution to the solution.

Here are some of the main features of LightGraph. It supports scene description provided as a chain of arguments through the command line. It supports multiple lights per scene including point lights, directional lights and environment lights with HDR map and proper importance sampling as shown in figure 2. Each rendered image is written to disk as an EXR image containing 3 Arbitrary Output Variables (AOV) that respectively isolates the contribution of single scattering, multiple scattering and emission. There is also an option to output the LightGraph geometry (created on the fly to compute multiple scattering) as ASCII files for debug or visualization purpose. It's also possible to send multiple camera rays in order to reduce aliasing. The built-in shader handles temperature grid and has a color ramp parameter that allows the user to remap temperature values to emission colors for volume such as fire or explosion as shown in figure 12. See the GitHub[1] page for more information about the implementation of LightGraph.

5 IMPLEMENTATION

Our method is implemented in C++ and is fully multithreaded using TBB from Intel [14]. We also tried to use OpenMP but this turned out to be much slower in terms of runtime performance. Multiple libraries have been used to facilitate implementation as well as to stick with some well establish standards in the visual effects industry. Those are discussed in greater details in the data structure subsection.

5.1 Shortest Path Finding as an Importance Sampling Heuristic

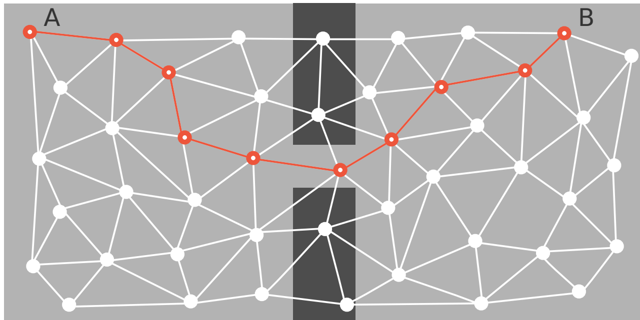

Figure 3: The background colors represent a 2D density grid where dark values are associated with denser regions. On top in white, we have a representation of a single 2D LightGraph. The path in orange shows how shortest path finding can efficiently find the least occluded path between point A and B. As long as at least one edge per graph passes through this bottle neck, every iteration will return a similar transmittance value associated with the (A,B) pair. This yields a lot less variance than relying on random walks to build the paths.

[1]LightGraph GitHub project: https://github.com/asiroisvigneux/LightGraph

Multiple Importance Sampling (MIS) is ultimately what has made Monte Carlo simulation a practical solution for light transport. It has the theoretical advantage of being unbiased, but even the state of the art still struggles to converge in a reasonable amount of time especially when multiple scattering is involved. Our approach has no theoretical guaranteed to converge to the correct solution, but it is empirically demonstrated to converge to a solution that appears visually correct in the *Result* section.

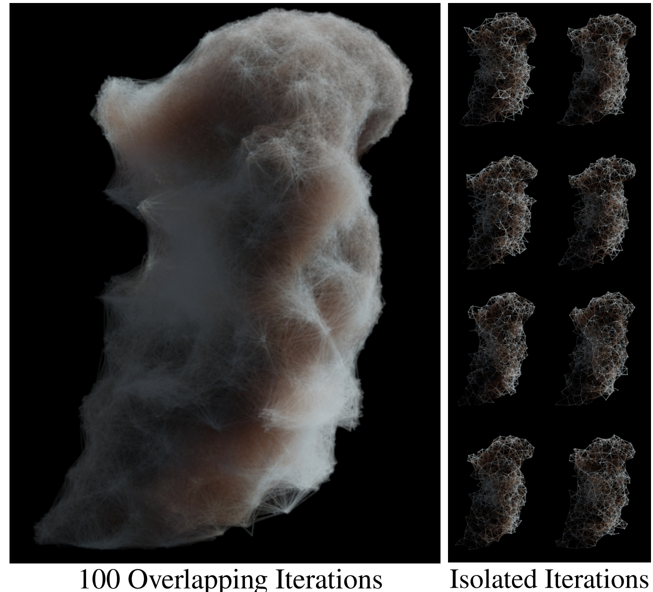

Figure 4: On the left we have a visualization of the geometry of 100 LightGraph iterations overlapping each other. On the right we have 8 isolated iterations out of the 100. The vertices of the graphs are colored by their value after *Radiance Diffusion*. The whole process takes under 8 secondes to be computed on the hardware described in the *Results* section.

Our method tries to avoid wasting time on paths that won't have a significant contribution to multiple scattering like described in figure 3. By using SPF, we force our solution to only sample paths of high radiance transfer. A strong prior of our method is that we assume that only considering the most valuable path between each pair of vertices in the volume is enough to properly estimate multiple scattering. We therefore focus our efforts in sampling what will have the most impact in terms of perceivable information. As it will be discussed later on, this heuristic is a lot more expensive to compute than MIS path tracing, but it's also a lot less random. Since the underlying graph that performs multiple scattering estimation is pretty coarse, it is very likely that the solution returned from a single LightGraph iteration will contain significant low-frequency variance. As we aggregate more of those iterations, see figure 4, this variance is expected to decrease until it's not perceivable as described in figure 7 in the *Rate of Convergence* section. The question of bias regarding this estimator will also be discussed in the *Results* section.

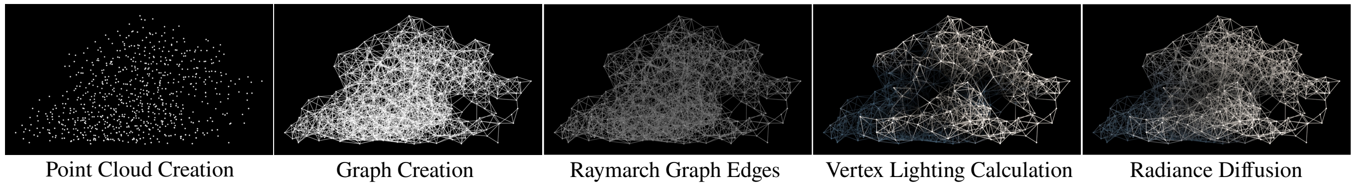

Figure 5: Visualization of the main steps of a single LightGraph iteration. The program has the option to output geometric information as ASCII file for debug or visualization purpose.

5.2 Data Structures

Our method is implemented using the OpenVDB library developed at DreamWorks Animation. OpenVDB is an Academy Award-winning open-source C++ library comprising a novel hierarchical data structure and a suite of tools for the efficient storage and manipulation of sparse volumetric data discretized on three-dimensional grids [12]. This format has become the standard to store volumetric data in the visual effects industry which makes it particularly well adapted to our use case. OpenVDB also provide a basic implementation of a simple ray marcher with all the utility function to efficiently perform intersection testing against their sparse data structure. This code was used as a base for our own implementation. OpenVDB depends on multiple external libraries including OpenEXR, developed by Industrial Light & Magic, which is used to store the rendered image in high-dynamic-range half floating-point format [5].

5.3 Algorithm Overview

Algorithm 1: Rendering with LightGraph

```
Result: ExrImage
Load VdbGrids from disk;
for iter ← 1 to maxIter do
    Point Cloud Creation;
    Graph Creation;
    Ray March Graph Edges;
    Shortest Path Finding Solve;
    Vertex Lighting Calculation;
    Radiance Diffusion;
end
Aggregate vertex from all iterations to a sparse regular grid;
Rasterize Multiple Scattering to a volumetric ScatterGrid;
foreach pixel i, j ∈ ExrImage do
    ExrImage[i, j] ← Ray March(VdbGrids, ScatterGrid);
end
```

5.4 Multiple Scattering Estimation

Point Cloud Creation: We start by scattering points inside the volume with dart throwing to maximize the quality of our estimate using as little points as possible [10]. We have determined empirically that 750 points per LightGraph iteration was a good balance between resolution in the graph and runtime performance. Points are only scattered in voxels containing density. Each point is added at a distance of at least δ to all other points to ensure an even distribution inside the density grid.

The user does not have to do anything special to enforce this constraint of 750 points since the system will dynamically adjust δ as dart throwing is performed. A first estimate of δ is made based on the size of the bounding box in order to be scale invariant. Then, a predefined polynomial function estimates the rate at which the points are expected to accumulate in the point cloud. A comparison with this expected value is made every 10000 points thrown, followed by an adjustment of δ. δ is increased to slow down the accumulation of points as it will force them to be further apart from each other. The opposite behaviour is obtained by decreasing δ. In total one million points are thrown in the density grid bounding box per LightGraph iteration which means that δ is adjusted at most 99 times per iteration. There is an early stopping mechanism that terminates dart throwing if the number of accumulated points exceeds 750. From our tests, early stopping is rarely triggered and, when it is used, the iteration is always about to finish which is the expected behaviour. The quality of the point cloud distribution would most likely be compromised if an iteration was to early stop after just a few dart throw. The main reason why the point cloud needs to remain small is the complexity of the shortest path finding (SPF) algorithm which is being used in the subsequent steps. Each iteration modifies its own version of δ to enable thread-safe read/write access as each iteration runs in parallel on separate threads. A separate random number generator instance is also created per iteration to ensure deterministic results of the algorithm.

Graph Creation: We then build a boolean adjacency matrix to store the edges between the scattered points. A maximum of 12 connections with neighbouring points is allowed within a maximum radius of $1.7 \times \delta$.

Ray march Graph Edges: Each edges of the graph are then ray marched against the density grid of the input grid to compute the transmittance of each segment. Those precomputed transmittance values are then stored in another matrix of floating point values. A scalar parameter is made available to the user to adjust the density values separately for multiple scattering calculation. It is often useful to decouple the density used for single scattering from the density used for multiple scattering as seen in figure 17 in the *Results* section. Other studies have also concluded that this density decoupling would yield better fit to the Monte Carlo simulation in denser media [3].

Shortest Path Finding Solve: Next we find the path with the highest transmittance between each pair of vertices by running a modified version of the Floyd-Warshall algorithm [1]. The solve is performed per channel, such that if the volume being rendered

had wavelength dependant scattering value, it could in theory find separate optimal paths for each RGB components. This step will determine the path of highest transmittance by considering all possible paths between each pair of vertices in the current graph. The resulting transmittance is stored in a symmetric weight matrix after the SPF solve. The paths themselves are never stored.

Vertex Lighting Calculation: The radiance received at each vertex in the graph is calculated and stored in an array by ray marching the density grid with shadow rays towards each light source in the scene.

Radiance Diffusion: Before performing the radiance diffusion, we adjust the weight matrix computed in the SPF step to enforce energy conservation. We make sure that each vertex will not diffuse more energy than what it has received at the *Vertex Lighting Calculation* step. This is done by constraining the sum of all transmittance from one point to all other points of the graph to sum to one. This way, the graph will still contain the same total radiance before and after diffusion. We then perform radiance diffusion where each vertex of the graph will gather radiance from all other points according to the normalized weights.

Aggregate Points to Sparse Regular Grid: Once all LightGraph iterations are computed, points are then aggregated in a sparse regular grid to ensure fast lookup query. To achieve optimal performance, the resolution of the grid is dynamically adjusted to store approximately 8 points per voxel as recommended by OpenVDB [11]. This step is crucial to the technique as we might be dealing with a large number of points. The quality of this acceleration structure will have a big impact on performance in the following step where we rasterize the gathered radiance to a volumetric grid.

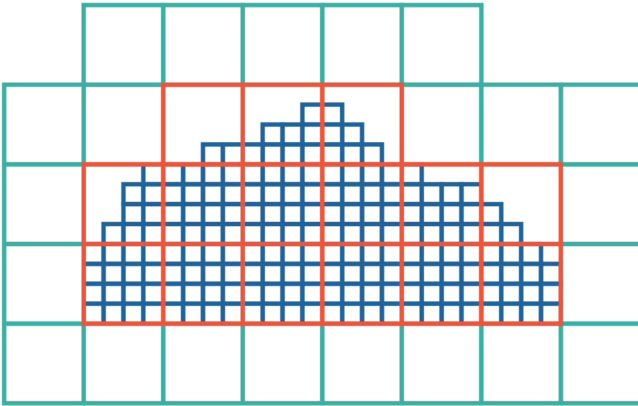

Figure 6: A 2D representation of the topology of the density grid in blue with, in orange, the multiple scattering grid (MSG) topology at 1/4 the resolution of the density grid without additional padding. The union of the green and orange grid shows the MSG after voxel padding to prevent masking artefacts at render time.

Rasterize Multiple Scattering to a Volumetric *Grid*: A multiple scattering grid (MSG) is created using the same sparse tree as the density grid, but at 4 times the original voxel size by default effectively creating a volume with a voxel count of $(\frac{1}{4})^3 = \frac{1}{64}$ of the provided density grid. This drop in resolution will maintain a low memory footprint, while still providing plenty of resolution to properly capture multiple scattering which is intrinsically low-frequency. This MSG prevents the algorithm from executing point cloud query at each sample while ray marching the grids at render time. Since the data structure is sparse, we activate a padding of one voxel surrounding every active voxel (for a maximum of $3^3 - 1 = 26$ voxels) in the down-scaled MSG to prevent artefacts due to the limited coverage of the MSG over the density grid, see figure 6. This coverage is important as we will use the density grid as a mask to the MSG at render time. We then iterate over each active voxel of the MSG and perform density estimation with an isotropic Gaussian kernel [16]

$$\widehat{MS}(x) = \sum_{i=1}^{N} y_i \frac{\kappa(x, x_i)}{\sum_{j=1}^{N} \kappa(x, x_j)} \tag{13}$$

where $\widehat{MS}$ is the estimated multiple scattering at voxel position x, $\kappa(\cdot)$ is the kernel with parameters $\mu = x$ and σ which defines the weights of all considered neighbouring point positions x_i.

The parameter σ^2 is derived from the *Points per Kernel* user parameter, let's call it k. We first need to define the maximum distance to look for other points in the point cloud. We know that the data structure holds an average of 8 points per voxel. Let's set γ to be the width of a voxel, we therefore have that

$$\gamma^3 = V \tag{14}$$

where V is a volume that is expected to contain 8 points. We can then multiply V by the ratio $\frac{k}{8}$ and solve for our maximum search radius r such that

$$\begin{aligned} \frac{4}{3}\pi r^3 &= V\frac{k}{8} \\ \frac{4}{3}\pi r^3 &= \gamma^3\frac{k}{8} \\ r &= \sqrt[3]{\frac{3k\gamma^3}{32\pi}} \end{aligned} \tag{15}$$

The expression of the Gaussian kernel in n dimensions can be simplified for our specific case to

$$\begin{aligned} \kappa(x_i; \mu, \Sigma) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x_i - \mu)^\top \Sigma^{-1}(x_i - \mu)\right) \\ &\propto \exp\left(-\frac{1}{2}(x_i - \mu)^\top \Sigma^{-1}(x_i - \mu)\right) \\ &= \exp\left(-\frac{1}{2\sigma^2}||x_i - \mu||^2\right) \end{aligned} \tag{16}$$

where $\Sigma = \sigma^2\mathbf{I}$ since the kernel is isotropic. We use a truncated Gaussian kernel and only consider points with weights greater than an empirically determined threshold of 0.1. We can now solve for σ^2 using this information.

$$\begin{aligned} \exp\left(-\frac{1}{2\sigma^2}||x_i - \mu||^2\right) &\geq 0.1 \\ -\frac{1}{2\sigma^2}r^2 &= \ln 0.1 \\ \sigma^2 &= -\frac{r^2}{2\ln 0.1} \end{aligned} \tag{17}$$

Table 1: Runtime of a LightGraph Iteration with Render. The multiple scattering computation is far from being the most expensive part of the algorithm considering a single iteration. As we increase the number of LightGraph iterations the computational cost will increase as well, but it should never take more than a minute to converge in virtually all possible scenarios.

LightGraph Step	Runtime in Seconds	Ratio
Point Cloud Creation	0.641	6.88%
Graph Creation	0.001	0.01%
Ray march Graph Edges	0.013	0.13%
Shortest Path Finding Solve	0.039	0.41%
Vertex Lighting Calculation	0.021	0.22%
Radiance Diffusion	0.011	0.11%
Rasterize Multiple Scattering	1.073	11.53%
LightGraph Iteration in Total	1.799	19.34%
Render Image	7.506	80.66%

5.5 Per Pixel Raymarching

At this point, we now have the multiple scattering stored in the MSG. We can therefore trilinearly interpolate it at any point inside the volume in constant time. It is therefore possible to use existing techniques from ray tracing to ray march the volume treating the multiple scattering as a simple emission grid.

In order to avoid artefacts that would occur from the resolution mismatch between the density and the MSG, we use the density grid as a mask over the MSG. For this mask, the density grid is clamped to the $[0, 1]$ range before being raised to a power of 0.25 by default which gives the best fit when compared to Monte Carlo simulations.

The image is then ray marched using traditional techniques to solve single scattering in participating media. Each primary ray samples the temperature grid (if stored in the input VDB file) as well as the MSG. Shadow rays are also cast from the primary volume samples and ray marched towards each contributing light of the scene. The temperature values are remapped to an RGB color derived from the color ramp parameter part of the built-in shader. When all pixels are done rendering, the RBGA value of each pixel along with the 3 additional AOVs are saved to disk as an half float EXR image.

5.6 Complexity Analysis

Most steps in the creation of the MSG involve an algorithmic complexity of $O(n^2)$ where n is the number of points in the point cloud generated in the first step. Unfortunately, the shortest path finding step (SPF) runs in $O(n^3)$ which breaks down as soon as the number of points reaches a certain amount. To work around this limitation we split the computation into λ separate LightGraph iterations that are then aggregated. Since we force $n \longrightarrow 750$, the SPF as well as all the other steps inside the iteration become very predicable in terms of runtime and can then be seen as a constant computation time. The whole LightGraph loop then runs in $O(\lambda n^3)$ which is a much more viable solution than $O((\lambda n)^3)$. This separation also makes the algorithm trivially parallelizable. The computation time is further reduced by rasterizing the multiple scattering estimate to the MSG effectively taking the algorithmic complexity of the rendering part from $O(ps(\log q + s))$ to $O(ps^2)$ where p is the number of pixels in the image, s is an upper bound on the number of camera / shadow samples per path and q is the number of points aggregated from all LightGraph iterations. Once computed the MSG effectively provides this information in a constant time. As we can see in table 1 the MSG creation is far from being the most costly part of the algorithm considering a single iteration. In terms of memory footprint, our method is in $O(\frac{1}{64}v + 750\lambda) = O(v + \lambda)$ where v is the number of voxels in the input density grid. On most modern machine, the additional memory usage should not be noticeable even with large λ values.

5.7 Advantage of Precomputations

One key advantage of this method is that the noise generated from the approximation is not perceivable unless the volume or the light is animated. The multiple scattering estimation can be very biased but still be visually believable since the noise is very low frequency compared to the noise we usually get with Monte Carlo integration. If the camera is moving around the volume, only the single scattering is recalculated to a new solution since the multiple scattering solution will always be the same regardless of the camera. If the light is changing, the underlying LightGraph iterations are still going to stay the same from frame to frame. Only *Vertex Lighting Calculation* and Radiance Diffusion will yield different results. The fact that the underlying structure is not perturbed makes the solution highly stable to light changes even with a small number of iterations as demonstrated with the cloud scene in figure 9 from the *Results* section. Finally, if the volume is animated, everything will change from frame to frame which might require more iterations to achieve temporal stability across the sequences.

5.8 Rate of Convergence

The number of points per kernel k as well as the number of LightGraph iterations λ are the two user parameters responsible for how stable the multiple scattering solution will be. In this section we focus on how those parameters influence the stability, runtime performance and level of detail of the solution in order to develop an intuition about what controls the quality of the estimate. To evaluate the rate of convergence of our method we use the Root Mean Square Error (RMSE) metric defined as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{18}$$

where x_i is the pixel value tested against the converged pixel value μ. From figure 7, we are able to extract an order of convergence of 0.135 which is quite bad in general, but in our case the orange line in the figure represents the approximate threshold where the error becomes unperceivable to the eye which happens after ≈ 35 secondes of computation in most cases.

Looking at figure 8 we observe that we can increase λ from 1 to 10 without paying any significant additional cost, thanks to our parallel implementation. As soon as we exceed the number of available threads on the machine, performance starts to decrease.

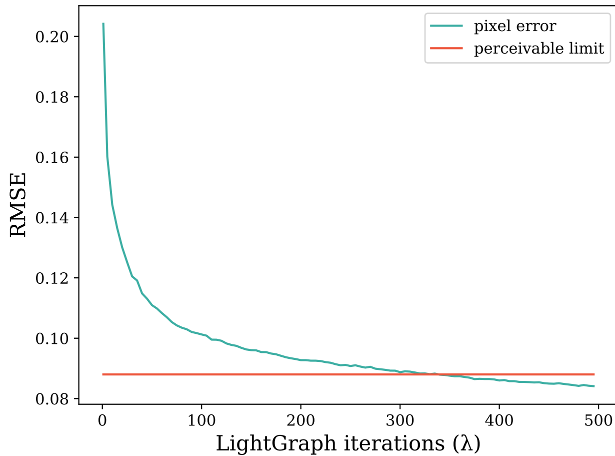

Figure 7: Root Mean Square Error (RMSE) of the pixel difference between a converged and a non-converged render with respect to the number of LightGraph iterations. The orange horizontal line represents the approximate point where the error becomes unperceivable to the eye.

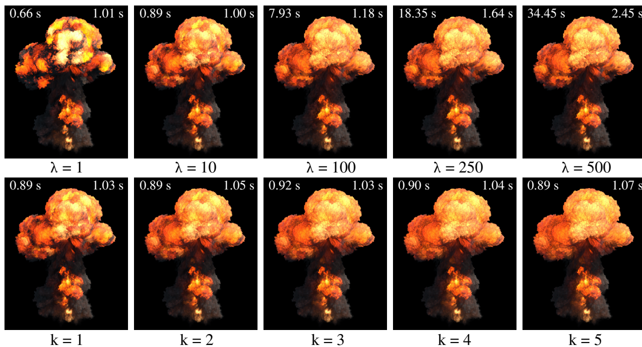

Figure 8: Rate of convergence w.r.t λ and k. The first row shows the results of increasing the number of LightGraph iterations λ using a fix number of points per kernel $k = 1$. The bottom row shows the results of increasing k when rasterizing the multiple scattering estimate to a low-resolution grid using a fixed $\lambda = 10$. The runtime at the top of each image represents the amount of time spent at calculating the LightGraph data structure (left) and rasterizing it to a volumetric grid (right).

The results from the second row indicate that increasing k is a lot less costly than increasing λ, but we also see that this tends to smooth out the estimate which in some case might not be desirable. In a case where the volume is not animated, we could get perfectly good results with $\lambda \approx 25$. If we have an animated sequence of volume and can't increase k to preserve the details, we might need to go to $\rightarrow 500$ in order to reach temporal stability across the sequence. Note that k is always multiplied by λ to make the kernel size independent from λ. This means in practice that a combination of $k = 5$ and $\lambda = 10$ defines a kernel that uses 50 points per voxel estimate.

6 RESULTS

In order to see how our technique compares with state of the art solutions, we chose to render multiple scenes using both LightGraph (our method) and the latest available version of Pixar's production renderer: RenderMan 23.2 [15]. We have tried to render both images within a similar timeframe to see what both softwares could generate under a tight render time budget. We have also provided a more converged version of the RenderMan image as a reference. For RenderMan we use their path tracer integrator with a limit of 10 multiple scattering bounces if not specified otherwise. The hardware used for the tests is an Intel Hexa-Core i7-3930K CPU @ 3.8GHz with 64GB of RAM. Each image is rendered at 960x540 which correspond to a quarter of a Full-HD frame in terms of pixel count. The render specification of each image will be given in each subsection respectively.

Note that as opposed to path tracing where the render time scales linearly with the amount of pixel in the image, our method decouples the multiple scattering computation from the image size. This means that only the single scattering part of the algorithm, which is relatively cheap, will depend linearly on the number of pixels. The multiple scattering is therefore computed independently of the pixel count which is especially efficient for high-resolution images of volume covering a large portion of the frame.

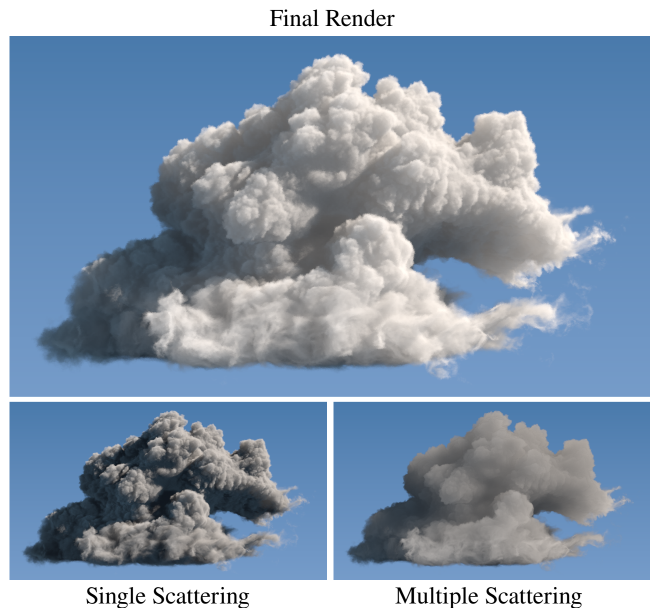

Figure 9: Render of the Disney Cloud using LightGraph. Single and multiple scattering contribution are isolated at the bottom. Render Specifications: Lighting: 3 directional lights, Iterations: 25, Points per Kernel: 1, Scatter Scale: 3.0, Scatter Density Scale: 0.5, Scatter Density Min: 0.025, Scatter Density Mask Power: 0.25, Volume Color: 1.0, Scattering: 1.5.

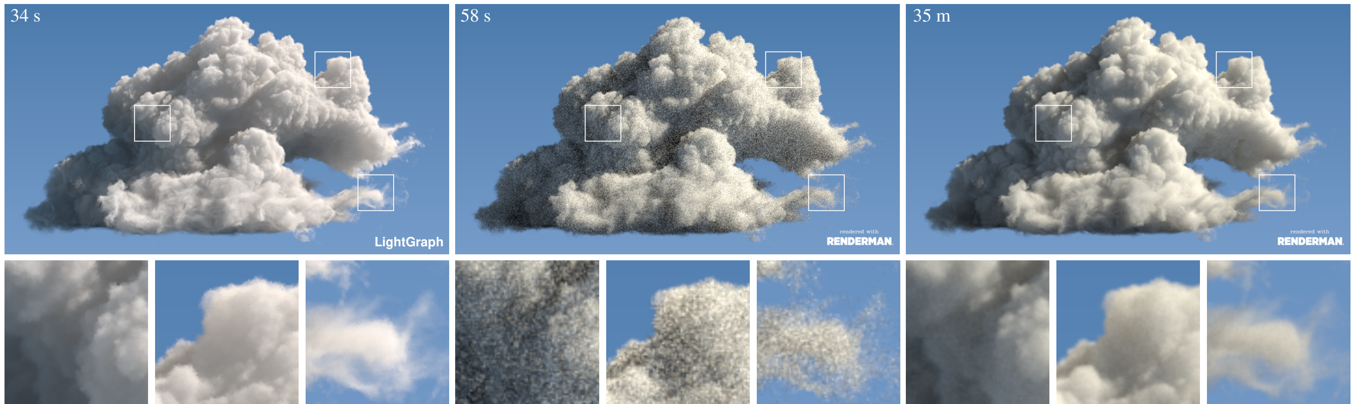

Figure 10: Render of the Disney Cloud Data Set (quarter resolution) with 24 million voxels comparing both LightGraph and RenderMan 23.2. As seen on the right, the physical nature of clouds makes it very hard for Monte Carlo path tracing to converge in a reasonable timeframe.

6.1 Cloud

As previously mentioned, we use an overly simplified phase function model where we treat every volume as isotropic for the sake of simplicity. Therefore, the clouds rendered in figure 10 won't feature the correct phase function with forward and backward scattering peaks. That being said, we see that in a very short amount of time our method achieves a noise-free image that would be perfectly stable to both camera and light animation. Regarding the image on the right, the path tracing approach does not take advantage of the low-frequency nature of the expected solution for multiple scattering. Each pixel is solving its own individual problem which results in a lot of unnecessary calculation being done. In a setting where the image is only 960×540 it might not be a problem to let the render converge in ± 2h using the technique on the right. The problem is that most animated and live action feature are now rendering at 4k which means 16 times more pixels and thus 32h to render a single frame. Even assuming a studio has access to infinite parallel render power on a render farm, this kind of turnaround times really starts to hurt production.

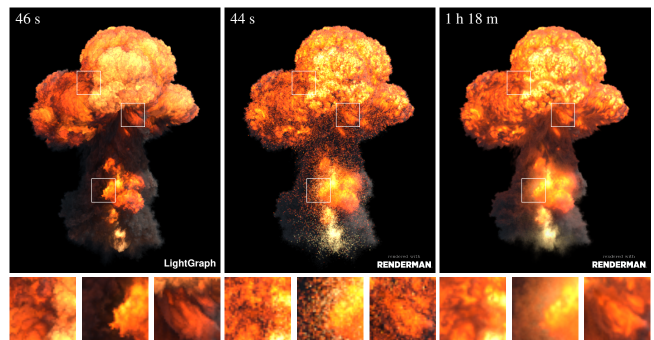

Figure 11: Render of an Explosion with 31 million voxels comparing both LightGraph and RenderMan 23.2. Looking at the image in the middle from RenderMan, the high frequency noise generated by the multiple scattering of the emission component makes the image unusable in production.

6.2 Explosion

Rendering explosion is a very expensive process because of the volumetric lights casting soft shadows embedded in highly heterogeneous density field. Most light emitted from within the explosion is absorbed by the density. Only the light emitted close to the edge of the volume or in area of low density will be able to travel longer distance before being absorbed. This is a very complex situation to handle with traditional path tracing since most paths won't be able to connect the camera with self-emitted light. This problem is particularly visible in the middle render of figure 11. This high frequency noise can eventually be cleaned up using variance threshold adaptive sampling but the process takes a very long time to reach convergence. On the other hand, our method will be able to capture the contribution of short paths by adjusting the size of the kernel used to estimate multiple scattering. The longer paths will also be properly approximated using SPF. For each iteration, within the finite set of possible paths of the current graph, light will be scattered using the least occluded path which makes our method particularly good in condition where only a few very specific paths would have a strong contribution to the final result. It is possible to obtain a very believable but biased solution using just a few iterations (±25) with our method, but the result will be unstable in cases with volumetric fields animation. This animated scene has required 500 LightGraph iterations to reach a stable flicker-free solution.

6.3 God Rays

The scene rendered in figure 15 is probably the most challenging in terms of single scattering because of the optically thin media filling most of the screen where we see shafts of light from one cloud layer to another. Multiple scattering, in this case, would be primarily visible in optically thick areas such as the bottom and top cloud layers. The contribution of multiple scattering is especially important in a backlit setting like this one because it gives important

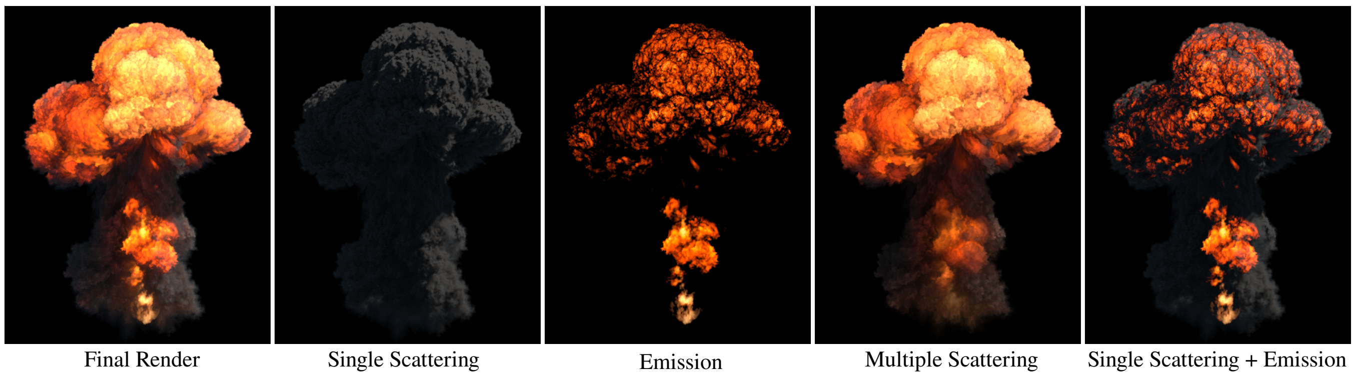

Figure 12: Render of an explosion with self-illumination using LightGraph. Emission along with single and multiple scattering contribution are isolated. A composite of the emission with the single scattering (last image at the right) is also provided to show the explosion without multiple scattering contribution. Render Specifications: Lighting: one directional light and one environment light with a constant blue color, Iterations: 500, Points per Kernel: 1, Scatter Scale: 3.0, Scatter Density Scale: 1.0, Scatter Density Min: 0.025, Scatter Density Mask Power: 1.0, Volume Color: 0.1, Scattering: (1.0, 1.25, 1.5).

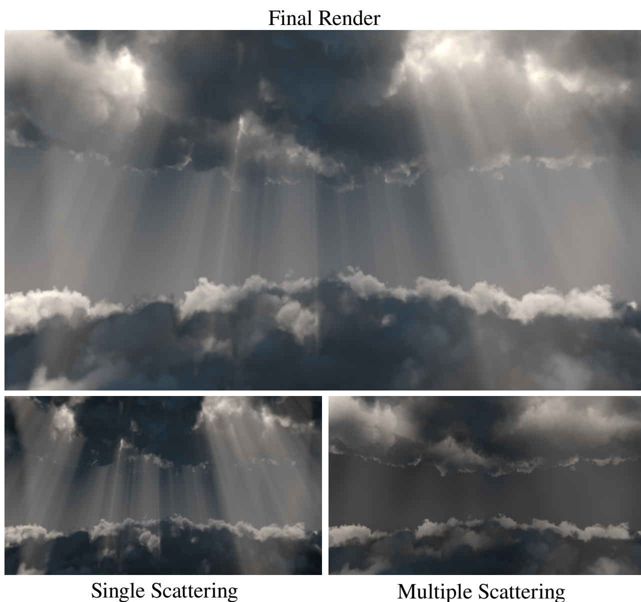

Figure 13: Render of sunbeams shining through openings in clouds using LightGraph. Single and multiple scattering contribution are isolated at the bottom. Render Specifications: Lighting: 3 directional lights, Iterations: 25 Points per Kernel: 2, Scatter Scale: 3, Scatter Density Scale: 1.0, Scatter Density Min: 0.05, Scatter Density Mask Power: 0.25, Volume Color: 1.0, Scattering: 1.5.

information regarding the depth and overall shape of volumetric objects. The image rendered with RenderMan is still very noisy even with almost twice as much render time. Their path traced version computes multiple scattering uniformly everywhere regardless of its contribution. In a real-world scenario, this scene would be split up in at least two render passes in order to properly control the

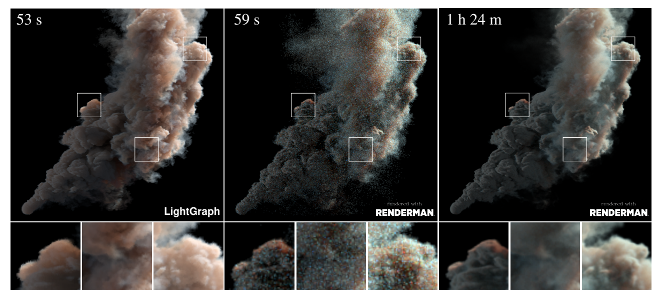

Figure 14: Render of a Large Smoke Plume with 38 million voxels comparing both LightGraph and RenderMan 23.2. Spectrally varying scattering increases the level of perceptual noise in the case of RenderMan. Render Specifications: Lighting: 3 directional lights, Iterations: 500, Points per Kernel: 2, Scatter Scale: 2.0, Scatter Density Scale: 1.0, Scatter Density Min: 0.075, Scatter Density Mask Power: 0.25, Volume Color: (0.8, 0.9, 1.0), Scattering: (0.75, 1.15, 1.5).

quality of the sharp god rays as well as the multiple scattering in the top and bottom cloud layers.

6.4 Large Smoke Plume

This scene features a large smoke plume with wavelength dependency that shifts the color of the light as it scatters through the participating media in figure 14. The image from RenderMan contains a large amount of high-frequency noise similar to the one found in previous examples. More specific to this scene, we see that the perceptual noise is amplified by the spectrally varying scattering of the smoke plume, the noise is no longer monochrome like it was in previous examples which makes it more visible. For the image on the left, our method accounts for this vector value scattering by multiplying the multiple scattering component by the inverse of the scattering parameter which gives it the expected tint.

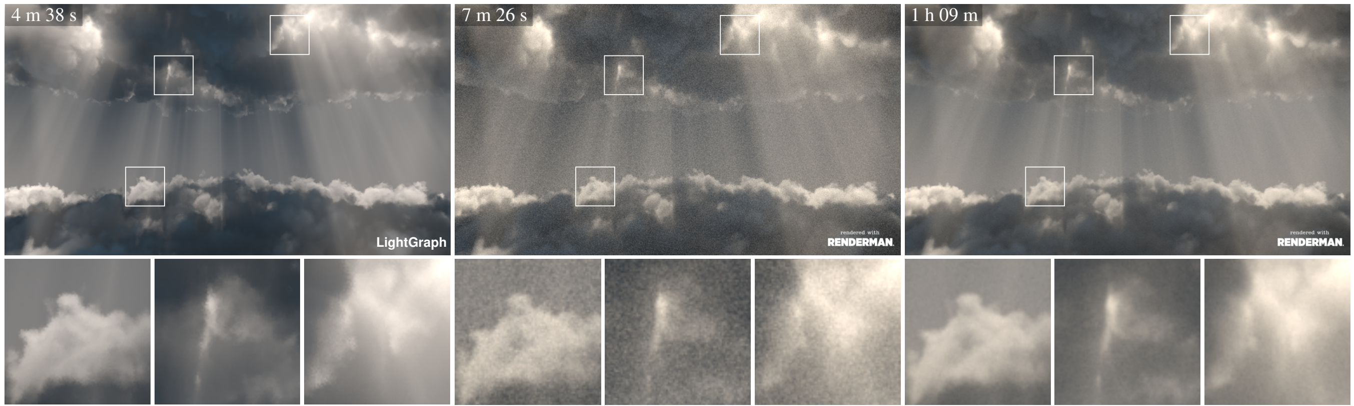

Figure 15: Render of God Rays with 65 million voxels comparing both LightGraph and RenderMan 23.2 with 2 multiple scattering bounces. For the image in the middle, the number of multiple scattering bounces was reduced from 10 to 2 to help reach convergence.

6.5 Snow

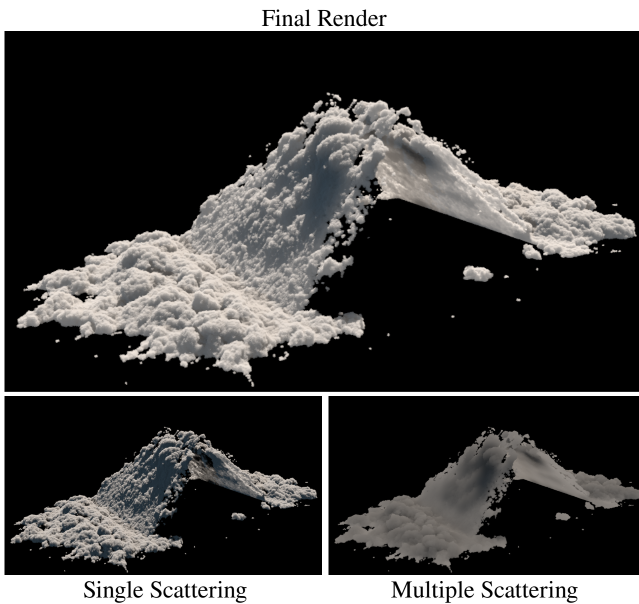

Figure 16: Render of a snow simulation using LightGraph. Single and multiple scattering contribution are isolated at the bottom. Render Specifications: Lighting: 3 directional lights, Iterations: 500, Points per Kernel: 5, Scatter Scale: 1.0, Scatter Density Scale: 0.001, Scatter Density Min: 0.025, Scatter Density Mask Power: 0.25, Volume Color: 1.0, Scattering: 1.5.

In this scene, we try to approximate the light transport of millions of individual ice crystals using high density participating media. Our method allows us to effortlessly decouple the volume density of single scattering from the volume density of multiple scattering. We can therefore avoid creating an explicit surface to model the snow. The left image in figure 17 shows both the detailed "surface" of the snow approximated by the very high density volume coupled with the soft multiple scattering effect obtain by scaling the density values by 0.001 for the multiple scattering computation. This decoupling is not possible with path tracing, which leads to a compromise where the edge of the volume is not sharp enough to properly represent a surface and the density of the participating media is just too high to properly let the light scatter inside to prevent the hard shadows from RenderMan's version. There would obviously be ways to recreate a similar look with RenderMan using a separate surface coupled with the volume, but those setups tend to take even longer to converge.

7 LIMITATION AND FUTURE WORK

Although this method has proven to be quite powerful compared to what is used in production at the moment, it still has a fair bit of limitations. In this section we discuss the weakness of the system as well as the possible improvement that could be made upon the current architecture.

7.1 Density Mask

To reduce memory usage and computation time, we store the multiple scattering in a low-resolution grid (MSG) since it only contains low-frequency information. To get proper results, we still need to use the high resolution density grid as a mask to the MSG while rendering. This masking operation leads to correct results in most cases, but can also introduce some artefacts. In the explosion scene, see figure 11, we observe that the multiple scattering is pretty weak at the bottom of the explosion while the emission contribution is clearly very important. When the density mask is applied, it reduces the multiple scattering contribution in areas of low density which leads to a biased estimate in those regions.

7.2 Screen Space vs World Space

Our solution precomputes the multiple scattering for the whole volume before starting the actual per pixel render. This is ideal in the cases where the volume being rendered fills most of the frame. In cases where we would only see a small portion of the volume

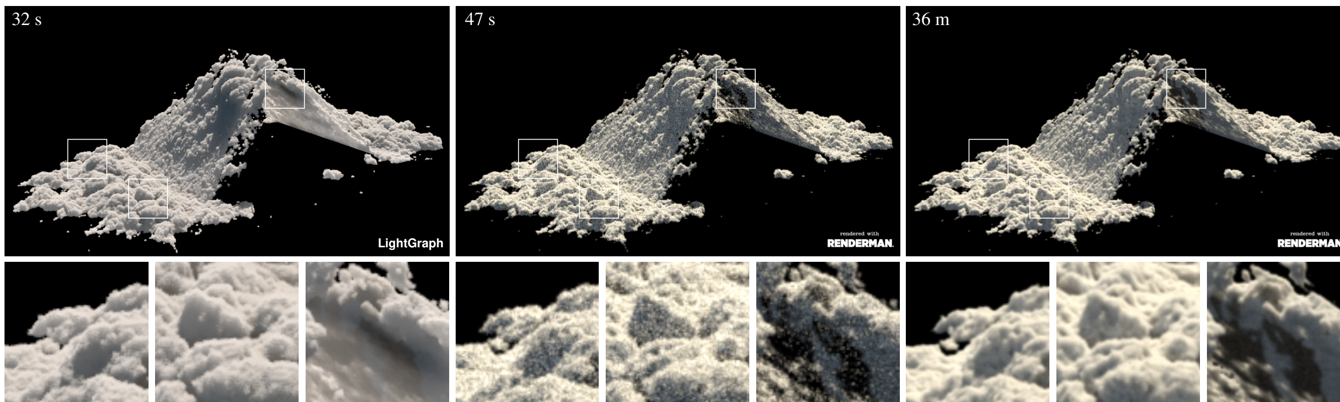

Figure 17: Render of Snow with 3 million voxels comparing both LightGraph and RenderMan 23.2. On the right, the high density of the snow completely occludes the light in some area leading to hard shadows that are less visually pleasing than the render on the left.

being rendered, we would still need to compute multiple scattering for the whole volume. Similarly, if we consider rendering a full cloudscape formed of thousands of cloud instances where some of those clouds are only visible as subpixel objets. The only way for our method to go though with this would be to compute the MSG of the clouds within the current render buckets and then flush this data from memory as soon as possible to make room for the MSG of the next buckets. This should work but would definitely require more memory than traditional path tracing.

7.3 Disconnected Islands

One problem that can arise while generating the graph is the presence of disconnected islands. In a scene where a cloud would be made out of three disconnected parts where one of them is receiving strong direct illumination (see figure 18). We might expect this first bit of cloud to scatter light towards the other pieces of the cloud that don't receive direct lighting. It turns out that since the point cloud is only scattered inside voxels with density greater than 0 and that connections beyond a certain search radius is prohibited, the two disconnected islands of density are most likely also disconnected in their LightGraph which will prevent them from transferring radiance between density islands. The reason why LightGraph is not filling the empty space with points is to focus the point cloud resolution where it matters the most, since the point count is capped at 750. There is probably a way to connect those density islands without wasting points in empty space, but no elegant solution has been found at the time of writing.

7.4 Sequence Optimization with Caching

Our method has been implemented in such a way that each frame is independent from one another. Depending on the sequence of frame that needs to be rendered some serious optimization could be made. Caching intermediate steps to disk could save a ton of time when rendering sequences. In a scenario where the volume and the lighting is static while the camera is moving, the MSG could be written to disk and then read back for the subsequent frames. Therefore, the multiple scattering would be calculated once for

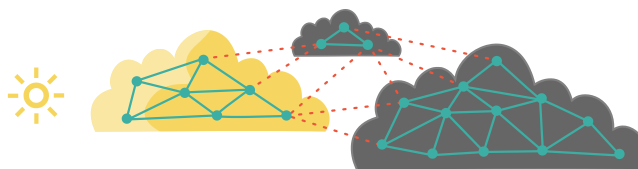

Figure 18: Disconnected Islands: The 3 clouds shown here represent a single density grid with disconnected density patches. The green edges represent actual connections in the LightGraph, the dotted red edges are connections that would be required to properly diffuse radiance to other disconnected density island. Since no points are kept in voxels of density 0 and since connection lengths are bounded by a maximum search radius, those red connections cannot exist. Thus the scattering only happens inside the island receiving direct illumination which is a problem.

the whole sequence and only the single scattering would need to be recomputed for every frame. In a low-resolution setting, this could even approach real-time performance. In the case where the volume is static but the camera and lighting are dynamic, it would be possible to reuse the generated LightGraph iterations by only recomputing the *Vertex Lighting Calculation* and the *Radiance Diffusion* steps before baking the multiple scattering to a grid. This would still save a significant amount of computation by having the slow parts of multiple scattering estimation being calculated once for the whole sequence.

7.5 Random Walks

It would be interesting to investigate random path generation instead of relying on shortest path finding. This would definitely eliminate a lot of constraints like the amount of point per iterations and could also yield a more statistically sound solution with less bias. On the other hand, this change could also lead to a large increase in noise and decrease the global performance. Since the solution is being rasterized down to a grid using a Gaussian kernel,

there might still be ways to converge much more quickly than with path tracing even if we use a similar heuristic to determine light paths.

7.6 GPU Implementation

With the recent wave of renderers ported to the GPU, we have seen that GPUs can greatly accelerate parallelizable algorithms to speeds way beyond what can be achieved even with the best CPUs available. It would be interesting to see how of an improvement could be gained since some scene are already approaching real-time performance with our current implementation. This could open up possibilities for interactive applications to view and manipulate volume rendered with multiple scattering.

7.7 Geometric Occluders

Geometric occlusion have not been implemented in the system due to time constraints, but the extension should be fairly straight forward. Each edge of the LightGraph just needs to be cast against the geometry in the scene before performing the *Raymarch Graph Edges* step of the algorithm. If the ray hits a geometry that lies in between the two points forming the edge, the transmittance should be set to 0 which is equivalent as ray marching a voxel of infinite density. The SPF algorithm will then avoid this edge as it normally would with very occluded edges. The low-resolution of the MSG might create small amount of light leaking around occluder surfaces but this should not create any noticeable artefacts.

7.8 Anisotropic Phase Function

Our method does not support anisotropic phase functions for now and therefore treats all volumes as isotropic. This choice was made to simplify the system, but this could also be a straight forward extension. The *shortest path finding solve* would only need to consider a phase function while evaluating the different branching options for the graph by looking up the angle between connected edges. This would definitely add a burden on the computation side, but it shouldn't be too hard to implement with the current architecture.

8 CONCLUSION

We introduced LightGraph which approximates multiple scattering in both optically thick and thin nonhomogeneous participating media using shortest path finding. Our method requires a small amount of precomputation to estimate multiple scattering in a matter of seconds instead of hours. The number of user parameters has been kept to a minimum while still maintaining a very high level of flexibility. By relying on efficient ray tracing techniques to evaluate single scattering and using our approximation for multiple scattering, we benefit from the strengths of both techniques. Our method is heavily based on the reasonable assumption that multiple scattering is a very low-frequency phenomenon which makes it particularly well suited for density estimation algorithms. We have also shown that this technique yield visually compelling noise-free images at a fraction of the computational cost of Monte Carlo integration which makes it a potentially practical solution for the visual effects industry. With LightGraph we chose bias over variance, flexible over simplified and "good-looking" over physically accurate.

REFERENCES

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms.* MIT press. 693–697 pages.
[2] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* 15–22.
[3] Kyle Hegeman, Michael Ashikhmin, and Simon Premože. 2005. A lighting model for general participating media. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games.* 117–124.
[4] Henrik Wann Jensen and Per H Christensen. 1998. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques.* 311–320.
[5] Florian Kainz, Rod Bogart, and Piotr Stanczyk. 2009. Technical introduction to OpenEXR. *Industrial light and magic* (2009), 21.
[6] James T Kajiya and Brian P Von Herzen. 1984. Ray tracing volume densities. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 165–174.
[7] Simon Kallweit, Thomas Müller, Brian Mcwilliams, Markus Gross, and Jan Novák. 2017. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–11.
[8] David Koerner, Jamie Portsmouth, Filip Sadlo, Thomas Ertl, and Bernd Eberhardt. 2014. Flux-limited diffusion for multiple scattering in participating media. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 178–189.
[9] Eric P Lafortune and Yves D Willems. 1996. Rendering participating media with bidirectional path tracing. In *Rendering techniques' 96.* Springer, 91–100.
[10] Don P Mitchell. 1987. Generating antialiased images at low sampling densities. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques.* 65–72.
[11] Ken Museth, Nick Avramoussis, and Dan Bailey. 2019. OpenVDB. In *ACM SIGGRAPH 2019 Courses.* 1–56.
[12] Ken Museth, Peter Cucka, Mihai Aldén, and David Hill. 2020. OpenVDB. https://www.openvdb.org.
[13] Mark Pauly, Thomas Kollig, and Alexander Keller. 2000. Metropolis light transport for participating media. In *Rendering Techniques 2000.* Springer, 11–22.
[14] Chuck Pheatt. 2008. Intel® threading building blocks. *Journal of Computing Sciences in Colleges* 23, 4 (2008), 298–298.
[15] Pixar. 2020. RenderMan. https://renderman.pixar.com/.
[16] Christian Robert. 2014. Machine learning, a probabilistic perspective. , 507–513 pages.
[17] Jos Stam. 1995. Multiple scattering as a diffusion process. In *Rendering Techniques' 95.* Springer, 41–50.
[18] László Szirmay-Kalos, Mateu Sbert, and Tamás Umenhoffer. 2005. Real-Time Multiple Scattering in Participating Media with Illumination Networks.. In *rendering techniques.* 277–282.
[19] Beibei Wang and Nicolas Holzschuch. 2017. Precomputed multiple scattering for light simulation in participating medium. In *ACM SIGGRAPH 2017 Talks.* 1–2.
[20] Magnus Wrenninge. 2012. *Production volume rendering: design and implementation.* CRC Press.
[21] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. 2008. Real-time smoke rendering using compensated ray marching. In *ACM SIGGRAPH 2008 papers.* 1–12.