Accelerating Path Tracing in Volumetric Clouds using Graph Structures

Tobias van den Hurk

Delft University of Technology

TU Delft

Accelerating Path Tracing in Volumetric Clouds using Graph Structures

by

Tobias van den Hurk

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on March 17th, 2026 at 9:30

Student number:	4938836	
Project duration:	April 25, 2024 – March 17th, 2026	
Thesis committee:	R. Marroquim,	TU Delft, Thesis advisor
	J. Urbano,	TU Delft
	G. Lu,	TU Delft, Daily co-supervisor
	C. Peters,	TU Delft, Daily co-supervisor

Cover:	Render of realistic cloud, model provided by Disney, rendered using the method presented in this thesis
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Rendering volumes using path tracing can produce stunning images, but the process is notoriously expensive. For some volumes such as clouds, there are cases where the volume has no interaction with other objects in the scene. For these cases, combined with an isotropic phase function, unidirectional path tracing from the camera computes many similar paths regardless of camera position. Computing these light paths beforehand to determine the full radiance transport through the volume speeds up the rendering stage drastically, with the result still being physically accurate.

In this thesis, we present a two-stage method designed for the aforementioned cases. The first stage is a precomputation stage, in which light paths are traced through the volume and stored in a space-efficient manner using a graph structure. Through this graph, radiance transport is then computed. In the rendering stage, the graph structure with radiance values can then be used to render the volume. The results show that our method can produce renders with negligible bias. They also show that the method is heavily constrained by space requirements as volume size increases, which results in an increase in bias for larger volumes. The main advantage of the method is its efficiency in rendering multiple images of the same object.

Contents

1

Introduction

During the 2010's, the method of using computers to render volumes, also known as participating media, made huge progress. Path tracing from the camera became the norm, making many older methods obsolete [5]. In the modern rendering method, paths are traced from the camera in a physically accurate way, accumulating or losing radiance as they travel through the medium. This physically based type of method gives the best accuracy, but has the downside of being more expensive. Previous methods all used approximations in their theory, required precomputation steps or only worked for certain types of media to make computation feasible [22].

Although path tracing only became the industry norm for rendering participating media during the 2010's, the theory was known and had been researched decades prior. Early methods already produced both theoretically and visually correct renders [20], but were limited by computational power. One of the reasons why path tracing was so expensive compared to other methods is that to evaluate the complex distribution of light, samples are taken from the distribution by randomly tracing light paths. This creates a large amount of variance, but can approach the correct result given enough computation time. Other faster methods solved this problem of variance by computing an approximation, instead of computing the physically correct light behavior. This reduces variance at the cost of introducing bias (error), but as long as the bias remains small, this is an acceptable trade-off. As Pérez et al. [22] surveyed, the majority of volume rendering methods limited variance by being deterministic.
One of these deterministic methods is the radiosity technique, first detailed by Gouraud [7] and later improved in many aspects, for example, to be more efficient and handle more types of lighting and surfaces [2] [6] [3]. It was also extended to work for participating media [24]. The theory of radiosity is one of the key inspirations for the method introduced in this paper.

In the present day, the issue of computational power has been solved, by both algorithmic and hardware efficiency improvements, and unbiased path tracing with negligible variance is possible on consumer hardware. Many courses and books now explain the theory of volumetric path tracing in detail [5] [4]. For this thesis, the PBR (Physically Based Rendering) book [23] was used to acquire almost all relevant knowledge of the subject. Besides a book, the authors also maintain an open-source rendering framework called PBRT (Physically Based Ray Tracer), with major ray tracing techniques implemented. This framework is used as a technological basis for this thesis. The main rendering technique available in PBRT is called VolPath (Volumetric Paths), which can rival industrial volumetric path tracers. This method is what our results are compared to and serves as the ground truth for this thesis.

When a volume is rendered by a camera-based integrator like VolPath, for some volumes with the right properties the path tracing work done in the volume is mostly independent of the camera position. This partial independence allows us to precompute the radiance transport through the volume, cache the result, and cheaply render the volume for any camera position. This advantage over 'traditional' camera-based renderers depends on the scene and medium properties, which are detailed in Chapter 4.

The method in this paper uses both path tracing theory and radiosity theory, linked together with a graph structure to render volumes. First, paths of light are traced through the volume and captured in the graph. Similar segments of the light paths are grouped together in the graph, and each node presents the average region of the medium around it. Without this grouping and averaging the size of the graph would be unbounded with respect to the number of traced rays. Like other methods mentioned earlier, this approximation comes at the cost of bias to make computation feasible.
After the process is complete, a directed graph is available, which shows how light travels through the volume. The edges of the graph are used to construct a sparse transport matrix T, where each edge corresponds to one entry in the matrix. The radiance distribution for the first bounce of light is computed for each node, which is stored in a vector L. These initial radiance values are then transported by computing $T \cdot L$, which gives the radiance distribution after one extra bounce. The sum of many radiance distributions for different depths can be evaluated by writing the problem as a Neumann series, $(\sum_{k=0}^{\infty} T^k) \cdot L$, which becomes our radiance cache.
To render the volume, the radiance cache is used. Although the precomputation stage to acquire the cache is slower than VolPath, the rendering stage of our graph-based method is typically much faster. The fast rendering stage makes our method suited for cases where many frames of the same scene are required. Due to approximation in the graph building stage, the final render can contain bias (see Chapter 5). This bias increases as the nodes of the graph increase in size, as the degree of approximation becomes larger.

2

Background

2.1. Volumetric Path Tracing Fundamentals

A light source emits rays of light, which can reach a volume. A volume has no hard boundaries, rays travel through it and potentially interact with the volume, which can change their properties such as direction. After some types of volume interactions, a new ray is formed. A sequence of these rays is a light path. The process of simulating a path of light is called path tracing.
To accurately trace a path through a volume, the light behavior in the volume is modeled into several different concepts.

Medium coefficients

In the model used by path tracing for simulating light behavior, three types of interactions are possible in participating media:

- *Scattering*, a particle collides with the volume and changes direction, but retains its energy.
- *Absorption*, a particle collides with the volume and loses its energy, stopping its travel.
- *Emission*, the volume emits energy in the form of a particle, in a random direction.

These three types of medium interactions lead to four possible ways of changing the energy of a ray, as scattering can both increase or decrease the number of particles in a ray, which is shown in Figure 2.1.

(a) In Scattering **(b)** Out Scattering **(c)** Absorption **(d)** Emission

Figure 2.1: The four possible ways the energy of a ray can change.

The three types of interactions (scattering, absorption, emission) are modeled by the coefficients μ_s, μ_a and μ_e, respectively. Their value is the probability density of the event occurring per unit distance, with a unit of the form m^{-1}, which can take any non-negative value. These coefficients can be constant throughout the medium, which is then called a *homogeneous* medium, or spatially varying, resulting in a *heterogeneous* medium. Typically, μ_t is used to indicate attenuation from all types of volume interactions, and is defined as $\mu_t = \mu_s + \mu_a + \mu_e$.
For this thesis, we assume medium coefficients are direction independent, and they are used in equations as functions over space, for example, like $\mu_s(p)$ where p is a point in space.

Transmittance

So far, we have considered attenuation at a single point in space. In rendering, the attenuation along a ray and the resulting remaining energy at the end are often needed. To denote the remaining energy, we use *transmittance*. The definition of transmittance between two points is the fraction of energy that leaves one point and arrives in the other, and is written as $T_r(a, b)$ for two points a and b.
The definition of transmittance depends on the optical thickness $\mathcal{T}$ of a ray, which is the integral of μ_t over the ray distance. The equation for transmittance is:

$$T_r(a, b) = e^{-\mathcal{T}} = e^{-\int_0^d \mu_t(a+t\omega)\mathrm{d}t} \tag{2.1}$$

where
d = the distance between a and b
ω = the direction from a to b

Transmittance has several interesting properties. It is reversible, which means that the direction of a ray between two points does not matter for its transmittance: $T_r(a, b) = T_r(b, a)$. It is also multiplicative, which means that the transmittance of a ray from a to c is equal to the multiplied transmittance of rays from a to b and b to c, where b is a point somewhere on the ray from a to c: $T_r(a, c) = T_r(a, b) \cdot T_r(b, c)$.

Phase function

When a particle scatters in the volume, it is redirected in another direction. Given the incoming direction ω_i and outgoing direction ω_o of a scattering event, the *phase function* is defined for the angle θ between the two directions, and is written as $f(\cos\theta)$ or $f(\omega_o, \omega_i)$. When applied on an angle, the resulting value of the phase function can be used as a weight for the direction change of the particle. Since the value of f is normalized, it is also a PDF. Given an incoming direction, this allows us to perform importance sampling on the phase function and sample a new direction based on the phase function's PDF.
If the outgoing direction of a scattering event does not depend on the incoming direction, then the phase function is called *isotropic* and has the same value for any angle θ. Its value is:

$$f(\cos\theta) = \frac{1}{4\pi} \tag{2.2}$$

The isotropic phase function is used for this thesis, which is one of the key assumptions that makes our method feasible. Its counterpart is the *anisotropic* phase function where outgoing direction does depend on incoming direction, which is commonly modeled by the Henyey-Greenstein function [9]. It contains a parameter g ranging from -1 to 1, which controls the direction of scattered light. A positive value of g results in forward scattering, whilst a negative value results in backward scattering. If g is zero, then the Henyey-Greenstein function is equal to the isotropic phase function.

2.2. Rendering Scenes containing Volumes

A light source emits radiance, a radiometric quantity. Radiance is received by the camera, which is how it sees the scene. This radiance can either reach the camera directly from the light source or interact with a volume before reaching the camera. To determine the camera's view, which is called rendering, we must determine how much radiance reaches the camera for each pixel. The Radiative Transfer Equation (RTE) [1] is used for this, which we give in two forms.

Radiative Transfer Equation

To determine the change of radiance in a point p and direction ω, the differential form of the RTE can be used. It accounts for the change of radiance due to volume scattering, absorption, and emission, which are the three types of volume interactions described in Section 2.1. The differential RTE is defined as:

$$\frac{\partial}{\partial\omega}L(p, \omega) = \mu_e(p)L_e(p, \omega) - \mu_t(p)L(p, \omega) + \mu_s(p)\int_\Omega f(\omega, \omega_i)L(p, \omega_i)\mathrm{d}\omega_i \tag{2.3}$$

where
L_e = radiance emitted by the medium
Ω = the set of all directions

Radiance emission and attenuation are handled by the first and second terms of Equation 2.3, respectively. Increase of radiance due to in scattering is represented by the third term, for which the incoming radiance from all directions must be calculated by recursive evaluations of the RTE in new directions.

For rendering, the total radiance in a point and direction is more useful than the change of radiance. The integral form of the RTE, which computes an integral along a ray, gives us the radiance in a point and direction:

$$L(p,\omega) = \int_0^d T_r(p,p') \left(\mu_e(p') L_e(p,\omega) + \mu_s(p') \int_\Omega f(\omega,\omega_i) L(p',\omega_i) \mathrm{d}\omega_i \right) \mathrm{d}t \tag{2.4}$$

where
p' = $p + t\omega$

Comparing Equations 2.3 and 2.4, the terms accounting for increase of radiance remain in the same form. The form of the term for attenuation has changed, however, and is now handled by transmittance instead.
The integral form of the RTE as specified by Equation 2.4 is perfectly suited for determining the radiance received by each camera pixel and rendering the camera's view. It is also what will be used for validating our graph-based volumetric rendering method in Section 4.2.

Monte Carlo Integration

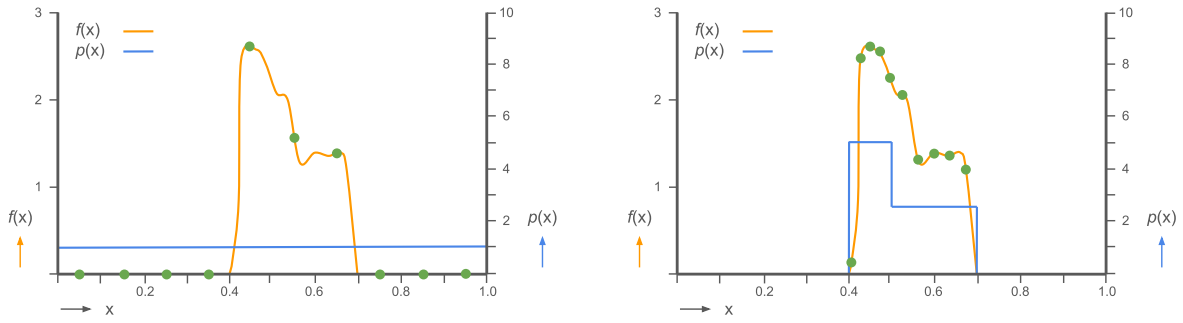
We now have the function we need to solve for rendering the scene, but it is a continuous and recursive function depending on a potentially irregular volume. This makes integrating the function analytically impossible, and we need a more efficient method instead. This method is Monte Carlo integration.

Monte Carlo integration approximates an integral by repeatedly taking a random sample of the function to be integrated, the cost of such a sample only grows linearly with the dimensionality of the function. The more samples there are, the better the approximation.
Formally, the definition of Monte Carlo integration for approximating a function $f(x)$ is:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \tag{2.5}$$

where
I = the integral to be computed
N = the number of samples
X_i = a sample from the space over which the function is defined
$f(x)$ = the function to be integrated
$p(x)$ = the sampling PDF used for drawing samples

The distribution of $f(x)$ can be of any shape, with potentially large regions having no value. The key is to sample points from the domain such that they have a significant value for $f(x)$.



(a) Sampling $f(x)$ using a uniform PDF.

(b) Sampling $f(x)$ using a PDF similar to $f(x)$.

Figure 2.2: Using a sampling PDF which is a rough estimate of $f(x)$ improves efficiency, by increasing the average value of the samples.

Samples are drawn according to $p(x)$, so the distribution of $p(x)$ is what determines the efficiency of MC integration. Choosing $p(x)$ to evaluate points in the domain with the highest possible average contribution, for example avoiding evaluating samples with zero contribution, is called importance sampling. In order to perform importance sampling, some knowledge of the distribution of $f(x)$ is required. A rough estimate can already make a significant difference, as shown in Figure 2.2.

RTE Integration using Path Tracing

The final piece needed to solve the rendering problem is how to get samples of the RTE for MC integration, for this we use path tracing. For a sample to have a value the path must connect the light source to the camera. Although light paths physically originate from the light source, any path connecting the light source to the camera is valid. Therefore, the tracing of paths can also start from the camera, which is generally much more efficient as paths only have to be traced from one position and a small set of directions, compared to potentially any direction and/or position for light sources.

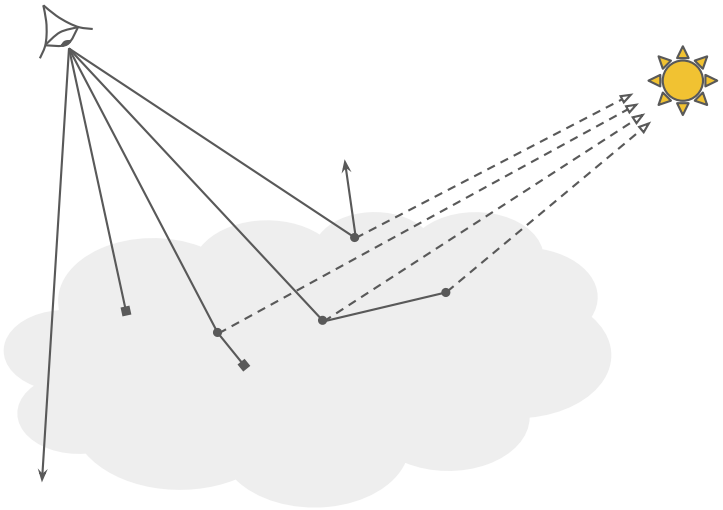


Figure 2.3: Example of paths with maximum depth of 2 being traced through a volume to render the scene. The traced paths are solid lines, circles on the paths are scattering events. Solid arrows indicate paths escaping the volume before reaching maximum depth, and squares indicate paths ending due to absorption. The dotted lines illustrate light importance sampling.

When tracing paths from the camera, finding paths that connect to the light source can still be very costly, as the probability of randomly intersecting the light source is usually very low or can even be zero. To solve this problem of inefficiency, importance sampling is used. After sampling a scattering event, the traced path is connected to the light source. Instead of a random direction a direction towards the light source is chosen, and the adapted sampling PDF ensures the average PDF-weighted sample value remains the same.

3

Related Work

Although camera-based path tracers like the VolPath integrator from PBRT perform the best for most scenes with participating media, other methods for rendering participating media are also still researched. These methods offer interesting techniques, and some of them can even outperform the state of the art for certain scenes.

Metropolis Light Transport was one of the first practical unbiased rendering algorithms. Introduced by Veach and Guibas [26], the method was later extended for participating media by Pauly et al. [21]. Unbiased methods were generally very costly to run, so approximation methods based on radiosity theory were the norm. MLT is unbiased, but much more efficient than other unbiased methods of the time due to the key novel idea of using Metropolis sampling to acquire light paths.
The general idea is to make finding paths with a non-zero contribution cheaper. To do this, existing knowledge of paths is used instead of performing naive sampling. The method starts by generating random paths from the light source. Additional paths are found by altering these paths slightly, also referred to as mutation. Not all new paths are guaranteed to have a high contribution however, as path mutation is a form of random search, so new paths are accepted based on their contribution. This results in the random search being directed towards paths of high contribution, and the efficiency of the MLT method is due to this high average path contribution.

Radiosity is an older method compared to the other methods presented in here, and is not used for general-purpose rendering anymore. It was introduced by Gouraud [7], and the concept was later called radiosity. Initially very limited in scope, it was later extended with additional features such as diffuse to diffuse surface lighting [6], occlusion and complex scene support [2], and participating medium support [24].
In the radiosity method, all surfaces in the scene are divided into patches (rectangular areas), and volumes into voxels. Each patch contains an amount of emitted radiosity. Each bounce of light is simulated by computing the new emitted radiosity of each patch, which is the sum of the emitted radiosity of all patches weighed by the respective form factors. A form factor is the fraction of radiosity emitted by one patch which reaches another patch, which is stored in a 2D matrix. The transport of radiosity is then performed by multiplying the form factor matrix with the vector of emitted radiosity per patch.
All of the steps detailed so far are done in a precompute stage. The final vector with radiosity values is then used for rendering. The concept of discretizing the scene into areas and computing the transported light to each area by solving a matrix vector product in a precompute stage, after which quick rendering is possible, was a key inspiration for this thesis.
The precompute stage of the traditional radiosity method costs $O(n^2)$ for both time and space, where n is the number of patches in the scene. Only after this stage is done an image can be rendered. A progressive variant of the radiosity method was introduced by Cohen et al. [3], which reduces the space requirement to $O(n)$ and also produces approximate results whilst the precompute stage is running.

Radiance Caching is a biased rendering method without a precompute stage, designed for participating media [13]. The method computes radiance values and their spatial gradients, and stores them in a

map. Radiance caching does not use a precomputed map, instead radiance computed during camera based rendering is cached. The view oriented nature of the method ensures only relevant radiance values are computed, which is more efficient than fully precomputing a map.
To render a point in the medium, first the cache is checked for nearby gradients. If none are found the local radiance and gradient is computed, stored in the cache and used for computing the path contribution. If cache values are found their values are extrapolated using their gradients to find the correct path contribution.
An improvement to the base radiance caching method was made by Marco et al. [19], who directly extended the original method for participating media. Instead of using first-order derivatives of the gradients to determine the spatial distributions of cache points, second-order derivatives are used as well to better account for sudden changes in media. The computation of gradients is also improved by taking occlusion into account for calculating the derivative. These improvements make the radiance caching method more suited for media interacting with surfaces, and also for media with more detail.
Despite the improvements by Marco et al. [19] it is noticeable that all the presented results only contain translucent, fog-like media. The original work by Jarosz et al. [13] shows one opaque volume, but with very low detail and clearly biased. As Marco et al. [19] show scenes with improved results for most of the original work but not for the opaque cloud, it is not clear whether the method is suited for rendering large opaque volumes, which is the focus of our work.

The methods described below are variants of bidirectional path tracing. It is a general classification for any bidirectional method, as a method can be called such when paths are (partially) traced from both light source and camera, but these methods still vary greatly. The main advantage of tracing paths from both light source and camera, is that for certain scenes it can be hard to find contributing paths for some areas if only the light source or the camera is used as starting point. By using both, rendering these areas becomes much more efficient.

Bi-Directional Path Tracing is a bidirectional method with equal contribution from both light and camera. Just like MLT, BDPT is another unbiased early path tracer, and has no precompute stage. First proposed by Lafortune and Willems [17], like in MLT the key is to find paths with the highest possible contribution. Paths are randomly traced through the scene from the camera and light, and the scatter points are saved. All scatter points from each path are connected to the points of the other, creating $O(n^2)$ paths from only 2 paths traced. As the contribution to the camera varies greatly between paths, it is key to use good weights for the paths when taking their weighted average.
Path weights are chosen according to surface properties of the camera path points [17]. For specular surfaces, the continuation of the camera path has higher weight, whilst for diffuse surfaces, the light path is more important. Despite the use of good weights, this first version of BDPT suffers from noise in the final image. This is due to the use of the geometric term in the rendering equation, which can give extreme contribution values for paths with long or short segments.
A significant step for BDPT efficiency was made by Veach and Guibas [27], who used multiple importance sampling to combine paths. Given two paths and all of their combinations, each path is seen as an estimator for MIS, which performs better the more estimators are available. Using each path's PDF, all paths are combined using MIS, which greatly reduces variance and also eliminates the geometric term noise present in the original work. Shortly after this publication [27], BDPT was extended with participating media support [18].

Photon Mapping is a biased bidirectional rendering method which traditionally requires a precompute stage. This two-stage method first traces paths from the light source to form a lighting map of the scene, then the scene is rendered using the map. Published by Jensen and Christensen [14], similar two-stage methods already existed at the time but were limited to mapping paths with only certain types of reflections. The photon mapping method is more general and can map all types of reflections, although for optimization, only caustic reflections are handled with the photon map. For other reflections, different types of precomputed maps are used. To estimate the irradiance coming into the surface point being rendered, the n nearest photons from the photon map are gathered. The combined energy of the gathered photons is normalized for the search area size.
The main strength of photon mapping is the increased efficiency for caustics compared to camera-based path tracers. Objects that create caustics have complex reflection and refraction models, just like participating media. Jensen and Christensen [15] realized this as well and extended their method to support participating media. Instead of gathering photons at surface scattering events, photons are

gathered at set intervals along a ray through the volume. This method is quite expensive as many search points per ray are required. This inefficiency was improved by introducing a method to find all photons in the volume near the ray much faster than performing many point queries [10].
A major drawback of the photon mapping method is that it uses a precomputed map for rendering, and the accuracy of this map is limited by the available memory. To achieve photon map accuracy not limited by space, Hachisuka et al. [8] introduced a progressive photon mapping method. This method does not have a precompute stage, and does not use the full map at once. Instead, it first determines all points to be shaded, and then starts tracing paths from the light source. After a path is traced, the irradiance contribution to shading points is calculated and then the path is discarded. This path tracing phase continues until the desired accuracy has been reached, at constant memory usage.

Photon Beams builds on the concept of photon mapping. Similar to photon mapping, it is also a biased bidirectional rendering method requiring a precompute stage, first proposed by Jarosz et al. [11]. Volumetric photon mapping traces photons through the volume and records the scattering locations in a map. The volume is then rendered by estimating the radiance at points along the rendering beam, at each point the map is queried to find nearby photons and the corresponding radiance. To get a correct radiance estimate for a point, enough photons have to be found, and therefore, the photon map must be densely populated.
To make better use of the info available during photon tracing, the photon beam method also records edges between traced photons. These beams indicate where photons traveled, and can also be used to estimate the in-scattered radiance for a point. As the photon beams record much more information than points, significantly less storage is needed with photon beams to maintain the same level of accuracy compared to photon points. Jarosz et al. [11] define that both radiance queries and photon paths can be represented as points and beams, and give the 9 different possible interactions between queries and photons, each with its own formula computation method.
The photon beams method [11] requires a precomputation stage to compute the photon beam map. As with other two-stage methods, a variant without a precompute stage of the photon beams method was developed[12]. This method can render a scene with arbitrary precision at constant memory usage.
An analysis of the performance of photon points versus photon beams is performed by Křivánek et al. [16], who conclude that the photon beams method does not universally outperform the photon point method, and the performance advantage depends on the type of medium. As there exist many techniques for rendering participating media, Křivánek et al. [16] proposed a new method to combine point-point, point-beam, and beam-beam photon mapping techniques from the work of Jarosz et al. [11]. Multiple Importance Sampling is used to combine these, together with regular Monte Carlo integration, such that all the estimators are used optimally. This technique extends the standard MIS as introduced by Veach [25], to combine the conceptually different photon mapping and MC integration techniques.

Monte Carlo integration has become the norm for rendering due to its simplicity, flexibility and general performance advantage over alternative techniques. Despite much research on these alternative rendering techniques, such as those detailed above, their performance advantage for certain types of scenes has decreased as the performance limit of these methods was reached. This can be seen by the dates of significant research papers released for the techniques analyzed above, as very few major improvements have been made in recent years. This is in line with the industry shift towards Monte Carlo integration [4].

4

Methodology

For this thesis, the scene and its properties are heavily constrained to create a simple scene, which allows us to keep the problem as narrow as possible.
The scene constraints are:

- Non-emissive medium
- Spectrally constant medium
- Isotropic phase function
- One distant light source
- Scene consists of one volumetric object

More details of these constraints, how they influence the method's efficiency, and the cost of solving the constraints are given in Section 4.8.

The method presented in this paper consists of four steps, the sequence of steps is visualized in Figure 4.1. Steps 1 to 3 are performed in a precompute phase, whilst Step 4 is the rendering phase.

1. **Building the Graph** Paths are traced through the volume, and are grouped together to form a graph consisting of nodes and edges. Initial tracing is performed from the light source, after which more paths are traced through sparse regions of the graph to reinforce them.
   Path segments can be grouped to the same edge multiple times, the weight of an edge is computed by calculating the number of times it was sampled, divided by the total number of samples leaving the edge's originating node.
2. **Computing Direct Radiance** Ratio tracking is performed from each node in the graph to the light source to determine the amount of direct incoming radiance to each node.
3. **Transporting Radiance** The direct radiance of each node is transported through the graph, using the edge weights computed when building the graph. A sparse matrix is used for this, where each edge of the graph corresponds to an entry in the matrix. Transport is performed by multiplying the matrix with a vector of radiance values. The final transported radiance values are stored in the graph, which now acts as a radiance cache.
4. **Rendering** The radiance cache is transported to the camera by performing MC integration for each pixel's ray. For each sampled scattering point which is within search ranges of graph nodes, those node's radiance values are used to determine the value of the sample.

The sequence of algorithm steps is illustrated in Figure 4.1, and each step is explained in more detail later in this chapter.

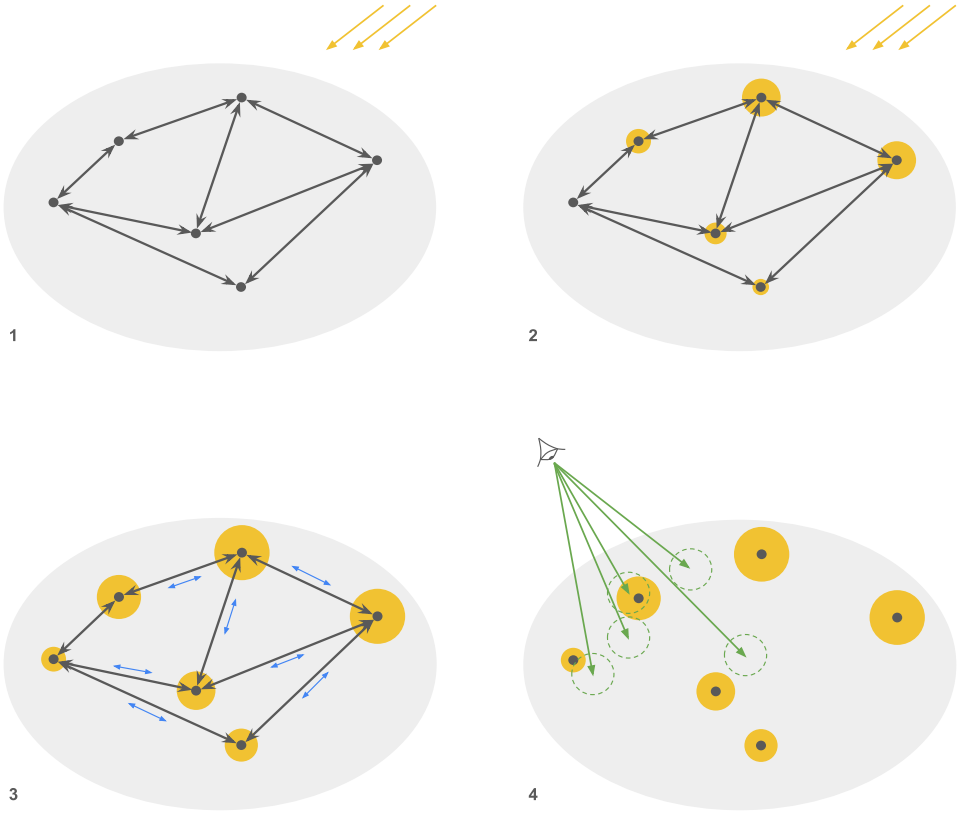


Figure 4.1: Overview of the four algorithm steps. Note that all edges are bidirectional for this example, which indicates that for each pair of nodes edges, for both directions have been sampled. The yellow circles indicate the quantity of radiance in each node. Blue arrows indicate radiance transport, green arrows are camera rays.

4.1. Graph Structure

The graph is the core of the method and forms a bridge between the four steps of the algorithm. It consists of nodes and edges, each node represents a region of the volume. By default, the shape of each region is a sphere with the same radius r, which the user predetermines. One exception to the default shape exists, which is explained in Section 4.3.1. The values stored in a node are the average values of its region. Edges connect nodes, and are directional. A KD-tree contains all the node positions in the graph, which allows for efficient spatial queries on the graph. All stages of the algorithm, including rendering, use the KD-tree.
During graph building, node overlap is not allowed (see Section 4.3.1), which limits the number of nodes in the volume. Duplicate edges are also not possible, which gives an upper bound to the size of the whole graph and, crucially, memory usage.
Nodes store radiance values as scalars of the light's emitted radiance, which requires less space than storing the radiance distribution over the full spectrum. During rendering, the scalars are transformed to full spectral radiance distributions using the light's emitted spectrum. This optimization is possible due to the medium being spectrally constant.

The graph has the following data structure, for which the C++ map structure ($\mathrm{std} :: \mathrm{map}$) and the PBRT 3D point structure ($\mathrm{pbrt} :: \mathrm{Point3f}$) are used:

```
class Graph
 - std::map<int, Node>                    // node id -> node
 - std::map<int, Edge>                    // edge id -> edge

class Node
 - int id
 - pbrt::Point3f position
 - float scattered_radiance_scalar
```

```
 - int outgoing_sample_count
 - float render_search_range
 - std::map<int, int> incoming_edges         // other node id -> edge id
 - std::map<int, int> outgoing_edges         // other node id -> edge id

class Edge
 - int id
 - int from_node_id
 - int to_node_id
 - int sample_count
```

4.2. RTE to Graph Algorithm Equation

In order to prove the correctness of our algorithm, we will start with the RTE as a basis. We then give a definition of the graph algorithm and rewrite the RTE to match the graph algorithm's equation. This proves that our algorithm correctly computes the incoming radiance to the camera.
How the given definitions of the graph algorithm are computed is shown later in this chapter.

4.2.1. RTE Definition

The recursive integral form of the Radiative Transfer Equation (RTE), which was introduced in Section 2.2, is used as a basis. Here we use a definition adapted from Equation 2.4 for a scene without emissive media and one directional light, with an upper bound to path length:

$$L(\mathrm{p},\omega,k) = L_d(\mathrm{p},\omega) + \begin{cases} \int_0^\infty T_r(\mathrm{p},\mathrm{p}')\mu_s(\mathrm{p}') \int_\Omega f(\omega\cdot\omega_i)L(\mathrm{p}',\omega_i,k-1)\mathrm{d}\omega_i\mathrm{d}t & \text{if } k>0 \\ 0 & \text{else} \end{cases} \tag{4.1}$$

where
p' = $\mathrm{p}+t\omega$
k = the maximum depth to evaluate the RTE to

In this equation the incoming radiance from the directional light for each point and direction is modeled as:

$$L_{d,p}(\mathrm{p},\omega) = \delta(\omega-\omega_l)T_r(\mathrm{p},l)L_e \tag{4.2}$$

where
l = the light source
$\delta(\omega)$ = Dirac delta distribution of the directional light
L_e = the emitted radiance value of the light source

The RTE can be used to calculate the incoming radiance to the camera, for all light paths with at most k bounces, by evaluating $L(\mathrm{p},\omega,k)$ for all camera pixels (p is the camera position and ω the direction associated with each pixel).

4.2.2. Graph Algorithm Equation Definition

The graph algorithm divides the full light path from camera to light source into three separate parts, and the value of each part of the light path is computed differently in the algorithm:

1. The first segment from the camera to graph
2. All of the segments in the graph
3. The segment from the graph towards the light

1. The first segment value from the camera is computed with MC integration. Although the camera ray connects to a graph, which contains values discretely distributed in space, the transmittance and scattering coefficient along the ray are integrated continuously. The value for incoming radiance is:

$$L(\mathrm{p},\omega) = \int_0^\infty T_r(\mathrm{p},\mathrm{p}')\mu_s(\mathrm{p}')L'[\mathrm{p}']\mathrm{d}t \tag{4.3}$$

where
$L'[\mathrm{p}']$ = radiance in the graph at point p (exact definition given at the end of this subsection)

2. For each segment in the discrete graph, when randomly sampling a next point from a point in node a, there is a probability of sampling a point in node b. This probability, which consists of sampling a direction and a distance, is computed for each segment.
The probability $p_{\mathrm{dir}}(a,b)$ of sampling a direction from a point in a which can intersect the sphere of b is equal to the solid angle of that sphere, divided by the solid angle of all possible directions.

$$p_{\mathrm{dir}}(a,b) = \frac{2\pi\left(1-\sqrt{1-r^2/d^2}\right)}{4\pi} \tag{4.4}$$

where
r = radius of sphere of b
d = distance from point in a to center of sphere of b

The probability $p_{\mathrm{dist}}(a,b)$ of sampling from a point in a a distance within the sphere of b consists of sampling no scattering event towards the sphere and then sampling a real-scattering event within the sphere. The probability of no scattering event occurring over a distance is the definition of transmittance.
The probability of a real-scattering event occurring within the sphere is the integral over the distance in b, of the transmittance to a point and scattering coefficient in that point. Assuming the volume of a node is constant, this integral can be worked out to a simple function.

$$p_{\mathrm{dist}}(a,b) \approx e^{-\int_0^{d-\frac{2}{3}r}\mu_t(\mathrm{p}_a+t\omega)\mathrm{d}t} \cdot \frac{\mu_s(b)}{\mu_t(b)}\left(1-e^{-\mu_t(b)\frac{4}{3}r}\right) \tag{4.5}$$

where
r = radius of sphere of b
d = distance from point in a to center of sphere of b
$d-\frac{2}{3}r$ = average distance from point in a to boundary of sphere of b (see Appendix C)
p_a = point in a
ω = direction of the ray
$\mu_s(b)$ = the scattering coefficient for the constant volume of b
$\frac{4}{3}r$ = the average distance in direction ω through the sphere of b (see Appendix C)

The two sampling operations with a probability for creating discrete segments are defined for a point in a to sphere b, the value for a segment is defined as the average sampling probability of all possible points in a to sphere b. Now $w(a,b)$ is the function that returns the segment value from node a to b.

$$w(a,b) \approx p_{\mathrm{dir}}(a,b) \cdot p_{\mathrm{dist}}(a,b) \tag{4.6}$$

3. The last segment in each path is always directed towards the light source, so the value of this segment is the phase function for the direction change and the transmittance towards the distant light. It is computed per node, as the average from multiple points in the node to the light. The value stored in each node is this segment value times the emitted radiance of the distant light source, and can be seen as incoming radiance.

$$L_{d,n}(a) = \frac{1}{4\pi}T_r(a,l)L_e \tag{4.7}$$

where
$T_r(a,l)$ = transmittance from node a towards the light source

With definitions for all three types of segment, we can define a recursive equation of the graph algorithm. A transport operator **T** is defined, for one step of light transport in the graph:

$$\mathbf{T} = \begin{bmatrix} w(1,1) & w(1,2) & \dots & w(1,n) \\ w(2,1) & w(2,2) & \dots & w(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ w(n,1) & w(n,2) & \dots & w(n,n) \end{bmatrix} \tag{4.8}$$

This operator works on the vector $\mathrm{L}_{d,n}$, which contains the $L_{d,n}(a)$ value for all nodes a. Computing $\mathbf{T} \cdot \mathrm{L}_{d,n}$ is equal to for each node computing the total transported $L_{d,n}(a)$ from all other nodes, i.e. one bounce of lighting. We use a Neumann series to compute the sum of k bounces of lighting. The process of computing $k-1$ steps of transport in the graph results in a vector of scattered radiance, and can be written as:

$$\mathrm{L}' = \left(\sum_{i=0}^{k-1} \mathbf{T}^i \right) \mathrm{L}_{d,n} \tag{4.9}$$

where
$\mathrm{L}_{d,n}$ = the vector of $L_{d,n}(a)$ values for all nodes a

We use the following notation to indicate the retrieval of the node value for point p stored in L': $\mathrm{L}'[\mathrm{p}]$. The equation for computing the radiance for a camera ray is now complete and becomes:

$$L(\mathrm{p}, \omega, k) = \int_0^\infty T_r(\mathrm{p}, \mathrm{p}') \mu_s(\mathrm{p}') \mathrm{L}'[\mathrm{p}'] \, \mathrm{d}t \tag{4.10}$$

where
k = the maximum number of light bounces

Note that this definition is only valid for $k > 0$, but this is not a problem as paths without any bounces do not occur in our scene (camera rays cannot intersect a distant light).

4.2.3. Linking the Two Equations

Given the RTE and graph definitions of $L(\mathrm{p}, \omega, k)$, we will rewrite the RTE definition to match the graph definition in form. The RTE is expanded once recursively, and since camera rays cannot intersect a distant light, the first $L_d(\mathrm{p})$ can be dropped. The phase function can be replaced with its isotropic constant value. This allows us to split the definition of the RTE into incoming radiance $L(\mathrm{p}, \omega, k)$ and scattered radiance $L'(\mathrm{p}, k)$, with the latter dropping the direction parameter. The new definitions become:

$$L(\mathrm{p}, \omega, k) = \int_0^\infty T_r(\mathrm{p}, \mathrm{p}') \mu_s(\mathrm{p}') L'(\mathrm{p}', k) \mathrm{d}t \tag{4.11}$$

$$L'_p(\mathrm{p}, k) = \int_\Omega \frac{1}{4\pi} \left(L_d(\mathrm{p}, \omega_i) + \int_0^\infty T_r(\mathrm{p}, \mathrm{p}') \mu_s(\mathrm{p}') L'_p(\mathrm{p}', k-1) \mathrm{d}t \right) \mathrm{d}\omega_i \tag{4.12}$$

Working out the multiplication in $L'_p(\mathrm{p}, k)$ gives:

$$L'_p(\mathrm{p}, k) = \int_\Omega \frac{1}{4\pi} L_d(\mathrm{p}, \omega_i) \mathrm{d}\omega_i + \int_\Omega \int_0^\infty \frac{1}{4\pi} T_r(\mathrm{p}, \mathrm{p}') \mu_s(\mathrm{p}') L'_p(\mathrm{p}', k-1) \mathrm{d}t \mathrm{d}\omega_i \tag{4.13}$$

The first term contains an integral over Ω which cancels out with the Dirac delta distribution in $L_{d,p}(\mathrm{p}, \omega)$. The second term contains the double integral over all directions and distances, which we rewrite to a sum over all discrete nodes in space. Each node **n** is defined to be a sphere with equal size, with constant properties in its volume, and no overlap between nodes is assumed for this derivation. By splitting the double integral into a sum of nodes, for each node now only a part of the full double integral has a contribution:

$$L'_p(\mathrm{p}, k) = \frac{1}{4\pi} T_r(\mathrm{p}, l) L_e + \sum_{\mathbf{n}}^{\mathcal{N}} \int_{\Omega(\mathbf{n})} \int_{d_1}^{d_2} \frac{1}{4\pi} T_r(\mathrm{p}, \mathrm{p}') \mu_s(\mathrm{p}') L'_p(\mathrm{p}', k-1) \mathrm{d}t \mathrm{d}\omega_i \tag{4.14}$$

where
$\mathcal{N}$ = the set of discrete regions
$\Omega(\mathbf{n})$ = the set of all directions towards the node **n**
d_1, d_2 = the minimum and maximum distance from point p in direction ω_i for which point p' is in **n**

We now have a discrete formulation of transport, with a double integral for each node. Each node is assumed to have a constant volume, so the double integral can be worked out to a simple formula.

- The integral over relevant directions $\Omega(\mathbf{n})$ becomes the average value for those directions, times the area of the directions, which is the solid angle of $\Omega(\mathbf{n})$.
- The integral over relevant distances from d_1 to d_2 is simplified by assuming no loss from absorption in the node. This allows us to split the integral into two parts: transmittance to the node, and transmittance times scattering coefficient in the node.

  For the average transmittance of directions, we use the transmittance of just one segment from p in direction to the center of the node, with the length being the average distance from p to the node boundary of the directions.
  As the coefficients are constant in the node, the average of the integral in the node for all directions becomes one integral over the average distance in the node. The integral over that distance x of the transmittance to each point times μ_s in that point can be worked out:

$$\int_0^x e^{-\mu_t(\mathbf{n})t}\mu_s(\mathbf{n})\mathrm{d}t = \frac{\mu_s(\mathbf{n})}{\mu_t(\mathbf{n})}\left(1 - e^{-\mu_t(\mathbf{n})x}\right)$$

The value of the double integral for one point to a node becomes:

$$w_p(\mathrm{p},\mathbf{n}) = \frac{2\pi\left(1-\sqrt{1-r^2/d^2}\right)}{4\pi} \cdot e^{-\int_0^{d-\frac{2}{3}r}\mu_t(\mathrm{p}+t\omega_{\mathrm{p},\mathbf{n}})\mathrm{d}t} \cdot \frac{\mu_s(\mathbf{n})}{\mu_t(\mathbf{n})}\left(1 - e^{-\mu_t(\mathbf{n})\frac{4}{3}r}\right) \tag{4.15}$$

where
r = radius of **n**
d = distance from p to center of **n**
$d - \frac{2}{3}r$ = average distance from point in a to boundary of sphere of b (see appendix C)
$\omega_{\mathrm{p},\mathbf{n}}$ = direction from p to the center of **n**
$\mu_s(\mathbf{n})$ = the scattering coefficient for the constant volume of b
$\frac{4}{3}r$ = the average distance in direction ω through the sphere of b (see appendix C)

Now we transform to discrete space by defining $w_n(\mathrm{n},\mathbf{n})$ and $L'_n(\mathrm{n},k)$ to contain the average value of $w_p(\mathrm{p},\mathbf{n})$ and $L'_p(\mathrm{p},k)$ for all points in n, respectively. The definition of the node to node weight $w_n(\mathrm{n},\mathbf{n})$ is equal to that of the graph's node to node weight $w(a,b)$ (Eq. 4.6). The definition of $L'_n(\mathrm{n},k)$ becomes:

$$L'_n(\mathrm{n},k) = \frac{1}{4\pi}T_r(\mathrm{n},l)L_e + \sum_{\mathbf{n}}^{\mathcal{N}} w_n(\mathrm{n},\mathbf{n})L'_n(\mathbf{n},k-1) \tag{4.16}$$

where
$T_r(\mathrm{n},l)$ = the average transmittance from all points in n to the light source

Given the function $L'_n(\mathrm{n},k)$, we can rewrite it to vector notation which operates on all nodes in discrete space at once:

$$\mathrm{L}'_n = \mathrm{L}_{d,n} + \mathbf{T}\cdot\mathrm{L}'_n \tag{4.17}$$

where
L'_n = vector of $L'_n(\mathrm{n},k)$ values for all nodes n
$\mathrm{L}_{d,n}$ = vector of $L_{d,n}(\mathrm{n})$ (Eq. 4.7) values for all nodes n
T = matrix of $w_n(\mathrm{n},\mathbf{n})$ values for pairs of nodes $(\mathrm{n},\mathbf{n})$

The new vector equation that we have defined is a Neumann series, i.e., it is a recursive function which starts with an initial vector ($\mathrm{L}_{d,n}$ in our case), and a 2D-matrix (our transport matrix) is applied to it in each recursive step. It can be rewritten in non-recursive form for k bounces of light as follows:

$$\mathrm{L}'_n = \left(\sum_{i=0}^{k-1}\mathbf{T}^i\right)\mathrm{L}_{d,n} \tag{4.18}$$

The final step of the RTE rewrite is to use the vector L'_n in the definition of $L(\mathrm{p},\omega,k)$. For this we use the following notation: $\mathrm{L}'_n[\mathrm{p}]$, which indicates the retrieval of the value for p stored in L'_n. Substituting

the L'_n vector into the $L(\mathrm{p}, \omega, k)$ definition gives the final rewritten RTE definition:

$$L(\mathrm{p}, \omega, k) = \int_0^\infty T_r(\mathrm{p}, \mathrm{p}')\mu_s(\mathrm{p}')\mathrm{L}'_n[\mathrm{p}']\mathrm{d}t \tag{4.19}$$

We have now reached a discrete definition of the continuous RTE, which is equal to the definition of the discrete graph algorithm.

4.3. Building the Graph

The graph is built by tracing paths through the volume, where each path has (potentially) several segments and scattering points. Paths are traced one at a time, at each scattering point it is checked if the point is in the volume of another node or not. If it is, the segment of the path is connected to the closest existing node. If not, a new node is created at the scattering point. Figure 4.2 visualizes this process. Note that all graph nodes must be in the volume, therefore path segments with one end outside the volume are not captured in the graph.

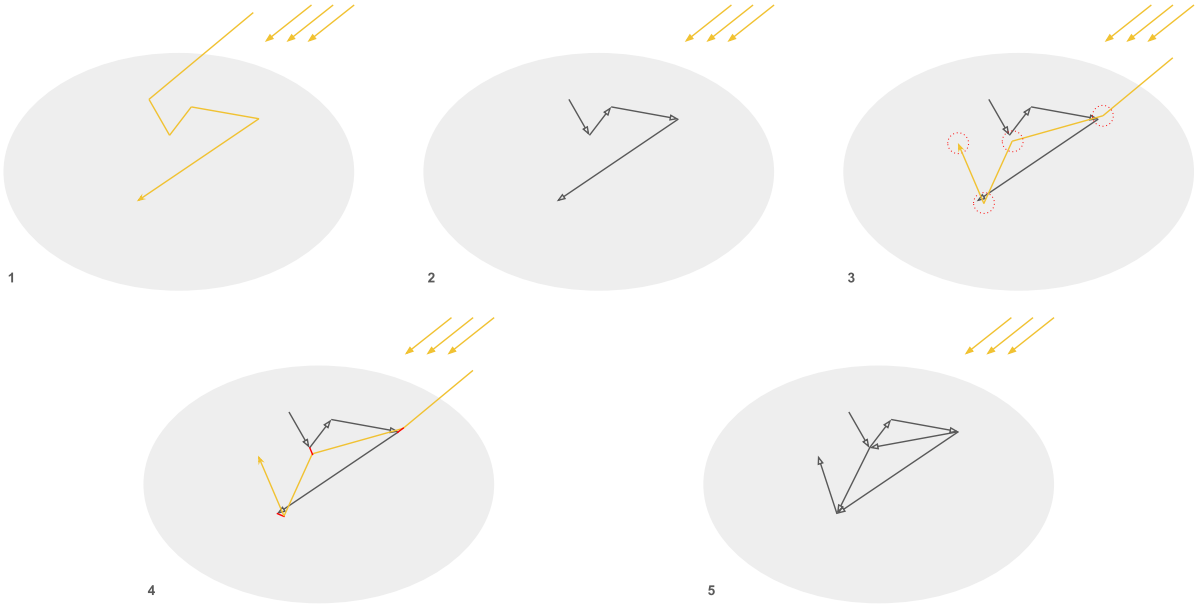
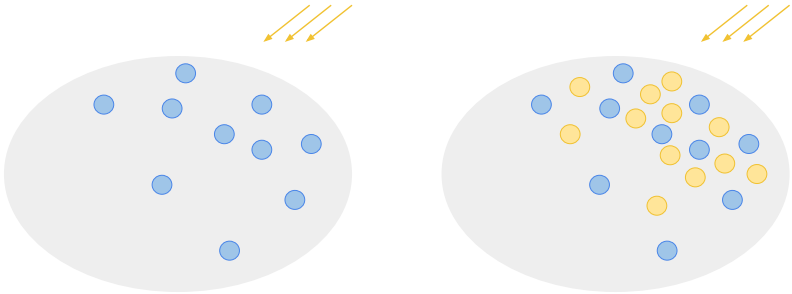


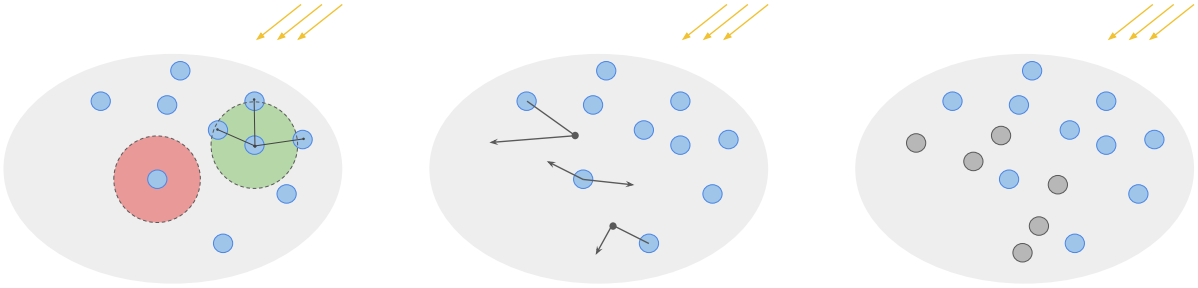
Figure 4.2: Process of building the graph by grouping traced paths, from top left to bottom right. The first traced path is captured in the graph without grouping. The second traced path finds nearby graph nodes at scattering points, and its scattering points are grouped to existing graph nodes.

The goal when building the graph is to get a graph that covers all regions of the volume with a non-zero contribution and has an even density throughout the regions it covers. A complete graph satisfies these criteria, but contains much more data than required to produce a good approximation of the volume. Instead, we follow a more efficient two-stage approach for graph building. In the first stage, paths are traced from the light source. These paths have a high contribution, and a large part of the contributing region of the volume is densely filled in this stage. Some regions of the volume can be hard to reach from the light, which makes increasing their density starting from the light source inefficient. Therefore, in the second stage, each node is reinforced by tracing rays from them, until their density is high enough.

There are two density requirements: number of nearby neighbors and number of outgoing edges. Nearby neighbors ensure there are enough nodes in the graph, whilst outgoing edges ensure there are enough edges in the graph. A simple comparison of ensuring neighbor density by tracing from the light source versus tracing from sparse nodes is given in Figure 4.3.



(a) Reinforcing the graph by tracing more rays from the light source, resulting in an uneven spread of nodes. Existing nodes are blue, new nodes from the light source are yellow.



(b) Reinforcing the graph by tracing rays through sparse regions, resulting in a more uniform spread of nodes. Existing nodes are blue, new nodes from reinforcing are gray. The node with a red search area does not find any neighbors, whilst the node with a green search area finds enough neighbors.

Figure 4.3: Two methods to reinforce a graph.

During path tracing, when sampling a new segment from point a, three options are possible:

1. A scattering event is sampled at a new point b, and a path segment is created between points a and b.
2. An absorption event is sampled, terminating the path.
3. No event is sampled due to the path leaving the volume, which terminates the path.

When a new segment sample occurs from a point in a node, the sample is added to the node's sample count. The number of times a segment between nodes is sampled is also counted.
Using the sample counts for edges and nodes, a weight can be determined for each edge. This sample-based edge weight is defined in Equation 4.20, and is an approximation of the exact edge weight given in Equation 4.6.

$$w_s(a,b) = \frac{s_e(a,b)}{s_n(a)} \tag{4.20}$$

where
a, b = nodes in the graph
$s_e(a)$ = number of samples of the edge from node a to node b
$s_n(a)$ = number of samples out of node a

The traced paths that are grouped into the graph are sampled according to the sampling PDF, which results in the samples out of a node being an unbiased estimate of the radiance transport to that node. The accuracy of the estimate depends on the number of samples in the graph, which is a reason why we aim to build a dense graph as discussed earlier in this chapter.

4.3.1. Shape of Node Regions

In Section 4.2 we define equations that represent the graph. The weight of an edge is the node to node sampling probability, the shapes of nodes are spheres. After defining the graph equations, we take the continuous RTE definition and rewrite it to discrete form. For the discrete RTE the volume is perfectly discretized into nodes, and the node shapes are spheres too. One of rewritten RTE forms is

Equation 4.16, which defines scattered radiance at a node. The term for indirect radiance is a sum over all nodes, for each node that node's weighted scattered radiance is taken. The weight is exactly equal to the edge weight of the graph, which is the sphere to sphere sampling probability. This shows that to determine scattered radiance, the space must be divided into regions such that for each node the node to node sampling probabilities sum to that node's total probability of sampling a new point. Any perfect discretization with correct node to node sampling probabilities will satisfy this. Spheres were chosen as node shapes for the discrete RTE as it gives a relatively simple function for the node to node sampling probability.

The graph's definition is shown to be equal to a discrete version of the RTE in Section 4.2, if a perfect discretization with spheres is made. Since this is impossible in practice, the graph uses spheres when possible for node shapes, but if node overlap occurs, the conflicting region is split between the two nodes. Each point in that region is assigned to the node with the closest center, which results in the nodes becoming pseudo-Voronoi cells. This is visualized in figure 4.4.

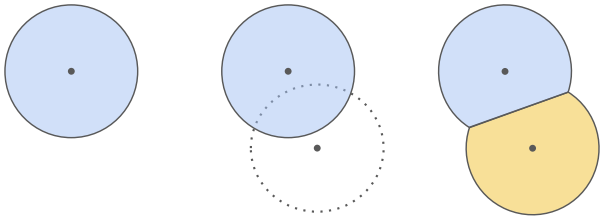


Figure 4.4: The three stages of forming a pseudo Voronoi cell, from left to right. At first, there is only the initial node. A second node center is found, which is not in the region of another node. Finally, the nodes' regions are evenly split such that they do not overlap.

The use of pseudo-Voronoi cells causes the graph implementation to not match the graph definition given in Section 4.2. Therefore, the defined graph edge weights cannot be used, but they would never be used regardless of node shape as edge weights are always approximated in the implementation, as defined in Equation 4.20. The use of pseudo-Voronoi cells gives a perfect discretization of the volume, and the node to node weights converge to the correct probability (Eq. 4.20 is used). As stated earlier in this section, these are the two criteria needed to be able to correctly determine the scattered radiance at a node. Since the rest of the graph algorithm is equal to the given definitions, it remains equal to the RTE despite not using spheres as node shapes.

4.3.2. Calculating Render Search Ranges

At this stage, the graph has been built, with many nodes spread throughout the volume. Although we have tried to make the graph fill all contributing regions of the volume by reinforcing it, it is still possible for contributing regions to not be covered by a graph node. During the rendering stage, no node and thus no radiance can be found in those regions, but there should be radiance present. This leads to negative bias in the rendering stage.
To prevent this bias, a separate search range is introduced for each node, which becomes the node's new sphere radius for the rendering stage. The goal is to set it such that any contributing point in the volume is in at least one node. Intuitively, this can be seen as inflating the size of the nodes until they completely fill the contributing part of the volume.

For the render search range computation, two functions are defined: $\mathrm{avgD}(a)$ gives the average distance to k neighbors for node a, and $\mathrm{searchR}(a)$ gives the rendering search range for node a. The number of neighbors k is the same for all nodes and is set by the user before precomputation begins.

$$\begin{aligned} \mathrm{avgD}(a) &= \frac{\sum_{i=1}^{k} D(a, \mathrm{NN(a, i)})}{k} \\ \mathrm{searchR}(a) &= \frac{\sum_{i=0}^{k} \mathrm{avgD(NN(a, i))}}{k+1} \end{aligned} \tag{4.21}$$

where

k = the number of neighbors to use
$\mathrm{NN}(a, 0)$ = node a
$\mathrm{NN}(a, i)$ = the i'th nearest neighbor of a
$D(a, b)$ = distance between node a and b

We do not simply use the average distance to the k nearest neighbors as the search range. This is because the node distribution in the graph can be irregular or sparse, and simply increasing node size to cover all space between them could result in covering non-contributing regions of the volume with a non-zero radiance value node.
Instead, the average of the node a and the k nearest neighbors' average distance to their k nearest neighbors is used. This is a trade-off between filling gaps that should contain radiance and leaving non-contributing regions empty. The value stored in each node a is $\mathrm{searchR}(a)$.

4.4. Computing Direct Radiance

Radiance in a node is stored as a scalar of the light's emitted radiance (see Section 4.1). Therefore we compute Equation 4.7 for each node without L_e, which becomes:

$$L_{d,n}(a) = \frac{1}{4\pi} T_r(a, l) \tag{4.22}$$

where
a = a node
$T_r(a, l)$ = the transmittance from node a to the light source

To compute the transmittance from a node to the light source, ratio tracking is used. The resulting value of ratio tracking a segment from the node to the light source is approximately the required transmittance. To get a more accurate estimate of transmittance, the average transmittance of multiple segments starting from different points in the node is used. The points are distributed evenly over a disk in the node, the disk is perpendicular to the light direction.

4.5. Transporting Radiance

After the direct radiance scalar values for the nodes have been computed, they have to be transported through the graph. The graph contains the sampled edge weights $w_s(a, b)$ (Eq. 4.20), which we put in a 2D matrix $\mathbf{T}$. The layout of the matrix is given in Equation 4.8. The $L_{d,n}(a)$ direct radiance values are put in a vector $\mathrm{L}_{d,n}$.
One step of radiance transport can be computed with $\mathbf{T} \cdot \mathrm{L}_{d,n}$, but we need the contribution from paths of many different lengths. The length of a path is defined as its number of bounces k, which can be computed with $k - 1$ transport steps. Formally, the total transported radiance is the sum of transported radiance for all possible lengths up to and including k, which is a Neumann series:

$$\mathrm{L}'_n = \underbrace{\mathrm{L}_{d,n}}_{\text{length 1}} + \underbrace{\mathbf{T} \cdot \mathrm{L}_{d,n}}_{\text{length 2}} + \underbrace{\mathbf{T}^2 \cdot \mathrm{L}_{d,n}}_{\text{length 3}} + \cdots + \underbrace{\mathbf{T}^{k-1} \cdot \mathrm{L}_{d,n}}_{\text{length } k} \tag{4.23}$$

This definition is only valid if $k > 0$, i.e. paths must have at least one bounce. This is because paths without a bounce cannot occur in our scene, as rays from the distant light cannot intersect with the camera.
In practice, the Neumann series is computed iteratively, as shown by the pseudo-code of Algorithm 1.

Although $\mathbf{T}$ is extremely large, it is also very sparse, so multiplying it with a vector remains efficient. Raising $\mathbf{T}$ to a power, as done in the Neumann series, reduces the sparsity and therefore the efficiency of computation. The main advantage of the iterative Neumann series computation is that a power of $\mathbf{T}$ never has to be used, which makes the cost of transporting radiance very small compared to the rest of the algorithm.
The final vector of scalar radiance values L'_n is stored in the nodes of the graph, which have positions in the volume. The graph nodes now form a radiance cache, which can be used for the final rendering step.

Algorithm 1 Iterative computation of L'_n

$\mathrm{L}'_n \leftarrow \mathrm{L}_{\mathrm{d,n}}$
$\mathrm{L}'_{n,\mathrm{bounce\_i}} \leftarrow \mathrm{L}'_n$
$i \leftarrow 1$
while $i < k$ **do**
  $\mathrm{L}'_{n,\mathrm{bounce\_i}} \leftarrow \mathbf{T} \cdot \mathrm{L}'_{n,\mathrm{bounce\_i}}$
  $\mathrm{L}'_n \leftarrow \mathrm{L}'_n + \mathrm{L}'_{n,\mathrm{bounce\_i}}$
  $i \leftarrow i + 1$
end while

4.5.1. Transport Example

Here, a simple example of radiance transport is given. The direction of transport is opposite to the edge direction, as edges are in the direction of the light and radiance is transported away from the light towards the camera. The graph consists of 4 nodes, which are connected as follows:

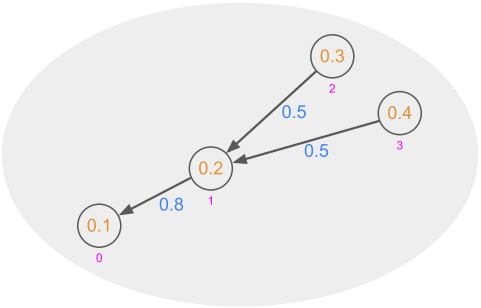


Figure 4.5: Graph of this section's illustrative example. The purple numbers are node indices, the yellow numbers in the nodes are L_n values, the blue values correspond to the edge weights of **T**.

From this graph the direct radiance vector $\mathrm{L}_{d,n}$ and transport matrix **T** are taken:

$$\mathrm{L}_{d,n} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.8 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}$$

The transported radiance vector L'_n can then be computed by working out the Neumann series, which for this example is done up to path length 3:

$$\begin{aligned} \mathrm{L}'_n = \quad & \mathrm{L}_{d,n} + && \mathbf{T} \cdot \mathrm{L}_{d,n} + && \mathbf{T} \cdot (\mathbf{T} \cdot \mathrm{L}_{d,n}) \\ = \quad & \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} + && \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.8 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} + && \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.8 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} \cdot (\mathbf{T} \cdot \mathrm{L}_{d,n}) \\ = \quad & \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} + && \begin{bmatrix} 0 \\ 0.08 \\ 0.1 \\ 0.1 \end{bmatrix} + && \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.8 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.08 \\ 0.1 \\ 0.1 \end{bmatrix} \\ = \quad & \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} + && \begin{bmatrix} 0 \\ 0.08 \\ 0.1 \\ 0.1 \end{bmatrix} + && \begin{bmatrix} 0 \\ 0 \\ 0.04 \\ 0.04 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.28 \\ 0.44 \\ 0.54 \end{bmatrix} \end{aligned}$$

4.6. Rendering

All of the previous steps of the algorithm are performed in a precompute phase, resulting in a radiance cache containing radiance scalars of the scene's light source. The cache consists of spherical nodes, each with a pre-calculated search range (Section 4.3.2), virtually increasing their size to fill the volume. A node represents the average of its region and can overlap with other nodes.
To transport the radiance cache to the camera, Equation 4.10 is slightly modified to include the emitted radiance, which gives:

$$L(\mathrm{p},\omega) = L_e \int_0^\infty T_r(\mathrm{p},\mathrm{p}')\mu_s(\mathrm{p}')L'_{\mathrm{search}}(\mathrm{p}')\mathrm{d}t \tag{4.24}$$

where
L_e = the full spectrum of radiance emitted by the light source
$\mathrm{p}' = \mathrm{p} + t\omega$
$T_r(\mathrm{p},\mathrm{p}')$ = transmittance from point p to p'
$\mu_s(\mathrm{p})$ = scattering coefficient at point p

Monte Carlo integration is used to solve the integral over the camera rays, as it gives an accurate estimate whilst evaluating only a fraction of all the possible segments. Once a scalar is determined by evaluating the segment, it is multiplied by the full spectral radiance distribution to get the true radiance value, which is used for determining the pixel color.

The radiance scalar is gathered from the cache by evaluating $L'_{\mathrm{search}}(\mathrm{p}')$ at point p'. It is defined as the weighted average radiance scalar of all nodes for which p' is within their search range, the weight of each value is its reciprocal squared node distance to p'. The formal definition is:

$$L'_{\mathrm{search}}(\mathrm{p}') = \frac{\sum_{\mathcal{C}} D(\mathrm{p}',\mathrm{p}_c)^{-2} \cdot \mathrm{L}'_n[\mathrm{p}_c]}{\sum_{\mathcal{C}} D(\mathrm{p}',\mathrm{p}_c)^{-2}},\ \mathrm{p}_c \in \mathcal{C} \text{ where } D(\mathrm{p}',\mathrm{p}_c) < \mathrm{searchR}(\mathrm{p}_c) \tag{4.25}$$

where
$\mathcal{C}$ = the set of nodes in the radiance cache

4.7. Bias of Graph Approximation

A node contains the average direct radiance scalar of its sphere. Given radiance that is distributed spatially varying throughout a volume, the average radiance scalar of a sphere is correct, but will be locally different from the true radiance distribution.
Take, for example, the direct radiance distribution throughout a homogeneous volume, which decreases homogeneously in the direction of the light. It is assumed that μ_s is 1, and that μ_a and μ_e are 0. The difference between the true radiance distribution and a node's radiance distribution lies in the approximation of transmittance in the node, which is a constant. This is illustrated in Figure 4.6. This transmittance approximation results in bias when a node's radiance value is transported to a receiver by integrating a ray over the node's radiance distribution.
The receiver can be any point or object, for this analysis we assume it is the camera. The bias analyzed in this section is for light and camera rays of equal direction, analysis for other direction combinations can be found in Appendix A.

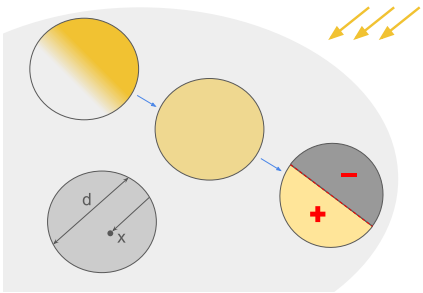
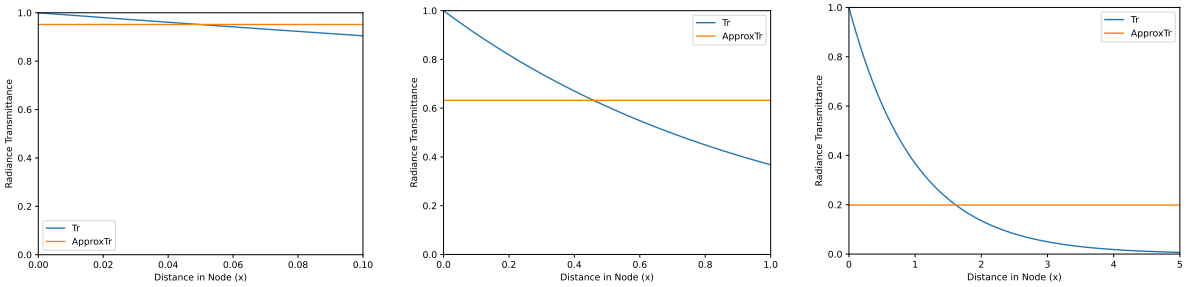


Figure 4.6: Process and consequences of the averaging of transmittance in a node is shown in the sequence of three nodes. Compared to the true radiance distribution, the approximation is too low near light entry and too high further away from light entry. The bottom left node visualizes the d and x terms.

In the plots in the section, both camera and light ray are conceptually to the left of the plot, with the node entry point being exactly on each plot's left boundary.
Let the diameter of the node be d, the distance from the node's entry point to another point further along the light direction be x, and $0 \leq x \leq d$. Then the transmittance from the entry point towards a point at distance x is called Tr, and it's approximation in a node is called $\mathrm{ApproxTr}$. Their definitions are:

$$\begin{aligned} \mathrm{Tr}(x) &= e^{-x} \\ \mathrm{ApproxTr}(d) &= \frac{1 - e^{-d}}{d} \end{aligned} \tag{4.26}$$

Tr is the homogeneous transmittance equation. $\mathrm{ApproxTr}$ is the constant value held in the node, which is the average value of Tr over d. It is defined as the integral of Tr over d divided by d.
Plotting Tr and $\mathrm{ApproxTr}$ for x, with d being the maximum x value of each plot (Figure 4.7), gives the following transmittance graphs:



(a) Tr and $ApproxTr$ up to 0.1 **(b)** Tr and $ApproxTr$ up to 1 **(c)** Tr and $ApproxTr$ up to 5

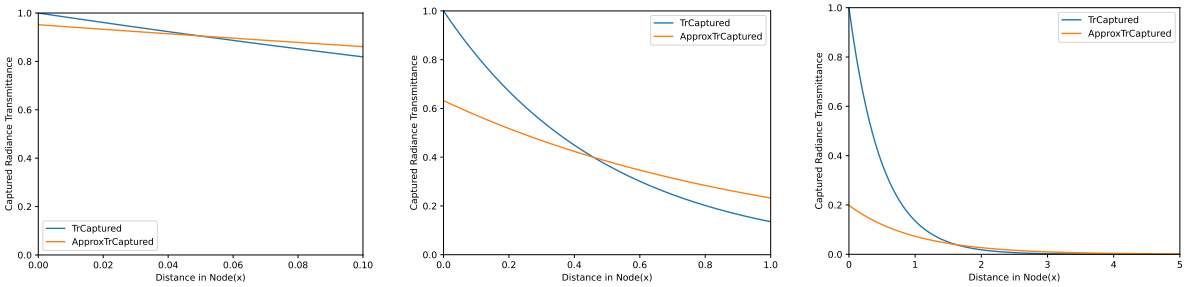
Figure 4.7: Transmittance and the node approximation over increasing distances. Although the average value of the two function over the range remains equal, the average difference increases.

The average transmittance for the domain remains correct regardless of node size d, but the average absolute difference increases as node size d increases.

To transport the radiance distribution towards the camera, the true transmittance and the node's transmittance are multiplied by the camera ray's transmittance. The camera rays are distributed over x by the homogeneous transmittance equation, multiplying it with Tr and $\mathrm{ApproxTr}$ results in the following:

$$\begin{aligned} \mathrm{TrTransported}(x) &= e^{-x} \cdot e^{-x} = e^{-2x} \\ \mathrm{ApproxTrTransported}(x, d) &= e^{-x} \cdot \frac{1 - e^{-d}}{d} \end{aligned} \tag{4.27}$$

Plotting these equations (Figure 4.8) gives the transmittance from the node's entry point to x and back to the entry point, which is the scalar of radiance reaching the node that is transported back out of the node. The plots show this value for single points, with d being the maximum value of x for each plot:



(a) $\mathrm{TrTransported}$ and $\mathrm{ApproxTrTransported}$ up to 0.1 **(b)** $\mathrm{TrTransported}$ and $\mathrm{ApproxTrTransported}$ up to 1 **(c)** $\mathrm{TrTransported}$ and $\mathrm{ApproxTrTransported}$ up to 5

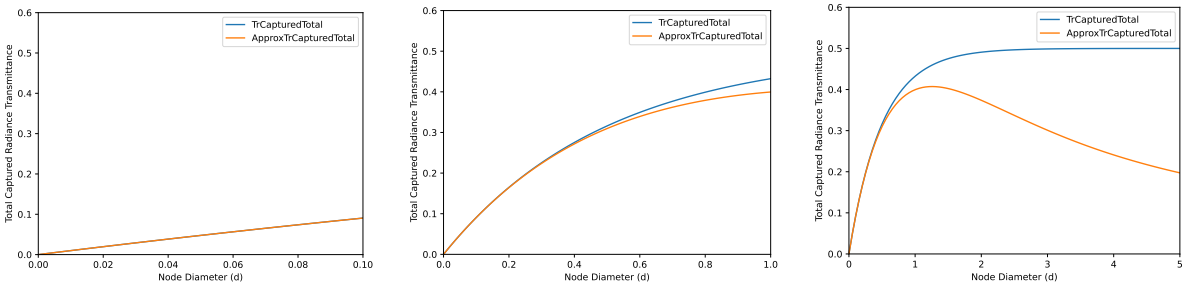
Figure 4.8: Transported radiance scalar and the node approximation over varying distances. Due to the introduction of radiance transport, the average of the two functions over the range is no longer equal.

Although for small node sizes the average error between the true radiance transported and the approximation is small, for bigger node sizes the transported approximation becomes significantly too small near the node's entry point.

The definition of the total radiance scalar transported to the camera is the integral of Equations 4.27 from 0 to d.

$$\begin{aligned} \mathrm{TrTransportedTotal}(d) &= \frac{1}{2}\left(1 - e^{-2d}\right) \\ \mathrm{ApproxTrTransportedTotal}(d) &= \frac{\left(1 - e^{-d}\right)^2}{d} \end{aligned} \tag{4.28}$$

Plots of the true integral's distribution and the approximate integral's distribution for increasing node size d (Figure 4.9) now give values depending only on node size:



(a) $\mathrm{TrTransportedTotal}$ and $\mathrm{ApproxTrTransportedTotal}$ up to 0.1

(b) $\mathrm{TrTransportedTotal}$ and $\mathrm{ApproxTrTransportedTotal}$ up to 1

(c) $\mathrm{TrTransportedTotal}$ and $\mathrm{ApproxTrTransportedTotal}$ up to 5

Figure 4.9: Total transported radiance scalar and the node approximation for varying node sizes. The larger the node becomes the larger the error between the true distribution and the approximation.

The plots showing the total transported scalar of incoming radiance at the node's entry point and its node-based approximation, show that the approximation has a negative bias to the true value if the light and camera rays have the same direction.
For other angles between the light and camera rays, the bias is different. If the rays have opposite directions then the bias is positive, if they are perpendicular there is no bias, and the bias shifts gradually as the angle between the rays changes. A more extensive explanation of the bias for these other directions can be found in Appendix A.

4.8. Scene Constraints

At the start of this chapter, it was explained that the properties of the scene are constrained to keep the problem and implementation as narrow as possible. The constraints also serve to keep the graph algorithm's space and time complexity to a minimum. In this section, it is detailed how lifting the constraints affects the algorithm.

- **Non-Emissive Medium** An emissive medium emits light from (most of) its volume. Since light rays can now originate from anywhere, the full volume has to be covered by the graph. For large volumes, this is an issue, as without emission most of the volume is non-contributing and not covered by the graph. Therefore, the presence of emission makes the graph method scale worse for large volumes, from typically $O(n^2)$ to $O(n^3)$, where n is the volume radius.
- **Spectrally Constant Medium** If the medium is spectrally varying, all light behavior depends on its wavelength. In the PBRT framework used for this thesis 470 distinct wavelengths are used. This would require a separate graph for each wavelength in the spectrum, and even when building fewer graphs and interpolating between them, the runtime of the graph-building stage would still take roughly 100 times longer.
- **Isotropic Phase Function** If an anisotropic phase function is used, the outgoing direction of a light ray after scattering depends on the incoming direction. This would no longer allow us to group the outgoing light information of all incoming edges, but instead require us to keep the outgoing information for each incoming edge separately. This results in the transport matrix becoming 3-dimensional and the radiance vector a 2D matrix, considerably increasing the implementation complexity. The amount of data stored would typically remain the same, although spread across an additional dimension.
- **One Distant Light Source** Adding support for different types of light sources takes implementation time, but does not affect the algorithm's cost. More than one light source in the scene requires the graph to be built by tracing rays from many sources, but since the medium is spectrally constant and has an isotropic phase function, all rays can be grouped into the same graph. Direct radiance computation and subsequent transport must be done for each light source, but the radiance distributions through the graph can all be stored in the same graph, as one radiance scalar per light source. The cost of adding more light sources is therefore linear with the number of lights.
- **Scene Consists of One Volumetric Object** The graph built by our method must cover all contributing parts of the scene. Increasing the complexity and size of the scene requires an increasingly larger graph, with a larger amount of traced rays not contributing to the camera view. This drawback affects all methods tracing rays (partially) from the light source. The size of the required graph typically grows linearly with the volume of the scene.

5 Results

5.1. Preliminaries

The renders shown in this paper were produced on a system with an Intel i5-14400F processor and 32GB of RAM. Images were rendered at 128 SPP with the graph integrator, and reference images were rendered at 1024 SPP using the VolPath integrator.
Each volume was rendered by integrating light paths consisting of at most 10 segments, i.e. maximum depth of 10. Unless otherwise specified, $\mu_s = 1m^{-1}$ and $\mu_a = 0m^{-1}$.

Three different objects are used for the results of this section: the Disney cloud, the bunny cloud, and the cube. Both the Disney cloud and bunny cloud have been adapted from the PBRT V4 scenes repository[1]. All scenes only contain the volume and a directional light source.
All renders shown in this paper are re-producible using the source code[2]. For each render an adapted $.\mathrm{pbrt}$ scene description file and a $.\mathrm{json}$ graph algorithm configuration file were used, which are also available[3].

To determine the type of bias of the graph method, it is important to distinguish between positive and negative error. Therefore, three types of error are measured on the images produced by the graph integrator: absolute error, positive error and negative error. Positive and negative errors are mutually exclusive, together they sum to the absolute error. The error is visualized per pixel in the form of error images, and is also available as a mean value over the full image.
Formally, the definitions of the mean value error metrics are:

$$\begin{aligned} MAE &= \frac{1}{n}\sum_{i=0}^{n} |g_i - r_i| \\ MPE &= \frac{1}{n}\sum_{i=0}^{n} \max(g_i - r_i, 0) \\ MNE &= \frac{1}{n}\sum_{i=0}^{n} \max(r_i - g_i, 0) \end{aligned} \tag{5.1}$$

where
n = the number of pixels
g_i = pixel i of the graph image
r_i = pixel i of the reference image

[1] https://github.com/mmp/pbrt-v4-scenes/tree/master
[2] https://github.com/tsvdh/AcceleratedVolRenderer
[3] https://github.com/tsvdh/PbrtScenes

5.2. Best Possible Graph Render vs VolPath Render

To test the best possible result the graph integrator can produce, a graph was built for the Disney cloud until it used all available memory. The node size was set as large as possible to allow for more edges, whilst keeping it below the threshold for visual bias.

Nodes	Edges	Duration	Memory
5.3M	131.8M	27m 17s	25GB

Figure 5.1: Reference and best possible graph render of the Disney cloud.
Volume radius: 51.6m, node radius 0.1m

The figure shows no directly visible difference between the reference and graph solutions. When zoomed in on the graph solution individual vertices can be made out, and the MPE of 2.87e-3 and MNE of 0.78e-3 tell us that there is positive bias present.

5.3. Graph Sparseness

One of the downsides of building the graph with random work is that there is no guarantee a specific region of the volume will be covered by a node. One solution to solve this is the use of Render Search Ranges (also called RSR, see Sections 4.3.2 and 4.6). The effect of using just the node range versus using render search ranges is tested on the Disney cloud.

	Nodes	Edges	RSR (Avg)	MPE	MNE
No RSR	5.3M	61.7M	-	1.66e-3	1.9e-3
RSR	5.3M	61.7M	0.064m	2.33e-3	1.15e-3

where
RSR(Avg) = the average render search range of all nodes

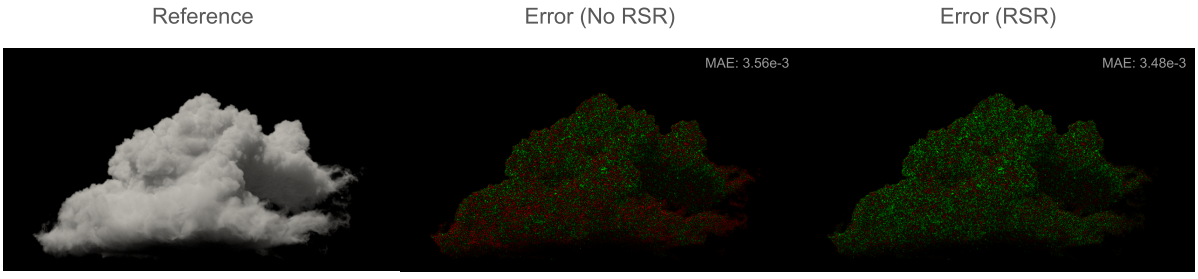


Figure 5.2: Effect of using render search ranges. Positive bias is green, negative bias is red. Error images are 8× brighter.
Volume radius: 25.8m, node radius 0.05m

The figure shows that the negative bias present due to sparseness of the graph is eliminated when using render search ranges, by filling the small gaps left in the graph using an average search range slightly larger than the node radius.

Another solution aimed at solving graph sparseness is graph reinforcing (Figure 4.3b). The effect of graph reinforcing combined with render search ranges is tested on the cube volume.

Reinforcing-RSR	Nodes	Edges	Time	RSR (Avg)	RSR (Std)
No-No	5.2M	21.5M	2m 38s	-	-
No-Yes	5.2M	21.5M	2m 48s	0.049m	0.011m
Yes-No	11M	92.1M	13m 29s	-	-
Yes-Yes	11M	92.1M	13m 30s	0.039m	0.004m

where
RSR(Std) = the standard deviation of all nodes' render search ranges

Reinforcing-RSR	MAE	MPE	MNE
No-No	1.49e-3	0.08e-3	1.41e-3
No-Yes	0.91e-3	0.28e-3	0.63e-3
Yes-No	0.65e-3	0.31e-3	0.33e-3
Yes-Yes	0.66e-3	0.44e-3	0.22e-3

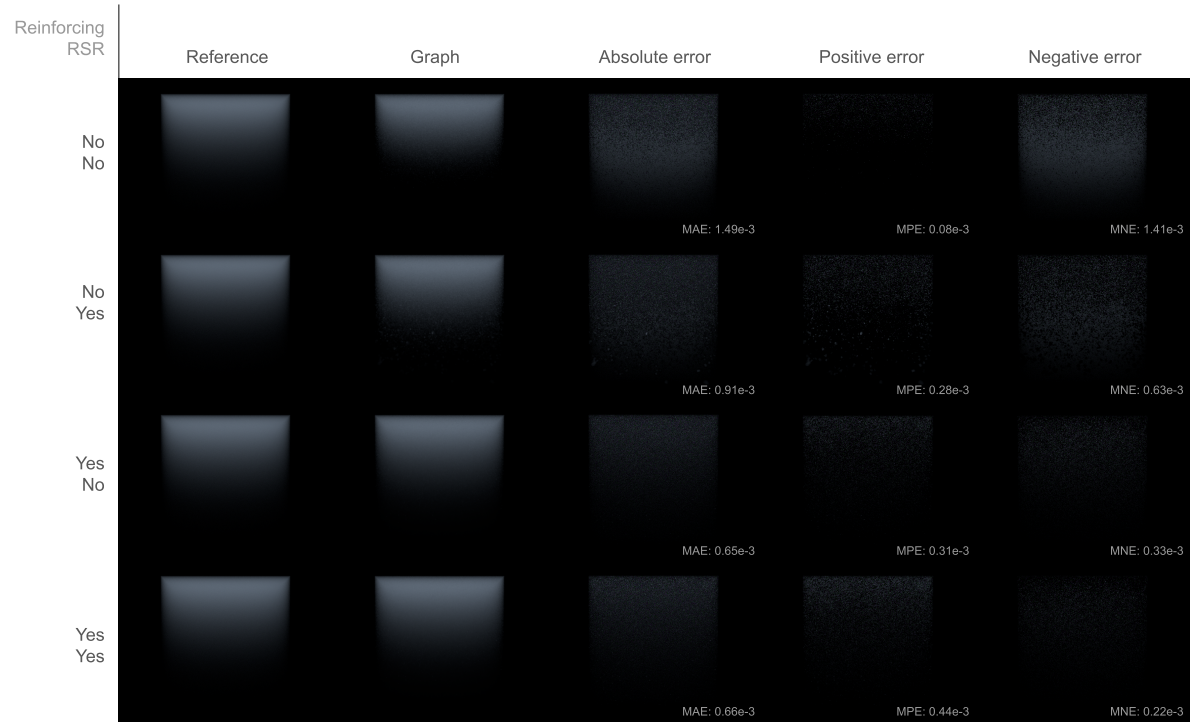


Figure 5.3: Effects of different graph sparseness solving tools. Error images are 4× brighter.
Volume radius: 8.66m, node radius: 0.03m

The figure shows a strong negative bias in the cube without any sparseness solving, which indicates that many camera rays are not finding any nodes in places where radiance should be present. Both tools significantly reduce the bias when applied on their own, but reinforcing is much more effective than render search ranges when applied individually, and produces a smoother image.
Applying both sparseness solvers together produces the least amount of negative bias. When the graph is reinforced, there are fewer empty regions in the volume, which affects the render search ranges to be lower on average and less varying.

5.4. Node and Volume Size

The graph method works best if the node size is small enough so no significant bias is created due to discretization, but still large enough relative to the volume to fill all contributing regions of the volume. To see the effects of the node size being outside this ideal range, firstly the node size is varied and afterwards the volume size.

Node radius	Nodes	Edges	MAE	MPE	MNE
0.1m	15.1M	97.9M	3.9e-3	2.2e-3	1.7e-3
0.2m	3.5M	102.2M	5.7e-3	5.3e-3	0.4e-3
0.5m	333k	16.2M	18.4e-3	18.2e-3	0.2e-3
1.0m	58k	8.4M	38.3e-3	38.7e-3	0.1e-3

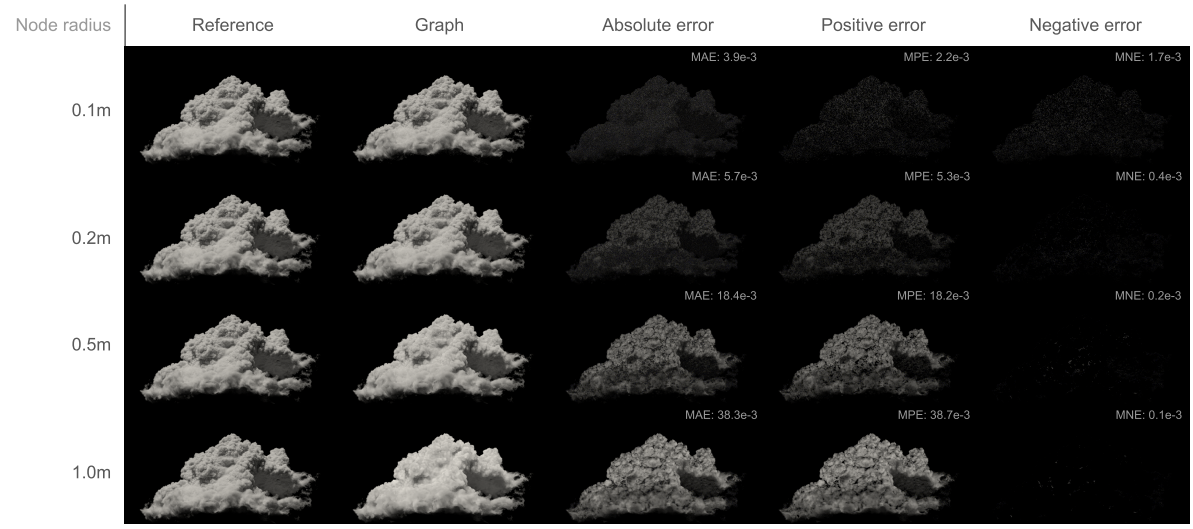


Figure 5.4: Effect of different node sizes with a static volume size.
Volume radius: 103.1m

Increasing the node radius causes the positive bias of the method to increase, and at extreme node sizes the volume becomes blurred due to details of the volume not being able to be captured in the graph.

For this test, the volume size was varied by scaling equally along all axes, but the amount of rays to be traced for graph building was kept constant. No reinforcing was performed during graph building.

Volume radius	Nodes	Edges	VolPath	Graph building	Graph rendering
51.6m	2.7M	33M	10m 25s	5m 31s	29s
103.2m	7.6M	39M	11m 34s	8m 14s	1m 37s
257.9m	21.4M	44.8M	14m 36s	17m 38s	9m 16s
515.8m	32.4M	48.1M	18m 15s	31m 18s	2h 19m 39s

Volume radius	MAE	MPE	MNE
51.6m	4.1e-3	3e-3	1.1e-3
103.2m	4.7 e-3	2.4e-3	2.3e-3
257.9m	9.1e-3	0.9e-3	8.2e-3
515.8m	18.1e-3	0.2e-3	17.9e-3

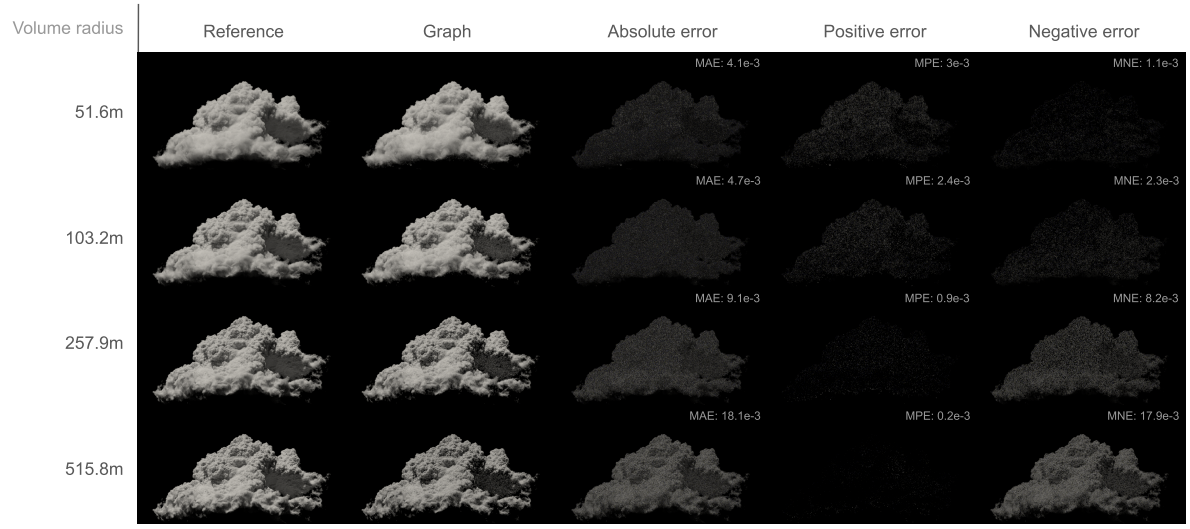


Figure 5.5: Effect of different volume sizes with a static node size.
Node radius: 0.1m

Despite render search ranges being used in this test, there is still a significant negative bias as the graph becomes sparse. This shows that render search ranges become less effective if the empty space to be covered becomes too large and irregular.

The amount of work required scales with volume size, as can be seen by the VolPath rendering times. This is due to fewer rays escaping the volume before maximum depth is reached. Both graph building and graph rendering show interesting trends in their running cost:

- Graph building scales worse than VolPath rendering, which is due to the increasing amount of nodes in the graph. Graph building needs to perform a radius search for nearby nodes at each ray scattering point, which becomes more costly as the number of nodes in the graph increases.
- Graph rendering scales very poorly with volume size, but this is not due to the number of nodes in the graph. Instead, it is due to the small ratio of node size to volume size, and the graph becoming very sparse for the larger volumes. For such sparse graphs, the render search ranges grow large and varying, which makes node search expensive to evaluate.

5.5. Absorption

To see how the graph method handles the possibility of light being absorbed, the real-scattering and absorption coefficients are varied, but the attenuation coefficient remains equal.

μ_a/μ_t	Nodes	Edges	Duration	MAE
0.0	1.3M	22.6M	7m 42s	0.9e-3
0.1	1.2M	17.7M	6m 59s	0.83e-3
0.25	1.1M	11.2M	5m 43s	0.69e-3
0.5	1M	5.9M	6m 42s	0.5e-3

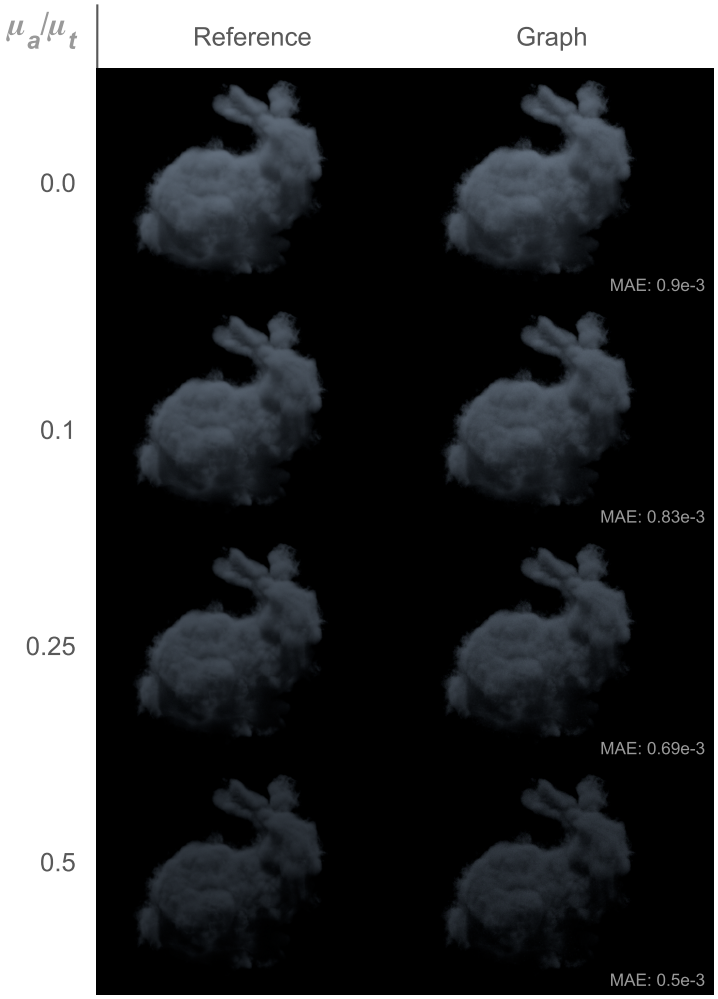


Figure 5.6: Effect of varying medium coefficients. $\mu_t = 10m^{-1}$
Volume radius: 11.7m, node radius: 0.02m

The results show that absorption is correctly handled. The results of the bunny cloud are more accurate than those of the Disney cloud, which is partly due to the bunny cloud having a higher attenuation coefficient, making it easier to build a dense graph.

6

Conclusion

In this paper, a novel method for rendering volumes is presented. The method is able to capture the radiance received by the volume and transport it through the volume. This is done by representing the behavior of light in the volume as a graph. The method shows that, depending on volume size, it is able to accurately capture and transport radiance using the graph and produce renders equal to the reference solution.

Our method traces and groups light paths to build the graph, which puts an upper bound on the required memory but also makes the method biased. A detailed analysis is performed on how exactly bias is caused by grouping path segments in Section 4.7.
The results produced by our method vary in accuracy to the reference solution, ranging from equal to very biased. The accuracy depends on the size of the graph nodes and the density of the graph, with larger graph nodes producing positive bias and a sparser graph causing negative bias.

6.1. Practical Applicability

In this thesis, the properties of the scene and volume have been heavily constrained in order to limit the scope to the core problem and make implementation feasible. All of the results in this work are renders of a scene with one volume and one directional light source. Although most of the results look realistic, this kind of scene will never be used in an actual application due to its simplicity. The type of scene closest to our testing environment would be a scene of a cloud in the sky, with several additional infinite lights modeling the atmospheric and ground scattering. Implementing these types of light sources for our graph-based method will give it a real-world application.

The main strength of the method is that it can cheaply render a volume, once the required graph has been built. This cheap rendering from any angle makes the method suitable for producing an animation where the camera moves through a static scene, and our graph-based method will massively outperform traditional renderers if the number of required frames is large.
A major limitation of producing an animation with our method is that only the camera can move through the scene. If any other part of the scene changes, a new graph is required. Although all of the graphs for the whole animation could be precomputed, the then linear time complexity with respect to animation length will be equal to the time complexity of traditional methods, nullifying our method's advantage.

6.2. Future Work

The implementation of the method has been kept simple by only supporting the simplest possible scene and type of participating medium. The exact constraints and indicators on how to solve them are detailed in Section 4.8.
The difficulty of solving these constraints varies, and so does the increase in space requirement. The simplest addition to the method would be support for more and varying light sources, which allows the method to be applied to realistic scenes. Allowing for emissive media increases the cost for large volumes, but is not complicated to implement. Another extension would be to allow for other objects in

the scene. This creates much more variety in possible scenes, but is complicated to implement in an efficient manner as each extra object adds another layer of graph building complexity.
Adding support for an anisotropic phase function is another possibility. This addition will be complex to implement, but typically does not cost anything extra to run.

A more advanced extension to the graph-based method would be to add support for photon beams, where not just photon scattering points are used for rendering, but instead their full travel paths are used. Using graph edges for rendering would allow for greater accuracy and efficiency, as there are many more edges than nodes, but it also requires major fundamental changes to every part of the method. Due to the greater efficiency of photon beams fewer nodes in the graph would be required, thus reducing the memory requirements of the graph. The use of photon beams would add better performance, but no new features.

References

[1] Subrahmanyan Chandrasekhar. *Radiative transfer*. Dover Publications Inc., 1960.

[2] Michael F. Cohen and Donald P. Greenberg. “The hemi-cube: a radiosity solution for complex environments”. In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’85. New York, NY, USA: Association for Computing Machinery, 1985, pp. 31–40. ISBN: 0897911660. DOI: 10.1145/325334.325171. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/325334.325171.

[3] Michael F. Cohen et al. “A progressive refinement approach to fast radiosity image generation”. In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’88. New York, NY, USA: Association for Computing Machinery, 1988, pp. 75–84. ISBN: 0897912756. DOI: 10.1145/54852.378487. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/54852.378487.

[4] Luca Fascione et al. “Path tracing in production: part 1: modern path tracing”. In: *ACM SIGGRAPH 2019 Courses*. SIGGRAPH ’19. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450363075. DOI: 10.1145/3305366.3328079. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3305366.3328079.

[5] Julian Fong et al. “Production volume rendering: SIGGRAPH 2017 course”. In: SIGGRAPH ’17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350143. DOI: 10.1145/3084873.3084907. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3084873.3084907.

[6] Cindy M. Goral et al. “Modeling the interaction of light between diffuse surfaces”. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’84. New York, NY, USA: Association for Computing Machinery, 1984, pp. 213–222. ISBN: 0897911385. DOI: 10.1145/800031.808601. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/800031.808601.

[7] Henri Gouraud. “Computer display of curved surfaces”. AAI7127878. PhD thesis. 1971.

[8] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. “Progressive photon mapping”. In: *ACM Trans. Graph.* 27.5 (Dec. 2008). ISSN: 0730-0301. DOI: 10.1145/1409060.1409083. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/1409060.1409083.

[9] L. G. Henyey and J. L. Greenstein. “Diffuse radiation in the Galaxy.” In: *apj* 93 (Jan. 1941), pp. 70–83. DOI: 10.1086/144246.

[10] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. “The Beam Radiance Estimate for Volumetric Photon Mapping”. In: *Computer Graphics Forum (Proceedings of Eurographics)* 27.2 (Apr. 2008), pp. 557–566. DOI: 10/bjsfsx.

[11] Wojciech Jarosz et al. “A Comprehensive Theory of Volumetric Radiance Estimation Using Photon Points and Beams”. In: *ACM Transactions on Graphics (Presented at SIGGRAPH)* 30.1 (Jan. 2011), 5:1–5:19. DOI: 10/fcdh2f.

[12] Wojciech Jarosz et al. “Progressive Photon Beams”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 30.6 (Dec. 2011). DOI: 10/fn5xzj.

[13] Wojciech Jarosz et al. “Radiance Caching for Participating Media”. In: *ACM Transactions on Graphics (Presented at SIGGRAPH)* 27.1 (Mar. 2008), 7:1–7:11. ISSN: 0730-0301. DOI: 10/cwnw78.

[14] Henrik Wann Jensen and Niels Jørgen Christensen. “Photon maps in bidirectional Monte Carlo ray tracing of complex objects”. In: *Computers & Graphics* 19.2 (1995), pp. 215–224. ISSN: 0097-8493. DOI: https://doi.org/10.1016/0097-8493(94)00145-0. URL: https://www.sciencedirect.com/science/article/pii/0097849394001450.

[15] Henrik Wann Jensen and Per H. Christensen. “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*. SIGGRAPH ’98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 311–320. ISBN: 0897919998. DOI: 10.1145/280814.280925. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/280814.280925.

[16] Jaroslav Křivánek et al. “Unifying Points, Beams, and Paths in Volumetric Light Transport Simulation”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 33.4 (July 2014). DOI: 10/f6cz72.

[17] Eric P. Lafortune and Yves D. Willems. “Bi-directional path tracing”. In: *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics ’93)*. Alvor, Portugal, Dec. 1993, pp. 145–153.

[18] Eric P. Lafortune and Yves D. Willems. “Rendering Participating Media with Bidirectional Path Tracing”. In: *Rendering Techniques ’96*. Ed. by Xavier Pueyo and Peter Schröder. Vienna: Springer Vienna, 1996, pp. 91–100. ISBN: 978-3-7091-7484-5.

[19] Julio Marco et al. “Second-order occlusion-aware volumetric radiance caching”. In: *ACM Transactions on Graphics (Presented at SIGGRAPH)* 37.2 (Apr. 2018). DOI: 10/gdv86k.

[20] S. N. Pattanaik and S. P. Mudur. “Computation of global illumination in a participating medium by monte carlo simulation”. In: *The Journal of Visualization and Computer Animation* 4.3 (1993), pp. 133–152. DOI: https://doi.org/10.1002/vis.4340040303. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/vis.4340040303. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/vis.4340040303.

[21] Mark Pauly, Thomas Kollig, and Alexander Keller. “Metropolis Light Transport for Participating Media”. In: *Rendering Techniques* 2000 (Nov. 2000). DOI: 10.1007/978-3-7091-6303-0_2.

[22] Frederic Pérez, Xavier Pueyo, and François X. Sillion. “Global Illumination Techniques for the Simulation of Participating Media”. In: *Rendering Techniques ’97*. Ed. by Julie Dorsey and Philipp Slusallek. Vienna: Springer Vienna, 1997, pp. 309–320. ISBN: 978-3-7091-6858-5.

[23] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering*. 2023. URL: https://pbr-book.org/4ed/contents.

[24] Holly E. Rushmeier and Kenneth E. Torrance. “The zonal method for calculating light intensities in the presence of a participating medium”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 293–302. ISBN: 0897912276. DOI: 10.1145/37401.37436. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/37401.37436.

[25] Eric Veach. “Robust monte carlo methods for light transport simulation”. AAI9837162. PhD thesis. Stanford, CA, USA, 1998. ISBN: 0591907801.

[26] Eric Veach and Leonidas J. Guibas. “Metropolis light transport”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76. ISBN: 0897918967. DOI: 10.1145/258734.258775. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/258734.258775.

[27] Eric Veach and Leonidas J. Guibas. “Optimally combining sampling techniques for Monte Carlo rendering”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 419–428. ISBN: 0897917014. DOI: 10.1145/218380.218498. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/218380.218498.

A

Bias of Graph Approximation for Other Angles

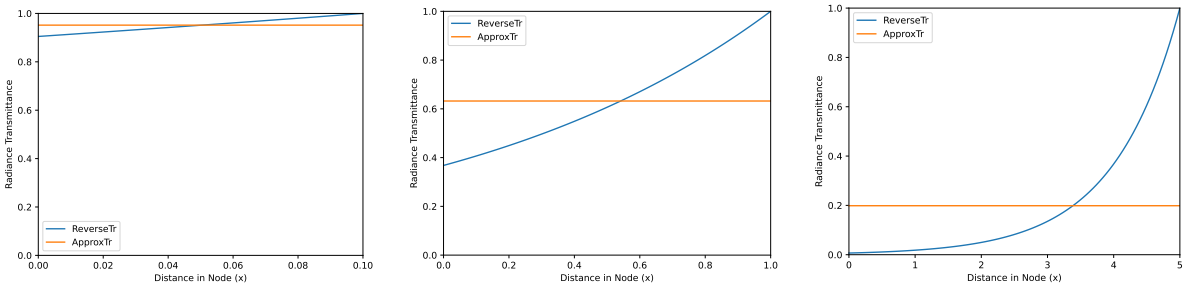
Here the explanation of the bias from averaging node volumes from section 4.7 is continued. The bias for equal light ray and camera ray directions, which is increasingly negative (graph value less than reference) as node size increases, has already been explained. Now the bias for other light and camera directions is explained.

A.1. Opposite Angle

In this example the node entry point and the camera ray direction remain equal to that of section 4.7, and the light ray direction is reversed. The camera and node entry point are still on the left, but the light source is on the right. Compared to the case of equal light and camera direction, the light's direct radiance distribution is now inverted, and increases as the distance x in the node becomes larger:

$$\begin{aligned} \mathrm{ReverseTr}(x, d) &= e^{-(d-x)} \\ \mathrm{ApproxTr}(d) &= \frac{1 - e^{-d}}{d} \end{aligned} \tag{A.1}$$

This inversion can be seen in figure A.1, where the transmittance is the inverse of the transmittance in figure 4.7. The node's approximation remains equal regardless of light direction. Just as for the case of equal light and camera direction, the average of the two equations remains the same for increasing node size, but the average absolute difference increases.



(a) $\mathrm{ReverseTr}$ and $\mathrm{ApproxTr}$ up to 0.1 **(b)** $\mathrm{ReverseTr}$ and $\mathrm{ApproxTr}$ up to 1 **(c)** $\mathrm{ReverseTr}$ and $\mathrm{ApproxTr}$ up to 5

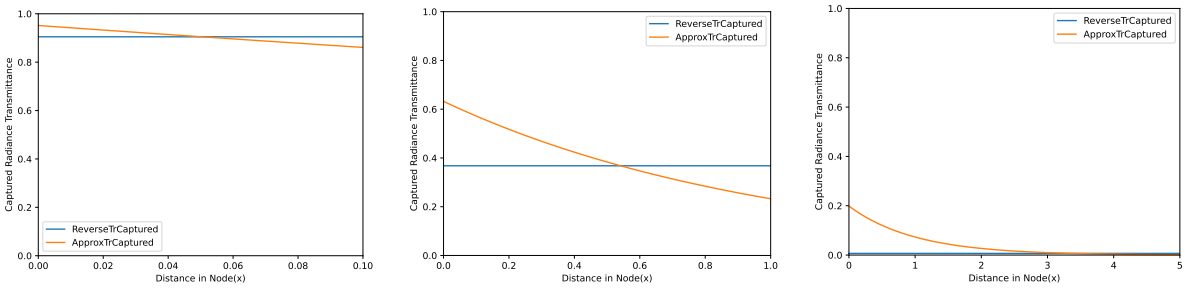
Figure A.1: Reverse transmittance and the node approximation over varying distances. Although the average value of the two function over the range remains equal, the average absolute difference increases.

Multiplying the light's transmittance with the camera transmittance gives a simple equation for the scalar of incoming radiance to the node transported to the camera, which is now a value only depending on

node size and constant through the node.

$$\begin{aligned} \text{ReverseTrTransported}(d) &= e^{-x} \cdot e^{-(d-x)} = e^{-d} \\ \text{ApproxTrTransported}(x, d) &= e^{-x} \cdot \frac{1 - e^{-d}}{d} \end{aligned} \tag{A.2}$$

Plotting the equations in figure A.2 with d the maximum of x for each plot, shows that for the reversed direction case the approximation becomes too large near the node entry point as node size increases.



(a) $\text{ReverseTrTransported}$ and $\text{ApproxTrTransported}$ up to 0.1

(b) $\text{ReverseTrTransported}$ and $\text{ApproxTrTransported}$ up to 1

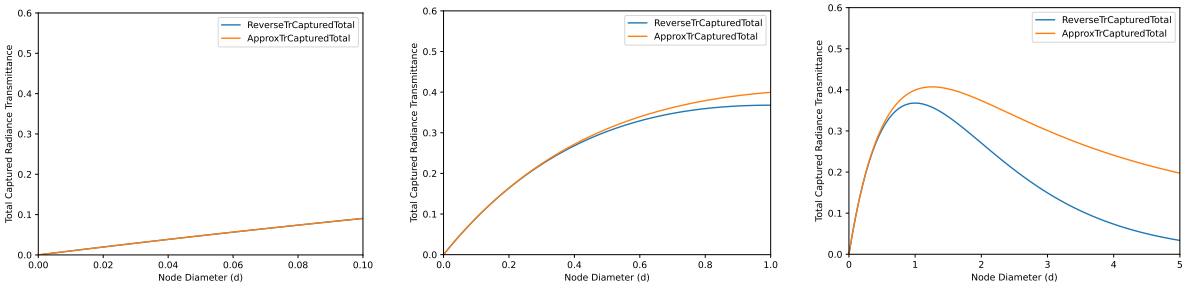
(c) $\text{ReverseTrTransported}$ and $\text{ApproxTrTransported}$ up to 5

Figure A.2: Transported reverse radiance scalar and the node approximation over varying distances. Due to the introduction of radiance transport, the average of the two functions over the range is no longer equal.

Integrating the transported radiance scalar equations from 0 to d gives equations related to just the size of the node.

$$\begin{aligned} \text{ReverseTrTransportedTotal}(d) &= e^{-d}d \\ \text{ApproxTrTransportedTotal}(d) &= \frac{\left(1 - e^{-d}\right)^2}{d} \end{aligned} \tag{A.3}$$

Plotting the total transported scalar of incoming radiance at the node's entry point shows that as the node size increases, the approximation value diverges further from the true value. Instead of too small, the approximation is now too large.



(a) $\text{ReverseTrTransportedTotal}$ and $\text{ApproxTrTransportedTotal}$ up to 0.1

(b) $\text{ReverseTrTransportedTotal}$ and $\text{ApproxTrTransportedTotal}$ up to 1

(c) $\text{ReverseTrTransportedTotal}$ and $\text{ApproxTrTransportedTotal}$ up to 5

Figure A.3: Total transported radiance scalar and the node approximation for varying node sizes. The larger the node becomes the larger the error between the true distribution and the approximation.

A.2. Other Angles

In section 4.7 and this appendix the two extreme cases of $0°$ and $180°$ difference between the light ray and camera ray have been discussed. For $0°$ difference the node approximation has a negative bias, whilst for $180°$ the node has a positive bias. For both cases the rays completely overlap each other, and we performed a comparison between two pairs of rays.
In the actual algorithm many camera rays are used to integrate the whole region of the node, but since all rays have the same direction and the medium is homogeneous, all of them have the same transmittance distribution. Therefore it suffices to only analyze one pair of fully overlapping rays. For other cases a much more complex analysis is required, only for a $90°$ angle the analysis can be simplified.
For the $90°$ angle case the camera rays are perpendicular to the light rays. Each ray has the same transmittance distribution and the final value returned to the camera is the average of all camera rays intersecting the node, so it suffices to analyze the average of all camera rays to a single light ray. This light ray can either have the true radiance distribution, or the node's approximation. Both distributions have the same average value over the full distance in the node, and as all camera rays have the same transmittance to the ray the average of the final distribution is equal. This means that for the case of perpendicular camera and light rays there is no bias.

We now have analyzed three angles, each with a different type of bias. For other angles besides $0°$, $90°$ and $180°$, the bias is harder to compute. What we can do is take the three data points and use them to estimate the bias for any angle.
For $0°$ the bias is negative, and for $90°$ the bias is zero. As the angle increases from $0°$ to $90°$, the negative bias will decrease until it disappears. For $180°$ the bias is positive, so the bias will become positive as the angle becomes greater than $90°$ and increase until the angle reaches $180°$.

B

Algorithm Configuration Details

The graph algorithm requires a configuration file, which specifies the algorithm's parameters. All parameters are explained below, in the same format as required by the program.

The algorithm traces rays from the light source, where it defines a 2D grid perpendicular to the light direction. This grid has two dimensions, each consisting of a certain amount of 'steps'. Each 'step' is a point from which a ray will be traced in the direction of the light.
The algorithm computes a default node radius based on the size of the volume, which performs well for most small and medium sized volumes.
Reinforcement is performed iteratively, with waves of rays being traced until the sparseness threshold has been reached.
The transmittance to each node is determined by computing the transmittance along multiple rays from points on a disk in the node to the light source. This disk intersects the center of the node, is perpendicular to the light direction, and the points are placed on the disk in a grid-like fashion.

- **graphBuilder**
  - *dimensionSteps*: |int| Number of 'steps' in each dimension of the light source grid
  - *iterationsPerStep*: |int| Number of rays to shoot per dimension
  - *maxDepth*: |int| Maximum depth all traced light paths
  - *radiusModifier*: |float| Modifier of the default node radius for determining the final node radius
  - **renderSearchRange**
    - *active*: |bool| Enables render search ranges
    - *neighboursToUse*: |int| Number of nearest neighbors to use for each node's computation
    - *runInParallel*: |bool| Enables multi-threading for computation
  - **edgeReinforcement**
    - *active*: |bool| Enables edge reinforcement
    - *unsatisfiedAllowedRatio*: |float| Allowed ratio of nodes with not enough outgoing edges
    - *reinforcementRays*: |int| Number of rays to shoot from each sparse node in each iteration
    - *edgesForNotSparse*: |int| Number of outgoing edges required for a node to be not sparse
  - **neighbourReinforcement**
    - *active*: |bool| Enables neighbor reinforcement
    - *unsatisfiedAllowedRatio*: |float| Allowed ratio of nodes with not enough neighbors
    - *reinforcementRays*: |int| Number of rays to shoot from each sparse node in each iteration
    - *neighboursForNotSparse*: |int| Number of neighbors required for node to be not sparse
    - *neighbourRangeModifier*: |float| Modifier of the node radius for determining the range in which the node's neighbors must be present
- **lightingCalculator**
  - *lightIterations*: |int| Number of times to trace each ray
  - *pointsOnRadiusLight*: |int| Number of points on the transmittance ray grid's radius
  - *bounces*: |array|int|| Number of light bounces for which a final graph is computed and written to disk, several numbers of bounces can be specified.
  - *runInParallel*: |bool| Enables multi-threading for computation

C

Average Distance through Regions

Equations 4.6 and 4.15, which define the point to node sampling probability, use the average distance through several regions. Here we show how these distances are determined. To keep things simple, the direction is assumed to be equal throughout the relevant region.

The average distance through a sphere is calculated by projecting the sphere on a plane, which is equal to dividing the volume of a sphere by the surface of a circle. This gives:

$$\text{dist\_in\_sphere}(r) = \frac{4}{3}\pi r^3 / \pi r^2 = \frac{4}{3}r \tag{C.1}$$

where
r = the radius of the sphere

To calculate the average distance from point p to the surface of the sphere, we need the distance d, which is the distance from p to the center of the sphere c. Computing d minus the average distance from c to the surface of the sphere, gives the required distance. Since c is in the middle of the sphere, the distance from c to the sphere surface is half of the average distance through the sphere. The formula becomes:

$$\text{dist\_c\_to\_sphere}(r, d) = d - \text{dist\_in\_sphere}(r)/2 = d - \frac{2}{3}r \tag{C.2}$$

where
d = the distance from point p to the center of the sphere c

Both equations are illustrated below:

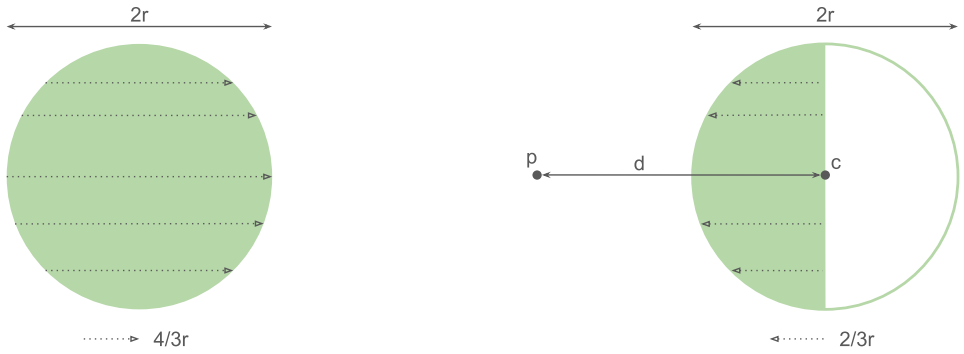


Figure C.1: Illustrations of how the two average distances are calculated. The region over which an average distance is computed is highlighted in green, with dotted lines showing the direction of averaging.