DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

An Analysis of Interactive, Physically Based Rendering of Clouds with Progressive Photon Mapping

Jean Paul Vieira Filho

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

An Analysis of Interactive, Physically Based Rendering of Clouds with Progressive Photon Mapping

Eine Analyse von Interaktiven, Physikalisch Basierten Rendern von Wolken mit Progressive Photon Mapping

Author:	Jean Paul Vieira Filho
Supervisor:	Rüdiger Westermann
Advisor:	Ludwig Leonard Mendez
Submission Date:	15.07.2020

I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.07.2020 Jean Paul Vieira Filho

Acknowledgments

I would like to thank everyone that supported me during the development of this research project.

First and foremost, I would like to thank M. Sc. Ludwig Leonard Mendez for his support and time taken throughout this research, advising in complex topics, providing test data for the implementations, and reviewing this writing. This extends to Prof. Dr. Rüdiger Westermann as well for supervising and providing guidance in this research topic.

I would also like to thank my family and friends for their kind motivation and companionship.

To all of you who assisted me in this accomplishment, my unwavering gratitude.

Abstract

This work focuses on analyzing the viability of interactive, physically based rendering of clouds on the GPU using progressive photon mapping. It introduces basic concepts from radiometry, light transport in participating media, and statistics. Afterward, it presents a collection of state-of-the-art techniques, with basic path tracing theory and an in-depth investigation of photon mapping related techniques. Subsequently, an implementation of path tracing is provided as a reference to compare with the progressive volumetric photon mapping renderer. A third practical approach of progressive photon beams is proposed as an insight on which improvements can be made on top of the analyzed technique. Finally, a comparison between the two implemented renderers is shown in terms of quality and performance. The progressive volumetric photon mapping implementation had a slower time per frame than the path tracing implementation. Furthermore, with equal rendering time, the path tracing renderer had better results. However, the major disadvantages in the implemented photon mapping algorithm can be overcome using progressive photon beams.

Contents

1 Introduction

Natural phenomena are highly complex physical occurrences that can be difficult to fully describe by analytical means. Sky clouds are amongst such phenomena and calculating light interaction with each water droplet in it is virtually impossible with today's technology. To overcome this hurdle, these interactions can be modeled statistically by combining Monte Carlo integration and radiometric concepts. This is a fact that exploited in computer graphics to produce high-quality visual representations of these clouds. These images are used in many fields today, ranging from artistic areas, such as architecture or animated movies, to more technical and scientific works, for instance, meteorology.

Path tracing is one of the most common rendering techniques to create physically based renderings. It simulates light paths backward, coming out of the virtual image plane, bouncing through the scene, and finally arriving at a light source. It has the downside of generally needing a great number of iterations, and therefore time, to achieve good results. Photon Mapping is another technique, introduced in the '90s, that approaches the rendering problem differently by first simulating the light distribution over the scene, and then gathering this distribution towards the camera. Both of these techniques have been expanded extensively over the years. While these extensions give performance improvements, a majority of the publications focus on offline rendering. With modern hardware capabilities and the recent introduction of native ray tracing to rendering APIs, it is worth investigating if these techniques can be used to achieve interactive, real-time rendering.

This thesis' goal is to analyze the suitability of the photon mapping algorithm for interactive progressive rendering in comparison to path tracing. Its secondary goals are achieving a path tracing implementation, a progressive volumetric photon mapping implementation, and provide an insight for a progressive photon beams implementation.

1.1 Outline

This introductory chapter 1 explains light radiation and statistical concepts in detail, which are the foundations of many rendering algorithms that produce high-quality, and realistic images in Computer Graphics. The majority of the content is based on the compilation of other publications provided by [PJH17] and [Fon+17].

Chapter 2 shows a background in the physically based rendering field, some of the state-of-the-art techniques, and explains path tracing and photon mapping in greater detail.

Next, chapter 3 demonstrates practical implementations of path tracing, progressive volumetric photon mapping, and progressive photon beams.

In chapter 4 the results of path tracing and progressive volumetric photon mappings are shown and analyzed.

Lastly, chapter 5 summarizes the research from this thesis and gives an outlook on possible academic works that can be performed based on the presented conclusions.

1.2 Symbol Table

This is a summary of the notation used throughout this thesis. Some variables may not be included here because they occur only once. Additionally, some concepts may be represented by more than one symbol due to different publications using different notations that may facilitate the reader understanding a specific content.

Symbol	Unit	Description
φ	Unitless	Albedo
σ_t	m^{-1}	Extinction coefficient
σ_s	m^{-1}	Scatter coefficient
σ_a	m^{-1}	Absorption coefficient
σ_n	m^{-1}	Free path coefficient
$\bar{\sigma}$	m^{-1}	Majorant coefficient
$T_r(\mathbf{x}, \mathbf{x}')$	Unitless	Transmittance function
$\theta(\mathbf{x}, \mathbf{x}')$	Unitless	Optical thickness function
$\mathbf{x}$	Unitless	A point in 3D space
$\vec{w}$	Unitless	A direction in 3D space
$\Phi(\mathbf{x})$	W	Flux / Power
$L(\mathbf{x}, \vec{w})$	$\mathrm{Wm}^{-2}sr^{-1}$	Radiance
$L_s(\mathbf{x}, \vec{w})$	$\mathrm{Wm}^{-2}sr^{-1}$	In-Scattering Radiance
$L_e(\mathbf{x}, \vec{w})$	$\mathrm{Wm}^{-2}sr^{-1}$	Emitted Radiance
$L_{in}(\mathbf{x}, \vec{w})$	$\mathrm{Wm}^{-2}sr^{-1}$	Incident Radiance
$L_{out}(\mathbf{x}, \vec{w})$	$\mathrm{Wm}^{-2}sr^{-1}$	Exitant Radiance
$f_{ph}(\mathbf{x}, \vec{w}, \vec{w}')$	sr^{-1}	Phase function
$f_s(\mathbf{x}, \vec{w}, \vec{w}')$	sr^{-1}	Bidirectional Scattering Distribution Function
$f_r(\mathbf{x}, \vec{w}, \vec{w}')$	sr^{-1}	Bidirectional Reflectance Distribution Function
$p(x)$	Unitless	Probability Density Function
$P(x)$	Unitless	Cumulative Distribution Function
ϵ	Unitless	Bias / Error

Table 1.1: Symbol Table

1.3 Radiometric Concepts

Radiometry is the science that studies the behavior of optical radiation - the humanly visible portion of the electromagnetic waves greater than infrared radiation and less than ultraviolet. Each physical body can produce and interact with the light constantly, changing wavelengths to produce a variety of colors - e.g. a rock reflecting sunlight or a house lamp emitting warm white light. The functions describing these visible colors are called *spectral power distribution* (SPD). They map the quantity of other radiometric units to wavelengths. These functions are complex to discretize, and a renderer needs to represent these functions to produce images.

In computer graphics, a typical model for SPDs is the *red, green, and blue* (RGB) color model, which is used in this thesis. It represents colors through three scalar components. These can be simply added to calculate the combination of multiple lights at a specific point.

Additional to the spectrum, there are other important concepts needed to render physically based images: radiance, flux, irradiance and radiosity, and intensity. Each of these concepts is based on energy and depends on wavelength. All these concepts are essentially different ways to measure photons carrying energy in the ambient. The energy a photon carries, measured in joules (J) is defined by:

$$Q = \frac{hc}{\lambda}$$

where c is the speed of light (m/s), h is Planck's constant (m^2kg/s) and λ is the photon's wavelength (nm).

This section explains these concepts, how they relate to each other, and why they are important in the rendering process.

1.3.1 Flux

Also named *power*, radiant *flux* represents the total energy traveling through a surface or region of space per unit of time. In rendering, this is useful because usually a scene is assumed to be in a steady-state for each synthesized image. Taking the limit to zero of the differential energy divided by the differential time results in the equation:

$$\Phi = \lim_{\Delta t \to 0} \frac{\Delta Q}{\Delta t} = \frac{\mathrm{d}Q}{\mathrm{d}t}$$

where Φ's unit is watts (W $\equiv$ J/s).

It is stated by [PJH17] that even though photons have discrete packets of energy, due to the extremely large amounts of photons compared to the rendering measurements in practice it is possible to calculate the flux for a light source using the previous formula or the total energy emitted by integrating over a time duration:

$$Q = \int_{t_0}^{t_1} \Phi(t) \mathrm{d}t$$

Light source's power is usually described using flux.

1.3.2 Irradiance and Radiosity

Measurements of flux are always performed over an area. The average density of power in an area can be interpreted as either *irradiance* (E), as arriving at a surface, or *radiosity* (M), as leaving a surface. Both these quantities are defined as

$$E = M = \frac{\Phi}{A}$$

with unit watts per square meter (W/m^2). It can be derived from this formulation that for a given point $\mathbf{x}$, the irradiance (and equivalently the radiosity), the following equation:

$$E(\mathbf{x}) = \lim_{\Delta A \to 0} \frac{\Delta \Phi(\mathbf{x})}{\Delta A} = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A}$$

1.3.3 Intensity

Solid angles are used to measure light intensity. Where *planar angles* measure the portion of the circumference of any 2D circle centered at an arbitrary point as factor of its radius, solid angles measure the portion of the area of any 3D sphere relative to its squared radius. Therefore, where 2D angles have *radians* (r) as their unit, 3D angles have *steradians* (sr). With R being the radius of a circle or sphere, this gives:

$$1\mathrm{r} = R_{2D} \qquad\qquad 1\mathrm{sr} = (R_{3D})^2$$

Similarly to irradiance and radiosity, we can compute the emitted power density emitted through a solid angle. This is defined as *intensity* (I), with unit watts per

solid angle (W/sr). Considering again the limit of a differential cone of directions $\vec{w}$ and a set of directions Ω results in:

$$I = \lim_{\Delta\vec{w}\to 0} \frac{\Delta\Phi}{\Delta\vec{w}} = \frac{\mathrm{d}\Phi}{\mathrm{d}\vec{w}} \qquad\qquad \Phi = \int_{\Omega} I(\vec{w})\mathrm{d}\vec{w}$$

Point lights are the only sources that use intensity. It describes the directional distribution of light [PJH17].

1.3.4 Radiance

The most fundamental radiometric quantity is the *radiance L*. It describes the directional distribution of power and can be seen as an extra step on top of irradiance and radiosity. Considering the previous two quantities per solid angles provides us with the equation:

$$L(\mathbf{x}, \vec{w}) = \lim_{\Delta\vec{w}\to 0} \frac{\Delta E_{\vec{w}}(\mathbf{x})}{\Delta\vec{w}} = \frac{\mathrm{d}E_{\vec{w}}(\mathbf{x})}{\mathrm{d}\vec{w}}$$

with unit ($\mathrm{W/m^2sr}$) and $E_{\vec{w}}$ being the projected area towards $\vec{w}$. In other words, it is the flux density per unit area per unit solid angle:

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}\vec{w}\mathrm{d}A^{\perp}}$$

where $A^{\perp}$ represents the projected area on a surface with a normal w.

Given this value in a problem, all the other radiometric quantities can be derived from it by integrating areas and directions. This property makes it the most important measurement in most rendering situations. Additionally, ray tracing makes use of the fact that radiance does not change when traveling in a vacuum.

[PJH17] introduces further definitions since the light interacting with a surface produces discontinuities when approaching from different sides. This makes sense when simulating surface reflections and refraction. In this thesis, however, only participating media interaction is present and L_{in} will be used to represent incident radiance, e.g. direct or scattered sunlight, and L_{out} for the exitant radiance. Therefore, it can be assumed that $L_{in}(\mathbf{x}, \vec{w}) = L_{out}(\mathbf{x}, -\vec{w})$.

1.4 Monte Carlo & Sampling

Calculating radiance reaching the camera in a scene requires solving complex integral equations. These equations rarely have an analytic solution to them. Monte Carlo provides a way to numerically compute the solution. In comparison to other approaches, Monte Carlo can achieve convergence with a rate that does not depend on the dimensionality of the integrand by employing randomness. By averaging multiple results on the same input from this process, the correct output has a statistically high probability to be achieved.

One major advantage in using Monte Carlo is that with random samples of the integrand function $f(x)$ it is possible to approximate the solution of its integral $\int f(x)\mathrm{d}x$. When simulating light transport, this can be applied to situations like gathering the radiance being scattered at an arbitrary point. It requires integrating over the sphere of all possible directions, but due to the complex nature of light behavior in the scene, this is almost always only possible numerically. With Monte Carlo, all it is needed is a set of directions, calculate incident radiance and scale it by the probability density function, and apply a weighting term.

There are disadvantages however when using Monte Carlo. The convergence of the algorithm is $O(n^{-1/2})$. In ray-traced based rendering, this means that more rays need to be traced. To reduce error by 50%, 4 times the current number of samples at an iteration is needed. Error from Monte Carlo integration demonstrates themselves as noise, where pixels values are off their correct value. In extreme cases, pixels may be too dark or too bright in comparison to their neighbors, which greatly compromises the rendering quality. State-of-the-art techniques focus mostly on reducing the required number of samples while reducing noise at the same time.

This section presents the basics of probability theory, followed by the definition of Monte Carlo estimator, and techniques that can improve the efficiency of such an estimator.

1.4.1 Probability Background

A *random variable* X represents a value selected with a certain probability p. A random variable can be discrete or continuous in its results, where the second one is more predominant in rendering. Adding up all probabilities must always result in 1. A new random variable Y can be acquired by applying a function f to another random variable, that is, $Y = f(X)$.

A *cumulative distribution function* (CDF) represents the probability that a random variable X takes a certain value or less and is defined as:

$$P(x) = Pr\{X \leq x\}$$

From the CDF, the *probability density function* (PDF) can be derived and it is defined as:

$$p(x) = \frac{\mathrm{d}P(x)}{\mathrm{d}x}$$

It describes the relative probability of a random variable taking on a particular value, always integrate to 1 and are non negative [PJH17]. Integrating the PDF over an interval results in the probability of a sample to be in the given range:

$$p(x \in [a,b]) = \int_a^b p(x)\mathrm{d}x$$

The *canonical uniform random variable* ξ is a variable with domain [0,1) and takes all values from it. It has great importance due to the fact that a sample from ξ can be correspondingly transformed to sample other arbitrary distributions - e.g. the discrete probability distribution of a dice roll result. The PDF for ξ is:

$$p_{\xi}(x) = \begin{cases} 1 & x \in [0,1) \\ 0 & otherwise \end{cases}$$

The average value of a function f with PDF $p(x)$ over its domain D is called the *expected value* $\mathrm{E}[f(x)]$, defined by the formula:

$$\mathrm{E}[f(x)] = \int_D f(x)p(x)\mathrm{d}x$$

Given the expected value, one can calculate the *variance* of a function. It is particularly useful to define how different Monte Carlo techniques improve convergence and quality of the result. For a function f, variance is defined as the squared deviation of the function from its expected value:

$$\mathrm{Var}[f(x)] = \mathrm{E}\left[(f(x) - \mathrm{E}[f(x)])^2\right]$$

Assuming independent variables, there are some properties of variance and

expected value that are used throughout this thesis:

$$\begin{aligned} \mathrm{E}\left[af(x)\right] &= a\mathrm{E}\left[f(x)\right] \\ \mathrm{E}\left[\sum_i f(X_i)\right] &= \sum_i \mathrm{E}\left[f(X_i)\right] \\ \mathrm{Var}\left[af(x)\right] &= a^2\mathrm{Var}\left[f(x)\right] \end{aligned}$$

$$\begin{aligned} \mathrm{Var}\left[f(x)\right] &= \mathrm{E}\left[(f(x))^2\right] - \mathrm{E}\left[f(x)\right]^2 \\ \sum_i \mathrm{Var}\left[f(X_i)\right] &= \mathrm{Var}\left[\sum_i f(X_i)\right] \end{aligned}$$

1.4.2 Monte Carlo Estimator

Given a 1D integral $\int_a^b f(x)\mathrm{d}x$, the PDF $p(x)$ with which $X_i \in [a,b]$ is sampled, and the estimator

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)},$$

the Monte Carlo estimator states that the expected value $E[F_N]$ is equal to the given integral. This can be extended to multiple dimensions by using a joint PDF. [PJH17] provides the proof in just a few steps.

This estimator has the advantage that the number of samples N and the error reduction rate of $O(\sqrt{N})$ is independent of the dimensionality of the problem. While for one dimension Monte Carlo might be slower than other techniques, for multiple dimensions it becomes performance-wise better than its counterparts. This makes it well suited for rendering problems.

1.4.3 Sampling

The previously defined estimator requires sampling for the probability density function. This section covers some basic techniques that can be used for that end. [PJH17] explains additionally Metropolis Sampling [Met+53], but since it is not used in this thesis it is not covered.

Inversion Method

By mapping one or more uniform random variables to the original distribution, the inversion method provides a straightforward sampling technique. It can be reduced to four simple steps as shown by [PJH17]:

1. Compute the CDF: $P(x) = \int_0^x p(x')\mathrm{d}x'$
2. Compute the inverse: $P^{-1}(x)$
3. Sample a uniformly distributed random number ξ
4. Sample the original distribution: $X_i = P^{-1}(\xi)$

Rejection Method

Sometimes it is not possible to integrate the function $p(x)$ or their CDFs are not invertible. For these cases the rejection method exits. Assuming a known distribution $p'(X)$ and some constant c exist, so that $p(X) < c \cdot p'(X)$ is valid, then a sample X distributed as $p(X)$ can be obtained as follows:

```
while (true)
{
    X = sampleDistribution();
    if( ξ < p(X)/(c · p'(X)) )
    {
        return X
    }
}
```

where c is some scalar constant and ξ is a uniformly distributed random variable. The efficiency of this sampling is highly dependant on how good of a bound $c \cdot p'(x)$ is for $p(x)$.

1.4.4 Russian Roulette & Splitting

There are techniques to improve the convergence of Monte Carlo estimators. For this, the concept of *efficiency* of an estimator F must be introduced:

$$eff[F] = \frac{1}{V[F]T[F]}$$

with $V[F]$ being the variance and $T[F]$ the time taken to compute its value. If an estimator F_1 takes less time than another estimator F_2 to achieve the same variance, the first is considered more efficient than the second one. This section introduces two techniques, *Russian Roulette* and *Splitting*, that improve the efficiency of Monte Carlo estimators.

Russian Roulette

Russian Roulette is designed to reduce the number of sample calculations that make small contributions to the overall picture. It works by setting a termination probability q and a new estimator F':

$$F' = \begin{cases} \frac{F-qc}{1-q} & \xi > q \\ c & \text{otherwise} \end{cases}$$

where c is an arbitrarily chosen constant (usually $c = 0$). That means that the calculation for these small contributions is entirely skipped and replaced by the chosen constant. This doesn't come with disadvantages: the variance will in the best case, where $c = F$, stay the same. Otherwise, it will always increase. Thus, the coefficient needs to be chosen in a manner that the variance increase is minimal.

Splitting

In comparison to the previous method, splitting increases efficiency by taking more samples - but distributing those samples through different dimensions of the integrand to more effectively reduce the variance. This means that in a situation, with

$$\int_A \int_B f(a,b)\mathrm{d}a\mathrm{d}b$$

and 100 Monte Carlo A samples taken, each with a single B sample, variance might be reduced faster by instead taking 5 A samples each with 20 B samples. This is especially useful when calculating A samples is expensive.

1.4.5 Bias

Bias is defined as the deviation from the correct value. In other words, bias is the difference between the expected value of an estimator and the function to be estimated:

$$\epsilon(F) = \mathrm{E}[F] - \int f(x)\mathrm{d}x$$

Introducing bias can help reduce the variance, however, this depends on the situation where it is employed. An example would be an unbiased estimator for rendering that introduces pixel artifacts due to great variance. If the goal is not necessarily to have a physically correct but just a more artistic representation of a real-world feature, perhaps a biased estimator with lower variance can provide more benefits such as less rendering time.

1.4.6 Importance Sampling

A Monte Carlo estimator has the property that it converges faster to the result if the distribution $p(X_i)$ used is similar to the function of the integrand that is being approximated. This technique is called *Importance Sampling*. When poorly chosen, however, it can lead to an increase in variance.

Importance sampling can be seen as taking values from the integrand function where the most contribution is located instead of uniformly sampling it. A function can be *perfectly importance sampled* when the resulting Monte Carlo weight results in 1.

Multiple Importance Sampling

Multiple Importance Sampling (MIS) solves the issue of multiple functions depending on the same sampling variable. For example, given a problem in the form of:

$$\int f(x)g(x)\mathrm{d}x$$

and assuming that there is no PDF proportional to $f(x)g(x)$, sampling from the same distribution for both of these functions will most likely introduce a large variance for one of them. This will poorly affect the final result due to the two variances being added.

The solution provided by MIS for this problem is to sample each function's distribution individually and introduce weights that reduce the effect of poorly chosen samples for the other functions. In the given example, this changes the Monte Carlo estimator to:

$$\frac{1}{n_f}\sum_{i=1}^{n_f}\frac{f(X_i)g(X_i)w_f(X_i)}{p_f(X_i)}+\frac{1}{n_g}\sum_{j=1}^{n_g}\frac{f(Y_j)g(Y_j)w_g(Y_j)}{p_g(Y_j)}$$

with n_f and n_g being respectively the number of samples taken from either the $f(x)$ distribution $p_f(x)$ or the $g(x)$ distribution $p_g(x)$, and w_f and w_g their associated weighting functions that enforce the convergence of the estimator to $\int f(x)g(x)$.

The weighting functions need to account for all the possible values that could be sampled from X_i and Y_i. [PJH17] recommends using the *balance heuristic* (1.1) and the *power heuristic* (1.2) as proven ways to reduce the variance:

$$w_s(x) = \frac{n_s p_s(x)}{\sum_i n_i p_i(x)} \tag{1.1}$$

$$w_s(x) = \frac{(n_s p_s(x))^\beta}{\sum_i (n_i p(x))^\beta} \tag{1.2}$$

$\beta = 2$ is a good value according to [Vea97].

1.5 Light Transport Components in Participating Media

Participating Media describes any volume that interferes in light's behavior between surface collisions. These can be in a gas form such as smoke, fog and atmospheric clouds, liquids like water and oil, or even solids like wax candles and glass sculptures. Since light interaction within these media happens through a colossal number of atomic particles and molecules, they are best modeled statistically. This can make simulating radiance complex through participating media.

In this section, it is shown that light scattering in participating media is governed by three components: absorption, emission, and scattering, with the last one being affected by phase functions. Additionally, these properties might be constant throughout the media, defined as *homogeneous media*, or varying, described as *heterogeneous media*.

1.5.1 Absorption

Absorption describes the probability density with which light is absorbed for each unit distance inside a medium, or analogously the *absorption cross section*, represented by the coefficient σ_a. In most situations, it varies only with the position, although it can also change based on the direction and spectral dimension. It can take values in the range $[0, \infty)$ and its unit is m^{-1}.

For a certain amount of radiance $L_{in}(\mathbf{x}, -\vec{w})$ coming from direction $-\vec{w}$ at point $\mathbf{x}$, the exitant radiance $L_{out}(\mathbf{x}, \vec{w})$ after the traverse of a differential volume with length $\mathrm{d}t$ is defined as:

$$L_{out}(\mathbf{x}, \vec{w}) - L_{in}(\mathbf{x}, -\vec{w}) = \mathrm{d}L_{out}(\mathbf{x}, \vec{w}) = -\sigma_a(\mathbf{x}, \vec{w}) L_{in}(\mathbf{x}, -\vec{w}) \mathrm{d}t$$

Solving this equation delivers the fraction of light that is absorbed by the media. The integral that describes this process is defined by

$$\exp\left(-\int_o^d \sigma_a(\mathbf{x} + t \cdot \vec{w}, \vec{w}) \mathrm{d}t\right)$$

where d is the distance traveled along the ray direction $\vec{w}$ from original point $\mathbf{x}$.

Figure 1.1: Absorption

1.5.2 Emission

This process is the theoretical opposite of absorption. For each unit of distance traveled inside the medium, the radiance is increased due to the conversion of energy from other sources, such as heat or nuclear decay. Given a differential volume distance, the differential equation of emission is defined as:

$$\mathrm{d}L_{out}(\mathbf{x}, \vec{w}) = L_e(\mathbf{x}, \vec{w})\mathrm{d}t$$

where L_e is the emitted light. This equation is based on linear optics assumptions where the emitted light does not depend on the incoming light L_{in}.

Figure 1.2: Emission

1.5.3 Out-Scattering, Extinction and Beam Transmittance

Out-scattering is the light interaction with the particles in a differential section of the media. Each time a light ray collides with a particle, it scatters in a different direction and therefore reduces the exitant radiance. The probability of this event happening is given by the scattering coefficient σ_s. Just like the absorption coefficient σ_a, it has unit m^{-1}. The out-scattering equation is given by:

$$\mathrm{d}L_{out}(\mathbf{x}, \vec{w}) = -\sigma_s(\mathbf{x}, \vec{w})L_{in}(\mathbf{x}, -\vec{w})\mathrm{d}t$$

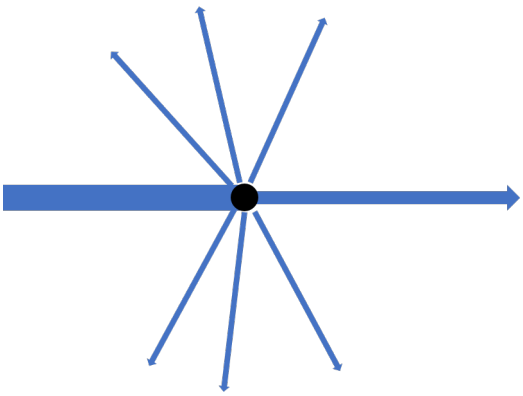


Figure 1.3: Out-Scattering

Adding σ_a and σ_s gives us the *attenuation* coefficient, also known as *extinction* coefficient:

$$\sigma_t(\mathbf{x}, \vec{w}) = \sigma_a(\mathbf{x}, \vec{w}) + \sigma_s(\mathbf{x}, \vec{w})$$

Two values of great importance are derived from the extinction coefficient: *albedo* (1.3), which represents the probability of scattering against absorption, and *mean free path* (1.4), representing the average distance travelled by a ray before colliding with a particle in the medium.

$$\varphi = \frac{\sigma_s}{\sigma_t} \tag{1.3}$$

$$\Delta\tilde{t} = \frac{1}{\sigma_t} \tag{1.4}$$

With the extinction coefficient, the radiance attenuation becomes:

$$\frac{\mathrm{d}L_{out}(\mathbf{x}, \vec{w})}{\mathrm{d}t} = -\sigma_t(\mathbf{x}, \vec{w})L_{in}(\mathbf{x}, -\vec{w})$$

and when solved, it gives the *beam transmittance* equation:

$$T_r(\mathbf{x} \to \mathbf{x}') = \exp\left(-\int_o^d \sigma_t(\mathbf{x} + t \cdot \vec{w}, \vec{w})\mathrm{d}t\right)$$

with $d = \|p - p'\|$ being the distance between two points, w the normalized direction between those points and T_r the beam transmittance between them. T_r is always a value between 0 and 1, which can then be multiplied by an arbitrary exitant radiance travelling through the medium:

$$T_r(\mathbf{x} \to \mathbf{x}')L_{out}(\mathbf{x}, \vec{w})$$

There are some properties of the beam transmittance that are of importance:

- The transmittance from a point to itself is always one $T_r(\mathbf{x} \to \mathbf{x}) = 1$
- If the extinction coefficient $\sigma_t = 0$, then the transmittance is $T_r(\mathbf{x} \to \mathbf{x}') = 1$ for every point $\mathbf{x}'$ in the medium.
- If the extinction coefficient is symmetric $\sigma_t(\mathbf{x}, \vec{w}) = \sigma_t(\mathbf{x}, -\vec{w})$ or does not depend on the direction at all, then it follows that $T_r(\mathbf{x} \to \mathbf{x}') = T_r(\mathbf{x}' \to \mathbf{x})$
- In every media, the transmittance is multiplicative along points of the same ray: $T_r(\mathbf{x} \to \mathbf{x}'') = T_r(\mathbf{x} \to \mathbf{x}')T_r(\mathbf{x}' \to \mathbf{x}'')$

- In homogeneous media, σ_t is constant throughout the medium, thus trivially solving the transmittance and giving the Beer's law: $T_r(\mathbf{x} \rightarrow \mathbf{x}') = \exp(-\sigma_t d)$

The negated exponent in the T_r equation is called the *optical thickness*

1.5.4 In-Scattering

Just like Out-Scattering can change light rays' course off a direction, In-Scattering models the rays that are refracted towards the evaluated direction w. This process is calculated with the help of a *phase function* denoted by $f_{ph}(\mathbf{x}, \vec{w}, \vec{w}')$, which represents the probability of a ray being scattered in the direction w' and is explained with more detail in the next section. The In-Scattering of radiance in a medium is thus defined as:

$$L_s(\mathbf{x}, \vec{w}_o) = \int_{S^2} f_{ph}(\mathbf{x}, \vec{w}_i, \vec{w}_o) L_{in}(\mathbf{x}, \vec{w}_i) \mathrm{d}\vec{w}$$

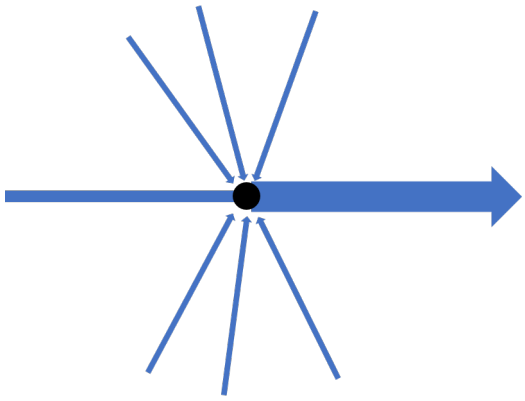


Figure 1.4: In-Scattering

1.5.5 Phase Functions

Phase functions represent the scattering properties of different media and can be parameterized or analytic models, corresponding to the shape and type of particles present in a volume. Furthermore, they are constrained to have a normalized integral over all directions to represent the probability distribution of a ray scattering in a particular direction w' from another direction w:

$$\int_{S^2} f_{ph}(\mathbf{x}, \vec{w}, \vec{w}') \mathrm{d}\vec{w}' = 1$$

Usually, media have an *isotropic* phase function, which means that all scatter directions have equal probability. Thus, it is valid to assume that $f(w, w') = f(w', w)$.

The opposite type of phase function is referred to as *anisotropic*, and it varies according to the angle given as input.

The only possible definition for an isotropic phase function, due to the normalization, is:

$$f_{ph}(\mathbf{x}, \vec{w}_i, \vec{w}_o) = \frac{1}{4\pi}$$

However, there are other functions that can simulate both isotropic and anisotropic behaviour with the aid of parameters. A popular function used is the Henyey-Greenstein phase function, which uses a single *asymmetry* parameter g to control the distribution of scattered light:

$$f_{ph_{(HG)}} = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 + 2g(\cos\theta))^{3/2}} \text{ where } g \in (-1, 1)$$

where θ is the angle between the two given directions. The closer the value of g to -1, the more light is scattered back in the incident direction. Conversely, the closer it is to 1, the higher the probability of light scattering along the incident direction. If the value is set to 0, the f_{HG} becomes the isotropic function. Figure 1.5 illustrates this behavior.

The Henyey-Greenstein parameter g represents the approximation of another arbitrary phase function multiplied by the cosine of the angle between the two directions. It can be thus calculated as:

$$g = \int_{S^2} f_{ph}(\mathbf{x}, -\vec{w}, \vec{w}')(\vec{w} \cdot \vec{w}')\mathrm{d}\vec{w}' = 2\pi \int_0^{\pi} f_{ph}(\mathbf{x}, -\cos\theta) \cos\theta \sin\theta \mathrm{d}\theta$$

The simplicity in the evaluation of $f_{ph_{(HG)}}$ compensates for its loss of accuracy. Furthermore, some more complex phase functions can be approximated by a weighted sum (to maintain normalization) of Henyey-Greenstein functions with different asymmetry values:

$$f_{ph}(\mathbf{x}, \vec{w}, \vec{w}') = \sum_{i=1}^{n} \vec{w}_i f_{ph_{(HG)_i}}(\mathbf{x}, \vec{w}, \vec{w}')$$

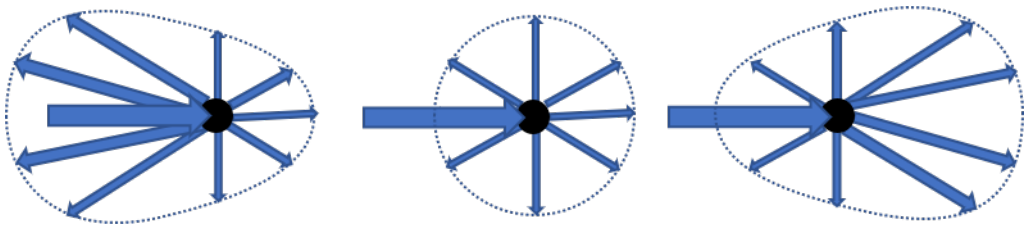


Figure 1.5: Henyey-Greenstein Phase Function

1.6 Light Transport Models

1.6.1 Light Transport Equation

Since it is important for the derivation used in many rendering techniques, the *light transport equation* (LTE) is briefly explained in this section. The LTE is based on the *energy balance* principle, where the difference between energy leaving Q_o and entering Q_i a system must be equal to the difference between the energy being emitted Q_e and absorbed Q_a:

$$Q_o - Q_i = Q_e - Q_a$$

Without considering any participating media, this gives us the radiance exitance for any point $\mathbf{x}$ in the direction $\vec{w}_o$ in a scene:

$$L_{out}(\mathbf{x}, \vec{w}_o) = L_e(\mathbf{x}, \vec{w}_o) + \int_{S^2} f_s(\mathbf{x}, \vec{w}_o, \vec{w}_i) L_{in}(t(\mathbf{x}, \vec{w}_i), -\vec{w}_i) |\cos\theta_i| \mathrm{d}\vec{w}_i$$

with $t(\mathbf{x}, \vec{w}_i)$ being the *ray-casting* equation that finds the point on the first surface intersecting a ray from $\mathbf{x}$ in direction $\vec{w}_i$, $f_s(\mathbf{x}, \vec{w}_o, \vec{w}_i)$ the *bidirectional scattering distribution function* (BSDF) modelling the probability of a ray scattering direction at the point $\mathbf{x}$ of an arbitrary surface, and θ_i the angle between the incident direction $\vec{w}_i$ and the exitant direction $\vec{w}_o$.

1.6.2 Equation of Radiative Transfer

"The equation of transfer is the fundamental equation that governs the behavior of light in a medium that absorbs, emits, and scatters radiation" [PJH17]. With all the previously defined concepts, the *equation of radiative transfer*, also referred as the *equation of transfer* or *radiative transfer equation*, can be assembled. It describes the change of radiance along a beam direction at a certain point in space through

participating media:

$$(\vec{w} \cdot \nabla) L_{in}(\mathbf{x}, \vec{w}) = -\sigma_t(\mathbf{x}) L_{in}(\mathbf{x}, \vec{w}) + \sigma_a(\mathbf{x}) L_e(\mathbf{x}, \vec{w}) + \sigma_s(\mathbf{x}) \int_{S^2} f_{ph}(\mathbf{x}, \vec{w}, \vec{w}') L_{in}(\mathbf{x}, \vec{w}') \mathrm{d}\vec{w}'$$

with the first term in the addition being the attenuation of light due to out-scattering and absorption, the second one the medium self-emission, and the third one the in-scattering from all directions.

1.6.3 Volume Rendering Equation

While the previous equation can be used in many finite element techniques, it is not suitable for the techniques used in this work, i.e. Path Tracing and Photon Mapping. According to [Fon+17], if we integrate both sides of the equation we arrive at the *volume rendering equation* (VRE):

$$L_{in}(\mathbf{x}, \vec{w}) = \int_{t=0}^{d} T_r(t) \Big[\sigma_a(\mathbf{x}_t) L_e(\mathbf{x}_t, \vec{w}) + \sigma_s(\mathbf{x}_t) L_s(\mathbf{x}_t, \vec{w}) + L_{in,d}(\mathbf{x}_d, \vec{w}) \Big] \mathrm{d}t$$

where d is the total length of the ray inside of the volume, s the distance from the ray's volume exit point a point along the ray, $L_s = \int_{S^2} f_{ph}(\mathbf{x}, \vec{w}, \vec{w}') L(\mathbf{x}, \vec{w}') \mathrm{d}\vec{w}'$ from the previous section, $T_r(t) = \exp\left(- \int_{s=0}^{t} \sigma_t(\mathbf{x}_s) \mathrm{d}s \right)$ the transmittance until the point $\mathbf{x}_t$ along a light ray, and $L_{in,d}(\mathbf{x}_d, \vec{w})$ being the radiance coming from outside of the volume at the end of the ray.

Where the equation of transfer describes where the radiance is going to (forward transport), the volume rendering equation describes where radiance is coming from at a certain point (backward transport). This makes it useful because it is possible to accumulate radiance coming to a camera plane position from the scene.

1.7 Scattering Sampling

There is still an issue to be solved: the previous models still need to simulate scattering events in participating media. There are two major categories for stochastic Monte Carlo that solve this problem, *tracking* and *PDF* approaches [Fon+17].

1.7.1 Tracking Approaches

This category relies on rejection sampling and Russian roulette (sections 1.4.3 and 1.4.4). Light beams are thus modeled as multiple photons bouncing around the scene and only one type of collision is taken into account at a time, keeping the photons' full radiance throughout the entire process.

The previous problem now becomes finding the free-path distance that a photon can traverse before any type of collision happens.

Closed-form Tracking

First used by Stan Ulam [Eck87], this approach assumes that the participating media volumes have simplistic characteristics: the extinction coefficient change is either constant, polynomial, or exponential. For the first and most important case, the volume would be a homogeneous medium.

In a homogeneous volume, where σ_t does not change in the space (section 1.5.3), the transmittance is

$$T_r(t) = e^{-\sigma_t t}$$

By normalizing the homogeneous transmittance $T_r(t)$, an estimator is created that can be used to sample free-flight distances. The PDF for this estimator is defined as:

$$p(t) = \sigma_t e^{-\sigma_t t}$$

which can then be perfectly importance sampled by applying the inversion method [PJH17] with

$$t' = -\frac{\ln(1-\xi)}{\sigma_t}$$

Applying this to the VRE without the part where the ray is outside of the

volume, simplifies it stochastically to:

$$L_{in}(\mathbf{x}, \vec{w}) = \int_0^d p(t) \Big[P_a(\mathbf{x}_t) L_e(\mathbf{x}_t, \vec{w}) + P_s(\mathbf{x}_t) L_s(\mathbf{x}_t, \vec{w}) \Big] \mathrm{d}t$$

where $P_a = \sigma_a / \sigma_t$ and $P_s = \sigma_s / \sigma_t$ (single scattering albedo). Since both P_a and P_s already represent the probabilities of either a scatter or an absorption happening, these can be used with Russian roulette to decide what effect is simulated.

Let $\mathbf{x}_0$ be the starting position for the simulation and $\mathbf{x}_{j+1} = \mathbf{x}_j - t_j \vec{w}_j$ be the next scattering event with distance t_j in the direction $\vec{w}_j$. For an arbitrary scattering event, if the distance range inside the volume is $(0, d)$ and the distance range after the volume boundary is (d, ∞), then the integral turns into:

$$\begin{aligned} L_{in}(\mathbf{x}_j, \vec{w}) = &\int_{t=0}^{d} p(t_j) \Big[P_a(\mathbf{x}_{j+1}) L_e(\mathbf{x}_{j+1}, \vec{w}_j) + P_s(\mathbf{x}_{j+1}) L_s(\mathbf{x}_{j+1}, \vec{w}_j) \Big] \mathrm{d}t \\ &+ L_{in,d}(\mathbf{x}_{d,j}, \vec{w}_j) \int_{t=d}^{\infty} p(t_j) \mathrm{d}t \end{aligned}$$

The second integrand represents no collision happening before the ray leaves the volume boundary, which essentially means the boundary radiance is returned. Skin subsurface scattering and atmosphere are some examples of simulations using this approach.

Regular Tracking

Regular tracking is applied where the volume is defined in piecewise constant sections. Closed-form tracking can be used inside of each section and then migrating into the next domain if no scatter happens inside of the current one. The radiance $L_{in,d}$ from the previous tracking method can then be interpreted as radiance coming from another part of the participating medium. This approach can become inefficient is the volume has too many sections.

Delta Tracking

This approach is for this thesis the most important one and is based on rejection sampling. Besides *delta tracking*, it is also called in the literature as *Woodcock tracking*, the *null-collision* algorithm, or *pseudo scattering*.

Since heterogeneous volumes cannot profit from the constant transmittance assumption, delta tracking provides a solution for this problem by introducing a

new *null-collision* coefficient $\sigma_n(\mathbf{x})$ and the *free-path* coefficient $\bar{\sigma}$. The null-collision σ_n represents an event where no collision happens and the simulation continues with the same direction and radiance. These coefficients homogenize the entire volume through the following definition:

$$\bar{\sigma} = \sigma_a(\mathbf{x}) + \sigma_s(\mathbf{x}) + \sigma_n(\mathbf{x}) = \sigma_t(\mathbf{x}) + \sigma_n(\mathbf{x})$$

The only condition is that the free-path coefficient $\bar{\sigma}$ needs to be *majorant* of $\sigma_t(p)$:

$$\bar{\sigma} \geq \sigma_t(\mathbf{x})$$

The null-collision coefficient $\sigma_n(\mathbf{x})$ is thus defined for an arbitrary point $\mathbf{x}$ inside the volume as:

$$\sigma_n(\mathbf{x}) = \bar{\sigma} - \sigma_t(\mathbf{x})$$

With the volume now being homogeneous, closed-form tracking can be used for heterogeneous volumes with the adjusted probabilities:

$$P_a(\mathbf{x}) = \frac{\sigma_a(\mathbf{x})}{\bar{\sigma}} \qquad P(\mathbf{x}) = \frac{\sigma_s(\mathbf{x})}{\bar{\sigma}} \qquad P_n(\mathbf{x}) = \frac{\sigma_n(\mathbf{x})}{\bar{\sigma}}$$

$$P_a(p) + P_s(p) + P_n(p) = 1$$

and the new VRE in recursive form:

$$\begin{aligned} L(\mathbf{x}_j, \vec{w}) = \int_{t=0}^{d} p_n(t_j) \Big[& P_a(\mathbf{x}_{j+1}) L_e(\mathbf{x}_{j+1}, \vec{w}_j) \\ & + P_s(\mathbf{x}_{j+1}) L_s(\mathbf{x}_{j+1}, \vec{w}_j) + P_n(\mathbf{x}_{j+1}) L(\mathbf{x}_{j+1}, \vec{w}_j) \Big] \mathrm{d}t \\ & + L_{in,d}(\mathbf{x}_{d,j}, \vec{w}_j) \int_{t=d}^{\infty} p(t_j) \mathrm{d}t \end{aligned}$$

The efficiency of delta tracking depends on how tight of a boundary $\bar{\sigma}$ is to $\sigma_t(\mathbf{x})$, and volumes can be subdivided into smaller portions to improve this. Both the path tracing and the photon mapping shaders contain code examples of these algorithms.

1.7.2 PDF Approaches

In comparison to tracking approaches, PDF approaches split the photon rays into fractions and simulate all of them. This permits the VRE integral to be separated:

$$L(p, w) = \int_{t=0}^{d} T_r(t) \sigma_a(\mathbf{x}_t) L_e(\mathbf{x}_t, \vec{w}) \mathrm{d}t + \int_{t=0}^{d} T_r(t) \sigma_s(\mathbf{x}_t) L_s(\mathbf{x}_t, \vec{w}) \mathrm{d}t + T_r(d) L_{in,d}(\mathbf{x}_d, \vec{w})$$

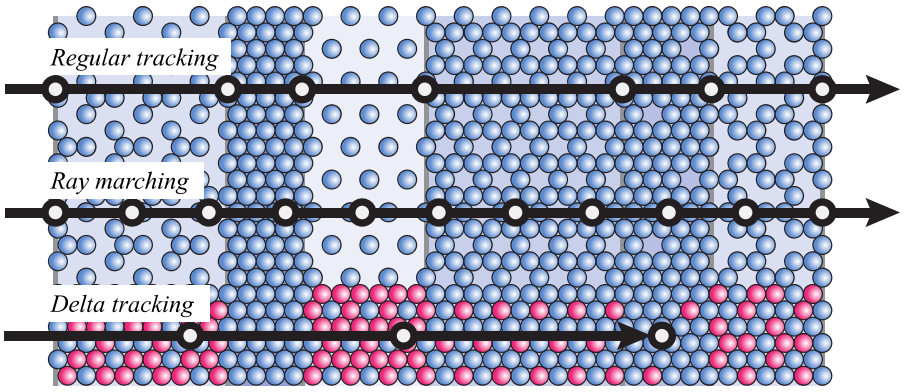


Figure 1.6: Tracking Comparison. Source [NSJ14]

To estimate radiance from emission and in-scattering, PDFs are built, and points are importance sampled from this distribution to find regions of most contribution. In these regions, radiance is then evaluated and added to the final result. Each sample needs to be attenuated accordingly along its originating ray. Therefore, the transmittance term T_r needs to be evaluated for each sampled position. Through caching, this process can be optimized but not in all situations.

Finding good transmittance estimators and accurate PDFs becomes the main challenge in this type of approach. In comparison to tracking approaches, this process takes more time but delivers higher quality estimates.

Ray Marching

This technique is relatively simple but introduces bias which can cause visible artifacts in the final image. For each ray traced through the volume, many samples are taken in steps along its segment, thus giving it the *marching* part of the name [PH89]. The major advantage of this method is that it does not require any additional data, and if the step sizes are very small, the estimate becomes very accurate (albeit slow) and can be used as a reference for other techniques' efficiency.

PDF Delta Tracking

This approach works by employing the previously shown non-PDF delta tracking to estimate if a collision happens before, resulting in $T_r(t) = 1$, or after a given

point, with $T_r(t) = 0$, at position t along a ray direction. While it is too inaccurate to be used on its own, it serves as a base for the next estimator.

Ratio Tracking

Introduced by [NSJ14], this unbiased estimator works by replacing the Russian roulette component of delta tracking by a Monte Carlo weight T. The weight represents an approximation of the probability of an extinction collision happening w.r.t. to the majorant at a given collision point:

$$T_{ratio}(d) = \prod_{i=1}^{K} \left(1 - \frac{\sigma_t(x_i)}{\bar{\sigma}}\right)$$

where K represents the number of sampled points that happened before reaching the distance d. These positions are determined using K iterations of delta tracking, using the previous sample point x_{i-1} for $i \in [0, K]$ as the start position. Figure 1.7 shows an example where $K = 3$ samples are taken before passing the distance d.

Residual Ratio Tracking

Also presented in [NSJ14], this technique combines Ratio Tracking with an additional *control extinction coefficient* $\bar{\sigma}_r$ that is constant throughout the volume. This changes the previous transmittance weight to a two component one defined as:

$$\begin{aligned} T_{control}(d) &= \exp\left(-\sigma_c t\right) \\ T_{ratio}(d) &= \exp\left(-\int_{s=0}^{t} \sigma_t(x_s) - \sigma_c \, \mathrm{d}s\right) \approx \prod_{i=1}^{K} \left(1 - \frac{\sigma_t(x_i) - \sigma_c}{\bar{\sigma}_r}\right) \\ T_{residual}(d) &= T_{control}(d) T_{ratio}(d) \end{aligned}$$

A new majorant over the residual extinction coefficient is introduced as $\bar{\sigma}_r = \sigma_t - \sigma_c$, and $\bar{\sigma}_r < \bar{\sigma}$ must be valid. If these conditions are met, less samples are needed from the extinction function $\sigma_t(x_s)$ due to the larger steps caused by $\bar{\sigma}_r$.

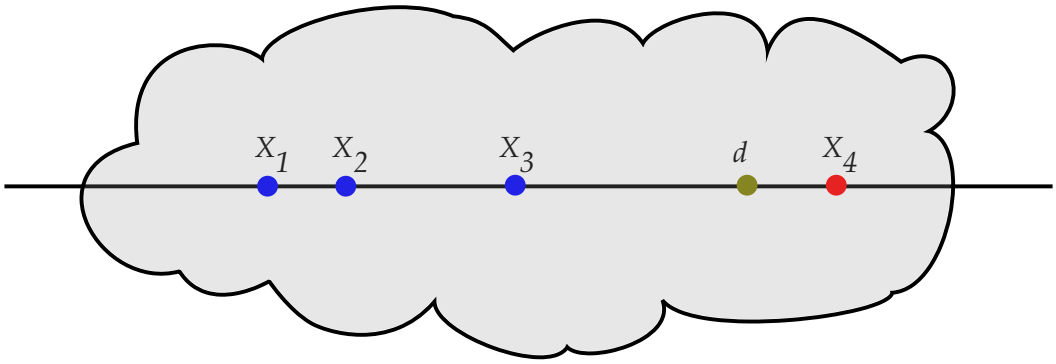


Figure 1.7: Ratio Tracking

2 Methodology

Many approaches have been developed over the years to represent realistic light behavior in scenes. Each of them has specific advantages that make them suitable for simulating certain physical effects, such as caustics, reflections, or global illumination. Some of these methods focus as well on the trade-off between accuracy and performance to achieve interactive rendering times.

This chapter presents some of the state-of-the-art techniques and focuses on two particular approaches and their derivations with emphasis on their volumetric rendering solutions: Path Tracing and Photon Mapping.

2.1 Overview

Light transport in participating media has evolved over the past decades in a variety of directions. While rasterization has great performance in simulating local illumination, the disjoint manner in which each pixel color in the image plane is calculated hinders its potential for global illumination. ray-tracing based methods, while more computationally expensive, can comfortably handle global illumination since it explicitly simulates light through a scene. The majority of the rendering APIs and graphics cards vendors currently offer ray-tracing support.

Hardware and Software Support

At the time of this writing, Vulkan recently released its ray-tracing extension [Koc20], and Microsoft offers its own DXR extension for DirectX 12 [Mic20a]. On the hardware side, NVidia has its RTX architecture [NVi20] specialized in accelerating ray-tracing and its related data structures calculations, and AMD has announced its equivalent in the upcoming RDNA2 GPU series [Adv20]. However, all these technologies require higher-end peripherals. Therefore, this thesis uses standard compute shaders in its implementation due to the compatibility with a wider array of graphics cards and platforms and focuses on the theoretical implementation of current techniques for volumetric global illumination.

Radiosity and Virtual Point Lights

First introduced in computer graphics by Goral et al. [Gor+84], *Radiosity* rendering is based on heat transfer from thermal engineering. It simulates flux propagation from light sources, which then is diffusely reflected through the scene and finally reaches the eye. This is achieved by discretizing the scene into tiles, and analyzing for each tile pair how much flux leaves a surface towards the other - hence the name Radiosity.

While the results for the traditional algorithm excel for diffuse reflections, it cannot handle volume rendering out of the box. Keller improved the technique with *Instant Radiosity*, also commonly referred to as *virtual point lights* [Kel97]. Instead of discretizing the scene and simulating every single interaction, light particles are traced into the scene using quasi-random walk [Kel98]. From there, the scene is rendered once per light particle acting as a point light, and the results are accumulated to form the final image. Volumetric scattering of these particles

is easily handled by the algorithm, but additional work is needed to account for light attenuation in participating media.

Novak et al. [Nov+12b] changed the representation of the virtual lights in participating media from points to rays, where the light rays have their origins and directions based on the scatter events. These virtual ray lights are then sampled multiple times to calculate the radiance at an arbitrary point and direction, resulting in a more densely sampled and less prone to singularities (extremely bright points) final accumulated result.

Progressive Virtual Beam Lights [Nov+12a] managed to eliminate artifacts and improve performance in comparison to Virtual Ray Lights by inflating the rays into beams, and surface points to spheres, both with finite thickness and radius. Additionally, at each frame, the radius of the beams is reduced progressively to achieve convergence to the reference images.

Radiometric Caching

Similar to Radiosity based methods, *Irradiance Caching* presents a solution for efficiently solving the diffuse indirect illumination component in global illumination. As mentioned by Ward et al. in the original publication [WRC88], this indirect illumination changes relatively slow across a surface. Therefore, instead of calculating a high-quality estimate for every pixel in the image plane, only a sparse representation is simulated and cached for interpolation wherever possible. The interpolation scheme applied is a weighted average using "Split-Spheres", which represents an upper bound for the irradiance gradient at the high-quality sample points. In a later publication, Ward and Heckbert [WH92] realized that the gradients could be calculated explicitly instead of storing split-spheres, thus enabling higher-order interpolation schemes of the samples.

Jarosz et al. extended the original approach in *Radiance Caching for Participating Media* [Jar+08] to enable volumetric effects. This is achieved by calculating the gradient of the transmittance between two points in space using two separate components: single and multiple scattering. The first component calculates the gradient relative to the position in the media, using samples at light-emitting and reflecting surfaces. The second distributes samples along multiple random walk-paths in the media and calculates the gradient by translating both the sample positions and their respective paths.

Recently, Marco et al. [Mar+18] improved the quality and rendering times of the final images by introducing occlusion awareness to the interpolation and using

both first and second-order derivatives in a Hessian-based interpolation scheme.

Monte Carlo Path Integration

Path tracing might be one of the more popular rendering techniques under the ray-tracing umbrella. First presented by Kajiya [Kaj86], it works by tracing paths from the camera, reflecting it onto the scene, and finally sampling light sources. For each pixel, these sampled path colors are then averaged to achieve convergence to the true value.

One of the most notable contributions to the original algorithm is the bidirectional extension from Lafortune and Willems [LW93]. A major flaw of the traditional approach is that not every path ends up sampling a light source due to occlusion. This extension improves on that aspect by tracing paths from the light in combination to those from the camera. It not only helps in better finding paths that carry light but also greatly reduces the variance through multiple importance sampling.

In [LW96], Lafortune and Willems extended Bidirectional Path Tracing to handle non-homogeneous participating media using random walk and directly sampling the light sources at each scatter point.

Veach and Guibas presented the Metropolis Light Transport [VG97], which is another variation of path tracing. Instead of created independent paths, this approach creates new paths based on previously emitted ones using a Markov chain. Pauly et al. then proposed a volumetric solution to this method in [PKK00].

Photon Mapping

Photon Mapping is a technique created by Jensen in [Jen96] which splits the rendering process in two passes: photon tracing and photon density estimation. The first part distributes each light sources' flux through the scene via particle tracing, where each collision with the scene deposits a photon with position, energy, and incident direction. At the second pass, these photons are gathered at their respective location with rays traced from the camera. This approach is particularly suitable for rendering caustics. Volumetric Photon Mapping [JC98] formulates a solution for participating media using the same two-pass approach, with an additional photon map that uses volume scattering process instead of surface collisions.

Jarosz, Zwicker, and Jensen presented in [JZJ08] a more efficient approach to gathering photons by expanding the photon points into spheres and ray-tracing these determine which ones affect each camera ray. This is the dual of gathering infinitesimal points with a beam, hence the name.

A major problem with standard photon mapping is that it requires a large number of photons for good results. Ideally, if the number of photons was infinite, the image would be unbiased. However, this would require infinite memory. Progressive Photon Mapping [HOJ08], by Hachisuka, Ogaki, and Jensen, solves this issue by offering a progressive photon mapping technique, where many frames are rendered and averaged progressively. It achieves this by creating gathering points and keeping statistics about previously sampled photons to reduce the gathering radius. Later, the possibility of distributed ray-tracing effects was added in [HJ09]. In [HJ10] a version of progressive photon mapping for GPUs was introduced by using a uniform hash grid in the scene and Russian roulette to handle collisions. Knaus and Zwicker eliminated the need for local statistics in [KZ11] by employing a probabilistic approach, where the radius reduction formulation reduces the average variance towards zero by slightly increasing the variance of each frame. Another advantage of this improvement is that each frame can be rendered independently of each other. This allowed photon mapping to run in parallel on the GPU and even in GPU clusters as shown by [Ped13].

Jarosz et al. extended the photon formulation in [Jar+11a] by introducing new types of photon data representations and query geometries based on points and beams, with different blurring kernels for the gathering. The photon beam data structure with the beam estimate blur in 1D increased performance significantly. Combining this publication with the one in [KZ11], Jarosz et al. presented a progressive approach that uses photon beams and beam gathering in [Jar+11b].

Bitterli and Jarosz recently presented in [BJ17] that the photon data representation and query geometry can be extended to higher dimensions and fully unbiased forms for volumetric photon mapping can be achieved. These higher dimensional samples are derived by taking the limit to infinity of the number of samples in the previous lower dimension, or equivalently, reducing the size of all the samples in a dimension to zero while keeping the covered range in that same dimension constant. These formulations are however quite complex since, for each added dimension, an additional distance sample needs to be calculated as well.

Deng et al. showed in [Den+19] that photon planes can be generalized to surfaces that can be integrated analytically in three dimensions during the estimation, and the remaining dimensions use then Monte Carlo Integration. This allows for

multiple importance sampling to mitigate singularities.

Combined Solutions

Considering that the previous methods have their strengths and weaknesses depending on the simulated effect, it makes sense to combine multiple of these techniques to achieve the best results possible. Hachisuka, Pantaleoni, and Jensen proposed in [HPJ12] a new path integral formulation using multiple importance sampling to combine photon mapping and path tracing. Afterwards, Křivánek et al. used in [Kři+14] different combinations from [Jar+11a] along with traditional path tracing to efficiently simulate different types of participating media. In [Hac+17], Hachisuka et al. expanded on [HPJ12] and [Kři+14] by adding more combinations of blurring kernels and using Monte Carlo path vertex estimation.

2.2 Path Tracing

The light transport equation from section 1.6.1 is quite complex to evaluate because of the implicit geometric relation given by the ray-casting function $t(\mathbf{x}, \vec{w})$. The integral in the equation can be reformulated to an integral over an area instead of an integral over directions on a sphere:

$$L(\mathbf{x}' \to \mathbf{x}) = L_e(\mathbf{x}' \to \mathbf{x}) + \int_A f_s(\mathbf{x}'' \to \mathbf{x}' \to \mathbf{x}) L(\mathbf{x}'' \to \mathbf{x}') G(\mathbf{x}'' \leftrightarrow \mathbf{x}') \, \mathrm{d}A(\mathbf{x}'')$$

with $G(\mathbf{x}'' \leftrightarrow \mathbf{x}')$ being the geometric term that represents both the visibility and the projected area of the solid angle between the points $\mathbf{x}''$ and $\mathbf{x}'$, and $f(\mathbf{x}'' \to \mathbf{x}' \to \mathbf{x})$ being the BSDF with the angles between $\mathbf{x}''$ and $\mathbf{x}'$, and $\mathbf{x}'$ and $\mathbf{x}$.

The previous equation can be used to further derive the foundation of Path Tracing. Instead of integrating over areas, it is more flexible to integrate over light paths in space. Given a position on the film plane $\mathbf{x}_0$ and an arbitrary point in the scene $\mathbf{x}_1$, the LTE can thus be formulated as:

$$L(\mathbf{x}_1 \to \mathbf{x}_0) = \sum_{n=1}^{\infty} P(\bar{\mathbf{x}}_n)$$

where $\bar{\mathbf{x}}_n$ is the path composed of the $n+1$ vertices $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n$ and $P(\bar{\mathbf{x}}_n)$ is the radiance scattered over the path:

$$P(\bar{\mathbf{x}}_n) = \underbrace{\int_A \int_A ... \int_A}_{n-1} L_e(\mathbf{x}_n \to \mathbf{x}_{n-1}) \times \left(\prod_{i=1}^{n-1} f_s(\mathbf{x}_{i+1} \to \mathbf{x}_i \to \mathbf{x}_{i-1}) G(\mathbf{x}_{i+1} \leftrightarrow \mathbf{x}_i) \right) \mathrm{d}A(\mathbf{x}_2) ... \mathrm{d}A(\mathbf{x}_n)$$

2.2.1 Original Technique

With the previous formulation, a single ray can be cast and reflected through the scene to accumulate radiance. Path tracing uses this property to estimate radiance coming at a pixel position in the final image. Additionally, participating media can be introduced in the same derivation process with the radiative transfer equation (Section 1.6.3) by considering volume scattering points as the vertices of a path. Analogously, surfaces can be interpreted as infinitely dense volumes with complex phase functions [Fon+17].

There is still the problem of the infinity in the summation. To solve this issue, Russian Roulette (Section 1.4.4) can be applied at each vertex of the path with termination probability q_i:

$$\frac{1}{1-q_1}\left(P(\bar{\mathbf{x}}_1)+\frac{1}{1-q_2}\left(P(\bar{\mathbf{x}}_2)+\frac{1}{1-q_3}\left(P(\bar{\mathbf{x}}_3)+...\right.\right.\right.$$

In theory, this technique works and converges to the correct result. However, choosing point pairs with equal probability will have an extremely slow convergence rate since, for any path segment in which its points have no visibility of each other, the radiance would be zero for its whole path. The same applies to paths that never sample light sources. Lastly, it would be impossible to sample the surface of emitters in the form equal or similar to those of point lights.

A Path Tracing algorithm should, therefore, construct paths incrementally with ray-tracing, guaranteeing the visibility between surfaces, and either always sample light sources when terminating the path or sample them at each surface intersection point.

2.2.2 Bidirectional Path Tracing

While still performing well in general situations, path tracing alone can have high variance in scenes with complex lighting arrangements - e.g. lights that are heavily occluded and mostly contribute with indirect illumination. Many of the traced paths in this case will not sample the light at any vertex. Bidirectional Path Tracing improves on the standard technique by tracing a path $\mathbf{q}_0, ..., \mathbf{q}_{s-1}$ originating from a light source in addition to the path $\mathbf{x}_0, ..., \mathbf{x}_{t-1}$ from the camera. This results in the following path:

$$\bar{\mathbf{x}} = \mathbf{q}_0, ..., \mathbf{q}_{s'-1}, \mathbf{x}_{t'-1}, ..., \mathbf{x}_0$$

where $s' \leq s$ and $t' \leq t$, and vertices at s' and t' are unoccluded w.r.t. each other.

Further refinements can be added to the algorithm as shown in [PJH17], such as reusing subpath combinations, avoiding the connection of paths where only one vertex from any of the two subpaths is used, and applying multiple importance sampling to combine different path generation approaches.

2.2.3 Participating Media

As presented in [LW96], both homogeneous and heterogeneous participating media can be included in the Path Tracing algorithm by combining random walk with direct sampling techniques. The method used in the publication for heterogeneous media was regular tracking. However, for this thesis, the more efficient delta tracking is used, as shown in Section 1.7.1.

2.3 Photon Mapping

Created by Jensen [Jen96], *Photon Mapping* can be used to efficiently simulate global illumination and caustics, and can be combined with other approaches to overcome disadvantages. Compared to Path Tracing, it has faster convergence times, and it approximates the flux distribution through the scene, instead of the irradiance like the methods presented in Section 2.1. It is however a biased method in its traditional implementation because it requires an infinite amount of photons to be traced into the scene to eliminate bias.

2.3.1 Original Technique

The original technique consists of two main passes, photon tracing, and radiance estimate. The first pass spreads photon points over the scene starting from the light sources. The second pass then gathers there photons to estimate the received radiance from the scene.

First Pass: Photon Tracing

The first step in this approach is to trace photons through the scene in a similar manner to path tracing. to better simulate different light phenomena, two types of photons are traced and stored in separated data structures: global photons and caustics photons. For each photon collision with the scene, its information (position, energy, and incident direction) is stored in the respective map - balanced kd-trees are used by the author in the original publication.

Caustics photons are shot towards specular reflecting objects and stored at diffuse surfaces. This is needed because to estimate the caustics radiance, the photons are evaluated directly and thus a large number of photons in the illuminated area are required. Global photons on the other hand are used indirectly and can be more sparsely distributed through the scene. Additional to both type of photons, shadow photons can be stored using the light rays at the intersections after the first one, to reduce the number of shadow rays traced in the second pass to compute direct illumination.

Second Pass: Radiance Estimate

To better formulate the solution, the LTE (Section 1.6.1) is split into a sum of multiple components (parameters are omitted for clarity):

$$\begin{aligned} L(\mathbf{x}, \vec{w}_o) = &\int_{S^2} f_r L_{i,l} |cos\theta_i| \mathrm{d}\vec{w}_i + \\ &\int_{S^2} f_{r,s}(L_{i,c} + L_{i,d}) |cos\theta_i| \mathrm{d}\vec{w}_i + \\ &\int_{S^2} f_{r,d} L_{i,c} |cos\theta_i| \mathrm{d}\vec{w}_i + \\ &\int_{S^2} f_{r,d} L_{i,d} |cos\theta_i| \mathrm{d}\vec{w}_i \end{aligned}$$

with

$$f_r = f_{r,s} + f_{r,d} \quad \text{and} \quad L_i = L_{i,l} + L_{i,c} + L_{i,d}$$

$L_{i,l}$ represents direct illumination from light sources, $L_{i,c}$ direct illumination from light sources through specular reflections (caustics), and $L_{i,d}$ diffusely reflected indirect soft illumination. $f_{r,d}$ and $f_{r,s}$ are respectively the diffuse and specular part of the BRDF.

For the direct illumination L_i, l, two approaches can be used: an accurate and an approximation. If there is a low number of shadow and direct illumination photons or a mixture of both close to a sample point in the scene, shadow rays are traced towards the light sources and these are evaluated directly. If these conditions are not met, only the global photon map is instead used.

The specular reflections are calculated through standard Monte Carlo ray-tracing combined with multiple importance sampling, and the caustics term is evaluated directly using the photons in the caustic map due to the inviability of Monte Carlo sampling

Lastly, the diffusely reflected radiance can be estimated with an accurate and an approximation similar to the first term. The approximation is based on the global photon map, and the accurate version uses multiple importance sampling based on the directions from the photon map.

To estimate the radiance leaving a point p in the direction w_o on an arbitrary

differential surface $\mathrm{d}A$, the following formulation is used:

$$\begin{aligned} L(\mathbf{x}, \vec{w}_o) &= \int_{S^2} f_r(\mathbf{x}, \vec{w}_o, \vec{w}_i) \frac{\mathrm{d}^2\Phi(\mathbf{x}, \vec{w}_i)}{\mathrm{d}A} \mathrm{d}\vec{w}_i \\ &\approx \sum_{p=1}^{N} f_r(\mathbf{x}, \vec{w}_o, \vec{w}_{i,p}) \frac{\Delta\Phi_p(\mathbf{x}, \vec{w}_{i,p})}{\pi r^2} \end{aligned}$$

The approximation of the equation represents a N-nearest neighbor search of surrounding photons (*ph*), using a sphere for $\mathrm{d}A$. According to [Jen96], fixed sized spheres with a varying number of photons is viable but delivers blurry results for a high photon density and wrong estimates for low density. In situations with low photon density, the author employs a cone-filter to reduce blurriness

2.3.2 Volumetric Photon Mapping

The extension introduced by Jensen and Christensen [JC98] made participating media simulation a part of photon mapping. From the radiative transfer equation (Section 1.6.2), a new formulation can be derived by integrating along a line from p_0 to p to estimate the radiance in participating media:

$$\begin{aligned} L(\mathbf{x}, \vec{w}_o) &= \int_{\mathbf{x}_0}^{\mathbf{x}} T_r(\mathbf{x}', \mathbf{x}) \sigma_a(\mathbf{x}') L_e(\mathbf{x}', \vec{w}_o) \mathrm{d}\mathbf{x}' \\ &+ \int_{\mathbf{x}_0}^{\mathbf{x}} T_r(\mathbf{x}', \mathbf{x}) \sigma_s(\mathbf{x}') L_i(\mathbf{x}', \vec{w}_o) \mathrm{d}\mathbf{x}' \\ &+ T_r(\mathbf{x}_0, \mathbf{x}) L(\mathbf{x}_0, \vec{w}_o) \mathrm{d}\mathbf{x}' \\ L_s(\mathbf{x}, \vec{w}_o) &= \int_{S^2} f_{ph}(\mathbf{x}', \vec{w}_o, \vec{w}_i) L_{in}(\mathbf{x}', \vec{w}_i) \mathrm{d}\vec{w}_i \end{aligned}$$

which represents with the first term the self-emitted radiance, the accumulated in-scattered radiance $L_s(\mathbf{x}, \vec{w}_o)$ in the second one and lastly the existing radiance at the beginning of the ray.

Just like the separation between global and caustics photon maps, a volume photon map is introduced. This improves performance by not inflating the size of the global photon map kd-tree and not having to categorize each photon as surface or volume.

The volumetric photon map has the capability of simulating direct and indirect illumination at a certain point in the volume. However, the authors state that direct illumination is better handled by traditional ray-tracing methods such as ray marching.

First Pass: Photon Tracing

Firstly, photons are traced into the scene from the light sources. Participating media, in this case, is represented by volumes, which are also considered in the ray-tracing collisions. If a volume is intersected, the photon has a chance to either interact with the medium or pass through it unaffected, which is decided by sampling its CDF:

$$P(\mathbf{x}) = 1 - T_r(\mathbf{x}_s, \mathbf{x})$$

where $\mathbf{x}_s$ is the entry point of the photon in the medium, and $\mathbf{x}$ is an arbitrary point along the ray. The transmittance is computed using ray marching. If an interaction occurs, Russian roulette is used to decide whether a photon is absorbed, in which case the tracing is terminated, or scattered, where a photon is deposited if it is not the first interaction with the medium. If there is no interaction, the ray-tracing simply moves on to the next intersection in the scene.

Second Pass: Radiance Estimate

Similar to how radiance is estimated for surfaces, the formula for participating media is:

$$L(\mathbf{x}, \vec{w}) = \frac{\mathrm{d}^2\Phi(\mathbf{x}, \vec{w})}{\sigma_s(\mathbf{x})\mathrm{d}\vec{w}\,\mathrm{d}V}$$

with a differential volume $\mathrm{d}V$ containing the photons instead of a differential area $\mathrm{d}A$, and the scattering coefficient σ_s. The volume is represented by the smallest sphere fitting the N-nearest neighboring photons.

Inserting this formulation into $L_s(\mathbf{x}, \vec{w}_o)$ gives:

$$\begin{aligned} L_{s,i}(\mathbf{x}, \vec{w}_o) &= \frac{1}{\sigma_s(\mathbf{x})} \int_{S^2} f_{ph}(\mathbf{x}, \vec{w}_o, \vec{w}_i) \frac{\mathrm{d}^2\Phi(\mathbf{x}, \vec{w}_i)}{\mathrm{d}\vec{w}_i\,\mathrm{d}V} \mathrm{d}\vec{w}_i \\ &\approx \frac{1}{\sigma_s(\mathbf{x})} \sum_{p=1}^{N} f_{ph}(\mathbf{x}, \vec{w}_o, \vec{w}_p) \frac{\Delta\Phi_p(\mathbf{x}, \vec{w}_p)}{\frac{4}{3}\pi r^3} \end{aligned}$$

which represents the indirect illumination at any given point $\mathbf{x}$ inside the volume towards the directions $\vec{w}_o$. Together with the direct light illumination in a volume $L_i, d(p, w_o)$, the in-scattered illumination is defined as:

$$L_s(\mathbf{x}, \vec{w}_o) = L_s, d(\mathbf{x}, \vec{w}_o) + \frac{\sigma_s(\mathbf{x})}{\sigma_t(\mathbf{x})} L_s, i(\mathbf{x}, \mathbf{x}_o)$$

Combined with ray marching, the final radiance estimate for points $\mathbf{x}_k$ along a ray inside a medium is:

$$\begin{aligned} L(\mathbf{x}_t, \vec{w}_o) &= \sigma_a(\mathbf{x}_t) L_e(\mathbf{x}_t, \vec{w}_o) \Delta \mathbf{x}_t \\ &+ \sigma_s(\mathbf{x}_t) L_s(\mathbf{x}_t, \vec{w}_o) \Delta \mathbf{x}_t \\ &+ \exp\left(-\sigma_t(\mathbf{x}_t) \Delta \mathbf{x}_t\right) L(\mathbf{x}_{t-1}, \vec{w}_o) \end{aligned}$$

with $\Delta \mathbf{x}_t = |\mathbf{x}_k - \mathbf{x}_{k-1}|$ being the step size and $\mathbf{x}_o$ the ray exit point of the volume or a surface point in the medium. In their implementation, the authors use an adaptive approach to halve the step size if the variation of the radiance between steps is too large.

2.3.3 Beam Radiance Estimate

Jarosz, Zwicker and Jensen improved the way the second pass of volumetric photon mapping gathers photons in [JZJ08]. In ray marching the gathered in-scattered radiance corresponds to:

$$L(\mathbf{x}, \vec{w}_o) \approx T_r(\mathbf{x}_s, \mathbf{x}) L(\mathbf{x}_s, \vec{w}_o) + \left(\sum_{t=0}^{S-1} T_r(\mathbf{x}_t, \mathbf{x}) \sigma_s(\mathbf{x}_t) L_{in}(\mathbf{x}_t, \vec{w}_o) \Delta \mathbf{x}_t \right)$$

This however does not present an accurate solution to the in-scattering integral:

$$\int_{\mathbf{x}_0}^{\mathbf{x}} T_r(\mathbf{x}', \mathbf{x}) \sigma_s(\mathbf{x}') L_{in}(\mathbf{x}', \vec{w}_o) \mathrm{d}\mathbf{x}'$$

The major contribution was to apply the measurement equation from [Vea97] and the generalized path integral formulation for participating media presented in [PKK00] to volumetric photon mapping.

The Measurement Equation

The *measurement equation* represents the quantity of incident radiance over rays in a scene:

$$I = < W_e, L > = \int_V \int_{S^2} W_e(\mathbf{x}, \vec{w}_i) L(\mathbf{x}, \vec{w}_i) \mathrm{d}\vec{w} \, \mathrm{d}V$$

with W_e being the importance function - an abstract "sensor" in the virtual scene. This is essentially a weighting function. Other methods can be represented with this formulation as well - e.g. Path Tracing and Radiosity.

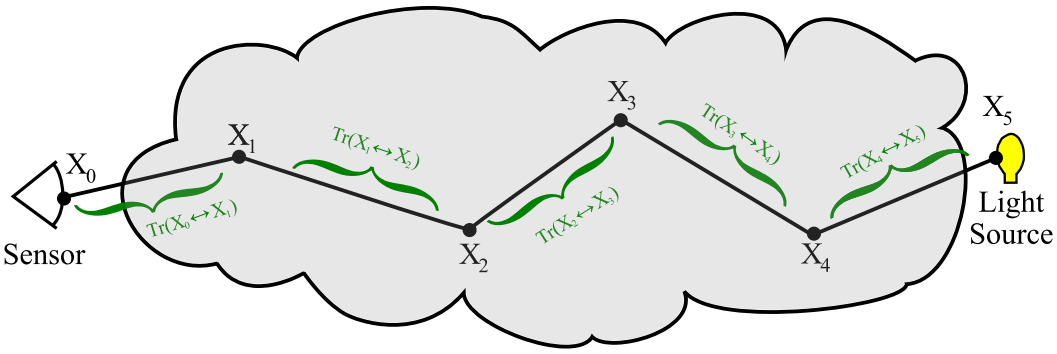


Figure 2.1: Light Path from Source to Sensor

In volumetric photon tracing, an unbiased estimation of the radiance in the scene can be seen as the discrete distribution of radiance using photons as sample points. This assumes that the scene is in an equilibrium state

In the publication [JZJ08] the authors derive a formulation of the equation for both surfaces and volumes using cases in some of the functions. For simplicity and focus on the use case of this thesis, only the cases relevant to volumetric rendering are presented here and all the vertices from a path are assumed to be mutually visible. The previous equation can be represented as radiance going out of differential areas, resulting in the following path Monte Carlo integration:

$$\mathrm{E}\left[\frac{1}{N}\sum_{i=1}^{N} W_e(\mathbf{x}_{i,0} \rightarrow \mathbf{x}_{i,1}) R_i\right]$$

with

$$R_i = \frac{\hat{L}_e(\mathbf{x}_{i,k_i} \rightarrow \mathbf{x}_{i,k_i-1})}{p(\mathbf{x}_{i,k_i}, \mathbf{x}_{i,k_i-1})} \prod_{j=1}^{k_i-1} \left(\frac{1}{q_{i,j}} \frac{f_{ph}(\mathbf{x}_{i,j+1} \rightarrow \mathbf{x}_{i,j} \rightarrow \mathbf{x}_{i,j-1}) \sigma_s(\mathbf{x}_{i,j+1}) T_r(\mathbf{x}_{i,j+1} \leftrightarrow \mathbf{x}_{i,j})}{p(\mathbf{x}_{i,j}, \mathbf{x}_{i,j-1}) \ \|\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j}\|^2} \right) \frac{\sigma_s(\mathbf{x}_{i,1}) T_r(\mathbf{x}_{i,1} \leftrightarrow \mathbf{x}_{i,0})}{\|\mathbf{x}_{i,1} - \mathbf{x}_{i,0}\|^2}$$

where $q_{i,j}$ is the Russian roulette probability of the path terminating at the vertex $\mathbf{x}_{i,j}$, and $\hat{L}_e$ the emission either from a light source, or the medium. The divisions by the squared length of the distance between two path vertices is the result of the conversion from solid angles to volume integration domain.

By setting $\alpha_i = R_i$ as the Monte Carlo weight, the measurement equation estimate is complete. Even though the derivation process is different from [JC98], there is an equivalence between the sample ray weights α_i and the photon power: $\Delta\Phi_i = \frac{\alpha_i}{N}$

Beam Radiance Estimate

The in-scattered radiance previously presented can be adjusted to be distributed over the ray space by integrating over the entire scene volume using the Dirac-delta function:

$$L_s(\mathbf{x}_t, \vec{w}_o) = \int_V \int_{S^2} \delta(\|\mathbf{x}' - \mathbf{x}\|) f_{ph}(\mathbf{x}', \vec{w}_o, \vec{w}_i) L(\mathbf{x}', \vec{w}_o) \, \mathrm{d}\vec{w}_i \, \mathrm{d}V(\mathbf{x}')$$

Setting $W_e = \delta(\|\mathbf{x}' - \mathbf{x}\|) f_{ph}(\mathbf{x}', \vec{w}_o, \vec{w}_i)$ transforms the above equation into the measurement equation, which can then be combined with the previously defined Monte Carlo integrator to create an unbiased estimator. In [Jen96] and [JC98] bias comes from the fact that the Dirac-delta function is replaced by a spherical blurring kernel:

$$L_s(\mathbf{x}_t, \vec{w}_o) \approx \frac{1}{N} \sum_{i=1}^{N} K_n(\|\mathbf{x}_i - \mathbf{x}_t\|) f_{ph}(\mathbf{x}_i, \vec{w}_o, \vec{w}_i) \alpha_i$$

with the kernel K_n defined as:

$$k_n(r) = \begin{cases} \frac{3}{4\pi d_n^3} & \text{if } r \in [0, d_n] \\ 0 & \text{otherwise} \end{cases}$$

where d_n is the distance to the n^{th} photon in the nearest neighbor search. Using the same approach, a derivation using a cylindrical kernel can be created. This is achieved by first reformulating the in-scattering along a ray equation to integrate over the whole volume in cylindrical coordinates

$$\int_{\mathbb{R}} \int_0^{2\pi} \int_{\mathbb{R}} \int_{S^2} \delta(r)(\mathrm{H}(t) - \mathrm{H}(t-s)) T_r(\mathbf{x} \leftrightarrow \mathbf{x}') \\ \sigma_s(\mathbf{x}') f_{ph}(\mathbf{x}', \vec{w}_o, \vec{w}_i) L_{in}(\mathbf{x}', \vec{w}_i) \, \mathrm{d}\vec{w}_i \, \mathrm{d}r \, \mathrm{d}\theta \, \mathrm{d}t$$

with $\mathrm{H}(x)$ being the Heaviside step function, $s = \|\mathbf{x} - \mathbf{x}_0\|$ the length of the cylinder, and (t, θ, r) the cylindrical coordinates from $(\mathbf{x}, \vec{w}_o)$.

Considering that photons will never land exactly on the ray, a blurring kernel must be used in place of the delta and step functions, introducing bias. The Monte Carlo estimator becomes then:

$$\frac{1}{N}\sum_{i=1}^{N} k(t_i, \theta_i, r_i) T_r(\mathbf{x} \leftrightarrow \mathbf{x}') \sigma_s(\mathbf{x}_i) f_{ph}(\mathbf{x}_i, \vec{w}_i, \vec{w}_o) \alpha_i$$

Since the integration already occurs over the ray, the kernel can be normalized in 2D.

Using a fixed radius kernel for the entire scene still introduces excessive blurring. The authors' solution to this problem was to use the duality formulation that the equivalent of finding all photon points inside a cylinder is the ray intersection of the spheres with the cylinder's radius and points' center positions. In that manner, each photon is assigned a radius based on the density in its neighborhood. However, this improvement comes at the cost of extra processing time to set each photon's radius. The Kernel used in the publication is:

$$k_i(\mathbf{x}, \vec{w}_o, s, \mathbf{x}_i, r_i) = \begin{cases} r_i^{-2} K_2(\frac{d_i}{r_i}) & \text{if } d_i \in [0, r_i] \\ 0 & \text{otherwise} \end{cases}$$

with d_i being the smallest distance from the photon i to the ray, and $K_2(x)$ the Silverman's two-dimensional bi-weight kernel [Sil86]:

$$K_2(x) = 3\pi^{-1}(1 - x^2)^2$$

2.3.4 Progressive Photon Mapping

Considering that the source of bias in photon mapping is the kernel blurring, if an infinite number of photons were to be traced, the solution would be unbiased. This is not possible in a single pass because photons are stored in memory. Hachisuka et al. presented in [HOJ08] a multi-pass photon mapping algorithm wherein each pass the result progressively converges to the real solution. The main idea is to increase the photon density while reducing the sampling radius in fixed gathering positions at each frame, thus heading towards an "infinite" number of photons. A major restriction of this algorithm is that it does not support participating media since the formulation is only for surface areas. However, it is the base for other improvements in later publications.

A new pass is introduced before photon tracing, which is a ray-tracing pass that creates gathering points for each pixel at the first intersection. This pass happens

only once at the beginning of the algorithm. Afterward, multiple photon tracing and radiance estimate passes occur in sequence to deliver an unbiased result.

Radius Reduction

In surface disc radiance estimate, the density estimate at a point $\mathbf{x}$ is defined as:

$$d(\mathbf{x}) = \frac{n}{\pi r^2}$$

where n is the number of photons found inside the disc, and r is the disc radius. Considering now that in the new density estimate the radius needs to be reduced while increasing the number of photons in the sampling area, a new density estimate can be formulated:

$$\hat{d}(\mathbf{x}) = \frac{N(\mathbf{x}) + M(\mathbf{x})}{\pi R(\mathbf{x})^2}$$

with $N(\mathbf{x})$ being the current number of photons sampled in the disc, $M(\mathbf{x})$ the newly added photons inside the same disc, and $R(\mathbf{x})$ the current radius of the disc at the sample position $\mathbf{x}$.

After reducing the radius, the photon density should increase as well. Given the reduced radius $\hat{R}(\mathbf{x}) = R(\mathbf{x}) - dR(\mathbf{x})$, $\hat{N}(\mathbf{x})$ photons should be inside the new area:

$$\hat{N}(\mathbf{x}) = \pi \hat{R}(\mathbf{x})^2 \hat{d}(\mathbf{x}) = \pi (R(\mathbf{x}) - dR(\mathbf{x}))^2 \hat{d}(\mathbf{x})$$

and additionally, to enforce the increase in the number of photons:

$$\hat{N}(\mathbf{x}) = N(\mathbf{x}) + \alpha M(\mathbf{x}), \quad \text{where } \alpha \in (0, 1)$$

Combining all the previous 3 equations delivers the radius reduction formula:

$$\hat{R}(\mathbf{x}) = R(\mathbf{x}) \sqrt{\frac{N(\mathbf{x}) + \alpha M(\mathbf{x})}{N(\mathbf{x}) + M(\mathbf{x})}}$$

Flux Correction

Since the radius reduces, the flux inside the radius needs to be adjusted as well. Each gathering point stores the total unnormalized flux from the current photons:

$$\tau_N(\mathbf{x}, \vec{w}_o) = \sum_{p=1}^{N(\mathbf{x})} f_r(\mathbf{x}, \vec{w}_o, \vec{w}_p) \Phi_p(\mathbf{x}_p, \vec{w}_p)$$

with w_o being the direction of the incident ray that created the gathering point at position x. Since keeping track of all the photons inside a gathering disc would be memory intensive. A better approach is to assume that the photon density and illumination inside the disc is constant, and scale the flux by the ratio between the old radius and the new radius:

$$\tau_{\hat{N}}(\mathbf{x}, \vec{w}_o) = (\tau_N(\mathbf{x}, \vec{w}_o) + \tau_M(\mathbf{x}, \vec{w}_o)) \frac{N(\mathbf{x}) + \alpha M(\mathbf{x})}{N(\mathbf{x}) + M(\mathbf{x})}$$

Progressive Radiance Estimate

The stored flux at the gathering points is still unnormalized. To estimate the radiance, they need to be divided by the total number of emitted photons $N_{emitted}$. This gives the following radiance estimate for each pass:

$$\begin{aligned} L(\mathbf{x}, \vec{w}_o) &\approx \frac{1}{\Delta A} \sum_{p=1}^{n} f_r(\mathbf{x}, \vec{w}_o, \vec{w}_p) \Delta\Phi_p(\mathbf{x}_p, \vec{w}_p) \\ &= \frac{1}{\pi R(\mathbf{x})^2} \frac{\tau(\mathbf{x}, \vec{w}_o)}{N_{emitted}} \end{aligned}$$

2.3.5 Parallel Progressive Photon Mapping on GPUs

Hachisuka and Jensen [HJ10] introduced an algorithm version of progressive photon mapping that runs on GPUs. The improvement works by introducing a spatial hash grid into the scene to store the photons. Since the memory constraint on the GPU is a problem, it is advantageous to limit the number of photons stored. Considering the probability of selecting a photon from n colliding photons in a grid cell to be:

$$p(\mathbf{x}) = \frac{1}{n}$$

Russian roulette can be applied and the selected photon is correspondingly divided by $p(\mathbf{x})$ in the radiance estimate pass. The selection process occurs in the photon tracing pass intrinsically through the parallel nature of GPUs. The only synchronization required is when incrementing the number of collisions happening in a cell.

2.3.6 Probabilistic Progressive Approach

Knaus and Zwicker analyzed in [KZ11] the progressive method for photon mapping using probability theory, and discovered that the radius reduction does

not depend on local statistics about the scene, and does not require any gathering process. Due to their formulation of the goal, it is also possible to directly simulate distributed effects like depth of field, and consider participating media with minor modifications.

Photon Mapping is a Monte Carlo estimation of the exitant radiance at a point in space in the form of:

$$L(\mathbf{x}, \vec{w}_o) \approx \frac{1}{M} \sum_{j=1}^{M} k_r(\mathbf{x}_j - \mathbf{x}) \gamma_j$$

where k_r is a kernel weighting function linearly scaled by a factor r, and γ_j is the contribution of a photon to the sum - that is, the photon power divided by its PDF and multiplied by the phase function/BRDF.

Error, Expected Value, and Variance

From the Monte Carlo formulation of the radiance estimate, the error function is defined as the difference between the true radiance value and the Monte Carlo estimate:

$$\epsilon(\mathbf{x}, r) = \frac{1}{M} \sum_{j=1}^{M} k_r(\mathbf{x}_j - \mathbf{x}) \gamma_j - L(\mathbf{x}, \vec{w}_o)$$

All the following derivations are made considering photon mapping on surfaces for simplicity, and the volume equivalent is given at the end.

Assuming that the PDF of the photons $p_l(x)$ inside the support of k_r to be constant and representing the photon values γ_j as independent samples of a random variable γ, the variance of the error is defined as:

$$\mathrm{Var}\left[\epsilon(\mathbf{x}, r)\right] \approx \frac{(\mathrm{Var}\left[\gamma\right] + \mathrm{E}\left[\gamma\right]^2) p_l(\mathbf{x})}{M r^2} \int_{\mathbb{R}^2} k(\psi)^2 \mathrm{d}\psi$$

Reformulating this equation, it is possible to derive a radius function given a desired variance:

$$r(x, \mathrm{Var}\left[\epsilon\right]) \approx \sqrt{\frac{(\mathrm{Var}\left[\gamma\right] + \mathrm{E}\left[\gamma\right]^2) p_l(x)}{M \mathrm{Var}\left[\epsilon\right]}}$$

Using the previous assumptions the expected value of the error is also shown to be:

$$\mathrm{E}\left[\epsilon(\mathbf{x}, r)\right] = r^2 \mathrm{E}\left[\gamma\right] \tau$$

for some constant τ that depends on higher-order derivatives of $p_l(\mathbf{x})$.

Problem Formulation

The main goal of the rendering algorithm is to compute the pixel color values in the form of:

$$c = \int \int W(\mathbf{x}, \vec{w}) L(\mathbf{x}, w) \mathrm{d}\mathbf{x} \mathrm{d}\vec{w}$$

where $L(\mathbf{x}, \vec{w})$ is the reflected radiance over all points and directions in the scene, and $W(\mathbf{x}, \vec{w})$ describes the contribution of the reflected radiance to the pixel value, allowing anti-aliasing, motion blur, depth of field, and glossy reflections. $W(\mathbf{x}, \vec{w})$ is calculated using ray-tracing.

In the original photon mapping, the main issue is that a trade-off between variance (noise) and expected error (bias) must be decided, i.e. noise can be reduced by introducing stronger blurring kernels. As mentioned in the publication [HOJ08], the progressive approach provides a solution where at the infinite limit of the photon count, both bias and noise vanish by converging to the true flux distribution in the scene.

Considering the pixel color equation as a Monte Carlo integral, we obtain:

$$\bar{c}_N = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{p_e(\mathbf{x}_i, \vec{w}_i)} W(\mathbf{x}_i, \vec{w}_i)(L(\mathbf{x}_i, \vec{w}_i) + \epsilon_i)$$

where $p_e(\mathbf{x}_i, \vec{w}_i)$ is the probability of tracing a ray towards the intersection point $\mathbf{x}_i$ from direction $\vec{w}_i$ coming from the camera, and ϵ_i the radiance estimation error of the i-th sample. The average error of the photon map radiance estimate with N samples can, therefore, be defined as:

$$\bar{\epsilon}_N = \frac{1}{N} \sum_{i=1}^{N} \epsilon_i$$

With these equations defined, the problem to be solved is thus:

$$\begin{gathered} \mathrm{Var}\left[\bar{\epsilon}_N\right] \to 0 \Rightarrow \mathrm{Var}\left[\bar{c}_N\right] \to 0 \ \text{ and} \\ \mathrm{E}\left[\bar{\epsilon}_N\right] \to 0 \Rightarrow \mathrm{E}\left[\bar{c}_N\right] \to c \ , \\ \text{as } N \to \infty \end{gathered}$$

where

$$\mathrm{Var}\left[\bar{\epsilon}_n\right] = \frac{1}{N^2} \sum_{i=1}^{N} \mathrm{Var}\left[\epsilon_i\right] \ \text{ and } \ \mathrm{E}\left[\bar{\epsilon}_n\right] = \frac{1}{N} \sum_{i=1}^{N} \mathrm{E}\left[\epsilon_i\right]$$

Probabilistic Radius Reduction

It can be seen from the previous average variance that it will converge to zero if each individual variance stays constant. This does not allow the average error to vanish. In a similar manner to [HOJ08] the variance is slightly increased at each sample with a constant $\alpha \in (0, 1)$:

$$\frac{\text{Var}\left[\epsilon_{i+1}\right]}{\text{Var}\left[\epsilon_i\right]} = \frac{i+1}{i+\alpha}$$

The α parameter decides how quickly the variance is increased per iteration. Given this formula, the sequence of i variances can then be defined as:

$$\text{Var}\left[\epsilon_i\right] = \text{Var}\left[\epsilon_1\right] \left(\prod_{k=1}^{i-1} \frac{k}{k+\alpha}\right) i, \text{ with } i > 1$$

where $\text{Var}\left[\epsilon_1\right]$ is the variance of the first iteration. From this formulation the variance of the average error is:

$$\text{Var}\left[\bar{\epsilon}_N\right] = \frac{\text{Var}\left[\epsilon_1\right]}{N^2} \left(1 + \sum_{i=2}^{N} \left(\prod_{k=1}^{i-1} \frac{k}{k+\alpha}\right) i\right)$$

Since the variance is inversely proportional to the square radius, as presented in $r(x, \text{Var}\left[\epsilon\right])$, the sequence of radius given the sequence of variances is:

$$\frac{r_{i+1}^2}{r_i^2} = \frac{\text{Var}\left[\epsilon_i\right]}{\text{Var}\left[\epsilon_{i+1}\right]}$$

which can be given explicitly as:

$$r_i^2 = r_1^2 \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k}\right) \frac{1}{i}$$

The expected error is proportional to the square radius, giving the sequence:

$$\text{E}\left[\epsilon_i\right] = \text{E}\left[\epsilon_1\right] \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k}\right) \frac{1}{i}$$

and the average expected error:

$$\text{E}\left[\bar{\epsilon_N}\right] = \frac{\text{E}\left[\epsilon_1\right]}{N} \left(1 + \sum_{i=2}^{N} \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k}\right) \frac{1}{i}\right)$$

The authors prove that these formulations give the same radii sequence as the approach proposed by [HOJ08] without any need for local statistics. The only needed parameters are the number of photons per iteration, the initial radius, and the α value. Furthermore, the generic formulation makes it is possible to use other photon mapping techniques as a black box combined with this algorithm.

Volumetric Changes

All the previous defined equations are trivially evaluated for volumetric kernels as well by including attenuation in the weighting term $W(x, w)$, and integrating over 3D space instead of 2D. This gives the radii sequence:

$$\frac{r_{i+1}^3}{r_i^3} = \frac{\text{Var}\left[\epsilon_i\right]}{\text{Var}\left[\epsilon_{i+1}\right]}$$

and has the same convergence properties of the surface formulation.

2.4 Higher-Dimensional Photon Maps

As mentioned before, photons represent the flux distribution in the scene through discrete samples. These samples so far have been considered to be points in space with incident direction and power. Jarosz et al. presented in [Jar+11a] how these sampled can be expanded to beams and combined with different sampling strategies. In [Jar+11b] a progressive algorithm was introduced to photon beams, and lastly Bitterly and Jarosz showed in [BJ17] that more spatial dimensions can be introduced to improve photon mapping even further.

This section explains how photon beams work in detail and briefly presents the more recent higher-dimensional photon samples.

2.4.1 Photon Beams

Photon beams are based on the photon marching technique, which is essentially a ray marching algorithm from the light sources. However, this technique is only used in the mathematical derivation of the formulas, and traditional random walk processes are used in its implementations.

Photon Marching

Through usage of ray marching, many photon samples are created along at uniform intervals along a ray. This is called by the authors a "discrete photon beam" and is defined as:

$$\Phi_{p_b} = \sigma_s \exp\left(-\sigma_t |\mathbf{x}_b - \mathbf{x}_t|\right) \Phi_b \Delta t$$

with Φ_b being the flux of the beam before the medium. Each photon position along this beam is then defined as:

$$\mathbf{x}_t = \mathbf{x}_b + t\vec{w}_b, \quad \text{with } t = [0, ..., N-1]\Delta t$$

where N is the number of photons in the beam. For multiple scattering, a scattering point is chosen along the beam using mean-free path calculations. Heterogeneous media can employ delta-tracking or the inversion method. With the new scattering position, a new beam is created in a direction sampled from the phase function and with initial power $\Phi_{b_{new}} = \frac{\sigma_s}{\sigma_t}\Phi_b$. Russian roulette can be applied, removing the albedo coefficient from the new power equation.

Photon Beam Radiance Estimate

Photon beams can be samples either using sampling points or beams similar to [JZJ08]. Each type can also be blurred differently by using kernels that operate in different numbers of dimensions. The kernel used does not change the correctness of the algorithm, just the resulting bias and variance. For clarification, the point sampling with spherical kernel is first shown before the beam sampling. Additionally, all the formulations assume homogeneous media but can be easily transformed by moving the scattering and extinction coefficients inside the integrations and summations.

Point Sampling

Given a sampling point, its sampling region R, a 3D spherical blurring kernel k_r and an arbitrary number of intersecting photon beams, the in-scattered radiance is computed as:

$$L_s(\mathbf{x}, \vec{w}) \approx \frac{1}{\sigma_s k_r(r^3)} \sum_{b \in R} \sum_{p_b \in R} f_{ph}(\theta_{p_b}) \Phi_{p_b}$$

Inserting the beam power equation into the previous one, and considering that photons share initial power and incident direction results in:

$$L_s(\mathbf{x}, \vec{w}) \approx \frac{1}{k_r(r^3)} \sum_{b \in R} f_{ph}(\theta_b) \Phi_{p_b} \sum_{p_b \in R} e^{-\sigma_t t_{p,b}} \Delta t$$

Taking the distance between photons towards 0 gives the equation for a continuous photon beam:

$$L_s(\mathbf{x}, \vec{w}) \approx \frac{1}{k_r(r^3)} \sum_{b \in R} f_{ph}(\theta_b) \Phi_b \lim_{\Delta t \to 0} \sum_{p_b \in R} e^{-\sigma_t t_{p,b}} \Delta t$$

Considering that the limit is equivalent to a Riemann sum integration, the final in-scattered radiance for a spherical sampling strategy is:

$$L_s(\mathbf{x}, \vec{w}) \approx \frac{1}{k_r(r^3)} \sum_{b \in R} f_{ph}(\theta_b) \Phi_b \int_{t_b^-}^{t_b^+} e^{-\sigma_t t_b} \mathrm{d}t_b$$

where t_b^- and t_b^+ are the intersecting points on the beam with the sphere. Further blurring kernels are presented in the paper [Jar+11a] but are not relevant for this thesis.

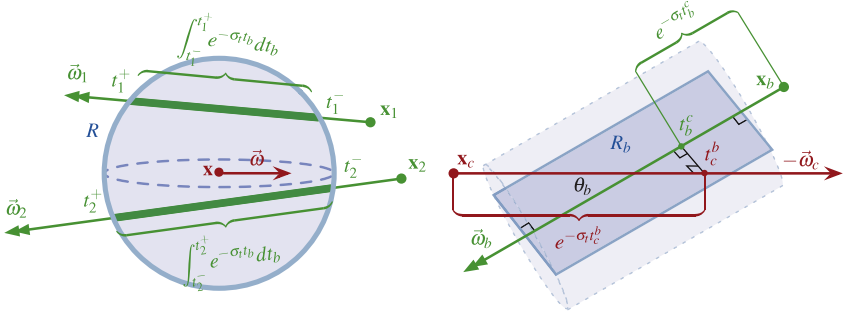


Figure 2.2: Photon Beam using Point Sampling (3D Blur) and Beam Sampling (1D Blur). Source [Jar+11a]

Beam Sampling

To estimate the radiance of a photon beam using the beam radiance estimate and a 3D blurring kernel, the in-scattered radiance from Point Sampling can be directly used with the beam radiance integral along the camera ray:

$$
\begin{aligned}
L_{in}(\mathbf{x}_c, \vec{w}_c, s) &\approx \frac{\sigma_s}{k_r(r^3)} \int_0^s e^{-\sigma_t t_c} \sum_{b \in R} f_{ph}(\theta_b) \Phi_b \int_{t_b^-(t_c)}^{t_b^+(t_c)} e^{-\sigma_t t_b} \mathrm{d}t_b \mathrm{d}t_c \\
&= \frac{\sigma_s}{k_r(r^3)} \sum_{b \in R} f_{ph}(\theta_b) \Phi_b \int_{t_c^-}^{t_c^+} \int_{t_b^-(t_c)}^{t_b^+(t_c)} e^{-\sigma_t (t_b + t_c)} \mathrm{d}t_b \mathrm{d}t_c
\end{aligned}
$$

with t_c^- and t_c^+ being the overlapping region of the extruded kernel along the photon beam with the camera ray, and $t_b^-(t_c)$ and $t_b^+(t_c)$ the beam length corresponding to this region. This formulation could be solved analytically for homogeneous media, but in the heterogeneous case, it becomes difficult to evaluate due to the double integration.

To improve the efficiency of the evaluation, a blurring kernel with lower dimensionality can be applied [Jar+11a]. For 2D kernels, two approaches can be taken: either integrate the 2D kernel along the camera ray intersection, expanding beams into cylinders, or using discrete photon beams, projecting each photon point onto the camera ray, and taking the limit of Δt to 0 similarly to the Point Sampling approach - this also expands the beam into a cylinder, but with the caps facing the camera instead.

Of much more interest to this thesis is the 1D blur dimensionality, which is also presented in [Jar+11a] since it has even better performance than both of the 2D blurring approaches.

Given a 2D blurring kernel in the shape of a rectangle with height h and width w, and the discrete formulation of photon beams, the in-scattered radiance can be reformulated to:

$$L_s(\mathbf{x}_c, \vec{w}_c, s) \approx \frac{\sigma_s}{hw} \sum_{b \in R_b} \sum_{p_b \in R_b} f_{ph}(\theta_{p_b}) \Phi_{p_b} e^{-\sigma_t t_{p,c}}$$

Inserting the photon beam power equation:

$$L_s(\mathbf{x}_c, \vec{w}_c, s) \approx \frac{\sigma_s}{hw} \sum_{b \in R_b} f_{ph}(\theta_b) \Phi_b \sum_{p_b \in R_b} e^{-\sigma_t t_{p,b}} e^{-\sigma_t t_{p,c}} \Delta t$$

to take the limit to 0 of the distance between the discrete photon beams, the height of the kernels must be adjusted to not overlap. The height is replaced by $h = \Delta t \sin\theta_b$ where θ_b is the angle between the photon beam direction and the camera direction. This makes the camera ray intersect a single kernel per beam and gives the equation:

$$L_s(\mathbf{x}_c, w_c, s) \approx \lim_{\Delta t \to 0} \frac{\sigma_s}{\Delta t w} \sum_{b \in R_b} \frac{f_{ph}(\theta_b) \Phi_b e^{-\sigma_t t_{p,b}} e^{-\sigma_t t_{p,c}}}{\sin\theta_b} \Delta t$$

Replacing w by the abstract notation $k_r(r)$, the solution is thus:

$$L_s(\mathbf{x}_c, \vec{w}_c, s) \approx \frac{\sigma_s}{k_r(r)} \sum_{b \in R_b} \frac{f_{ph}(\theta_b) \Phi_b e^{-\sigma_t t_{p,b}} e^{-\sigma_t t_{p,c}}}{\sin\theta_b}$$

Theoretically, if the camera ray is parallel to the photon beam, a singularity would be produced due to the $\sin\theta_b$ as a divisor, and the transmittance would not be able to be evaluated since the closest point would not be explicitly defined. The authors state however that in practice this is not a problem.

Photon Differentials & Global Radius Scaling

In previous techniques [JZJ08] [JC98] the radius of the samples is adjusted to compensate for the variation of sample density over the scene. For photon beams, photon differentials are used [SSF08]. In addition to the main ray, two auxiliary rays are traced with offset origins and directions from the main one. These three rays form a truncated cone that defines the kernel radius at each point along the photon beam. At each scattering point, the radius of the photon beam at that position is used as the radius of the new photon beam, and the differential rays' directions are set to be parallel to the main ray.

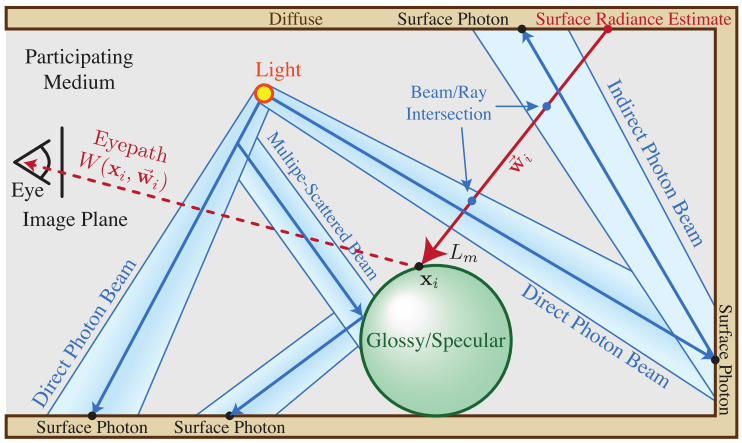


Figure 2.3: Scene with Photon Beams. L_m represents the radiance through the medium. Source [Jar+11b]

For surface interaction, further considerations can be taken into account [Jar+11a], but since the thesis' use case does not contain any surface this is not presented here.

Photon differentials are used at the light source emission to cover as much of the emission angle and area. The light's direction PDFs scaled by the number of photon beams emitted is used to set the differential rays' directions:

$$SolidAngle = \frac{1}{p_{light}(\vec{w}) \cdot N}$$

and the positions are set using a jittered grid, where the main rays' positions are the centers and the differentials are positions on the edges.

A global scaling factor R is used to change the radius of each photon beam uniformly, giving users the possibility to test the results from different photon beam widths.

Photon Beam Storage

Due to the amount of collision detection tests needed in the scene, the authors state that a *bounding volume hierarchy* (BVH) is the preferred data structure to store photon beams. Since many beams may overlap each other, each beam is split into multiple parts with unit aspect ratio. The BVH construction algorithm used is the *surface area heuristic* (SAH) [MB90].

2.4.2 Progressive Photon Beams

Given the Photon Beam combined with Beam Gathering and 1D Blur from [Jar+11a], a progressive version of photon beams was introduced by Jarosz et al. in [Jar+11b] by applying the same technique from [KZ11].

Beam Variance and Error

For simplicity, the incident radiance of a single photon beam along a ray is equivalently reformulated as:

$$L_{in}(\mathbf{x}, \vec{w}, r) \approx k_r(u)\sigma_s(\mathbf{x})\Phi_b T_r(\vec{w})T_r(\vec{v})\frac{f_{ph}(\vec{w}\cdot\vec{v})}{\sin(\vec{w},\vec{v})}$$

where v is the beam direction, and u is the distance of the vector connecting the closest points from the photon beam and the ray. The Monte Carlo estimation $\bar{c}_N$ for the pixel value c:

$$\bar{c}_N = \frac{1}{N}\sum_{i=1}^{N}\frac{W(\mathbf{x}_i, \vec{w}_i)L_{in}(\mathbf{x}_i, \vec{w}_i)}{p(\mathbf{x}_i, \vec{w}_i)}$$

The beam radiance approximation is defined as:

$$L_{in}(\mathbf{x}, \vec{w}, r) = k_r(u)\gamma + \epsilon(\mathbf{x}, \vec{w}, r)$$

where ϵ is error function: the difference between the real radiance and the approximated one using a kernel, and γ are the remaining terms collapsed in a single variable.

To apply the same technique from [KZ11], the distances u are considered to be independent and identically distributed samples from a random variable U with density $p_U^{\vec{w}}$. Equally, γ is also assumed to be i.i.d sampled and independent from u. The variance of a single beam is thus shown to be:

$$\mathrm{Var}\left[\epsilon(\mathbf{x}, \vec{w}, r)\right] = \frac{(\mathrm{Var}\left[\gamma\right] + \mathrm{E}\left[\gamma\right]^2)p_U^{\vec{w}}(0)}{r}C_1$$

where C_1 is a constant from the kernel, and $p_U^{\vec{w}}(0)$ the probability of the ray intersecting the photon beam. The variance thus increases linearly w.r.t. the kernel radius. The expected error is defined with some constant C_2 as:

$$\mathrm{E}\left[\epsilon(\mathbf{x}, \vec{w}, r)\right] = r\,\mathrm{E}\left[\gamma\right]C_2$$

thus decreasing linearly with the kernel radius.

For multiple beams, the varying radii used by photon differentials need to be taken into account. Given M beams, the radiance becomes:

$$L_{in}(\mathbf{x}, \vec{w}, r1, ..., r_M) = \frac{1}{M} \sum_{j=1}^{M} k_{r_j}(u_j)\gamma_j - \epsilon(\mathbf{x}, \vec{w}, r_1, ..., r_M)$$

and the variance derive from this is:

$$\mathrm{Var}\left[\epsilon(\mathbf{x}, \vec{w}, r1, ...r_M)\right] = \frac{1}{M} \frac{(\mathrm{Var}\left[\gamma\right] + \mathrm{E}\left[\gamma\right]^2) p_U^{\vec{w}}(0)}{r_H} C_1$$

with r_H being the harmonic mean of the radii:

$$\frac{1}{r_H} = \frac{1}{M} \sum \frac{1}{r_j}$$

The expected error becomes:

$$\mathrm{E}\left[\epsilon(\mathbf{x}, \vec{w}, r_1, ..., r_M)\right] = r_A \, \mathrm{E}\left[\gamma\right] \; C_2$$

with r_A being the arithmetic mean of photon beam radii.

Convergence of the Average Variance

Using the formula previously presented by [KZ11], convergence is achieved when the following is enforced:

$$\frac{\mathrm{Var}\left[\epsilon_{i+1}\right]}{\mathrm{Var}\left[\epsilon_i\right]} = \frac{i+1}{i+\alpha}$$

Given a first pass variance $\mathrm{Var}\left[\epsilon_1\right]$, the sequence of variances is:

$$\mathrm{Var}\left[\epsilon_i\right] = \mathrm{Var}\left[\epsilon_1\right] \left(\prod_{k=1}^{i-1} \frac{k}{k+\alpha} \right) i$$

The variance of the average error after N passes, which vanishes for $N \to \infty$, is thus:

$$\mathrm{Var}\left[\bar{\epsilon}_N\right] = \frac{\mathrm{Var}\left[\epsilon_1\right]}{N^2} \left(1 + \sum_{i=2}^{N} \left(\prod_{k=1}^{i-1} \frac{k}{k+\alpha} \right) \right)$$

Radius Reduction

With the convergence proven, the global scaling factor R from [Jar+11a] can be used to progressively reduce the radius. Using the same proportion from [KZ11], a sequence of radius scaling factors R_i can be derived from

$$\frac{R_{i+1}}{R_i} = \frac{\mathrm{Var}\left[\epsilon_i\right]}{\mathrm{Var}\left[\epsilon_{i+1}\right]} = \frac{i+\alpha}{i+1}$$

as

$$R_i = \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k}\right) \frac{1}{i}$$

Convergence of the Average Error

The expected error is proportional to the average radius. Given the error of the first pass ϵ_1:

$$\mathrm{E}\left[\epsilon_i\right] = \mathrm{E}\left[\epsilon_1\right] R_i$$

which gives the vanishing equation for $N \to \infty$ similar to the one from [KZ11]:

$$\mathrm{E}\left[\bar{\epsilon}_i\right] = \frac{\mathrm{E}\left[\epsilon_1\right]}{N} \left(1 + \sum_{i=2}^{2} \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k}\right) \frac{1}{i}\right)$$

Heterogeneous Media Considerations

To calculate the transmittance along a photon beam in heterogeneous media, the authors use a sum of Heaviside step functions that converges to the true transmittance function after averaging it through multiple passes. Given a distance parameter s along a photon beam and n previously calculated delta-tracking distance samples, the approximated transmittance can be calculated by:

$$T_r(\mathbf{x}, \vec{w}, s) = \mathrm{E}\left[\frac{1}{n} \sum_{j=0}^{n} \mathrm{H}(\mathrm{d}(\mathbf{x}, \vec{w}) - s)\right]$$

This is stored in a structure that the authors relate to *deep shadow maps* [LV00] and called it *progressive deep shadow maps*. For each beam traced in a pass, n samples are stored in the shadow map to later evaluate the transmittance in the radiance estimate pass. This approach maintains the convergence of the variance and the expected value of the average error. The only difference is the increase in variance in each pass.

2.5 Precomputed Transmittance

When evaluating radiance through participating media, the transmittance from the emitting lights to points in the volume needs to be evaluated multiple times. This cost can be reduced by pre-computing this value for static and media in the scene. In this section, two related approaches are presented.

2.5.1 Shadow Maps

Williams presented in [Wil78] a way of knowing if a point in space is occluded from a light source by rendering the scene once from the light perspective. This would store the normalized distance from the occluding geometry to the light in a texture. In the final rendering pass, this texture can be sampled to determine if another surface distance to the light is after or before the stored value. The result determines if it is located in the shadow or not w.r.t. to the specific light - hence the name *shadow map*.

This approach cannot handle participating media and is more efficient for non-transparent surface occlusion.

2.5.2 Deep Shadow Maps

Lokovic and Veach built upon the technique in [LV00] to render hair strands, fur, and smoke. Instead of storing just one value in a 2D image, multiple values are stored at each pixel. This results in a 3D texture. The publication handles both surface and volume transmittance. In this thesis use case, only the volume transmittance is considered. Given a volumetric extinction coefficient dataset, the transmittance function along a ray is calculated as:

$$T_r(z) = \exp\left(-\int_0^z \sigma_t(z')\mathrm{d}z'\right)$$

This function can be discretized into multiple linear segments defined by:

$$T_{r,i} = \exp\left(\frac{-(z_{i+1} - z_i)(\sigma_{t,i+1} + \sigma_{t,i})}{2}\right)$$

Any values between two vertices of a segment can then be interpolated to get an approximation of the transmittance until the distance z along a ray.

3 Implementation

The techniques in the methodology (Chapter 2) are all mathematically formulated. These still need to be combined and converted to an actual computer program to produce images, which has its inherent challenges.

This chapter describes the practical part of this thesis. It introduces the main tools that were used in the process. Additionally, it explains the general structure of the renderer and shows the implementation of the methods shown.

3.1 Rendering Backend and Debugging Tools

For the implementation of the previously mentioned methods, the Vulkan API was chosen to achieve high performance to allow interactive framerates. Since most of the material available online as reference is written in OpenGL, this was also the choice for the shading language. Dear ImGui is also integrated into the renderer to tweak small settings without the need to recompile the entire program. All of the debugging process involved using RenderDoc in situations where it supported or manually changing the shaders to discover the location of bugs.

3.1.1 Vulkan

Launched in 2016 by the Khronos Group, Vulkan [Khr20b] is aimed to be a low-overhead compute and graphics API with support across multiple platforms. It allows developers to have more control over the GPU and achieve better performance by balancing the GPU and CPU usage. Unlike in previous generations APIs, the developer must explicitly allocate memory for each resource used by the GPU. Furthermore, hardware commands must be also allocated before being submitted to the GPU. It uses shader format SPIR-V, which is a pre-compiled binary version of shaders. LunarG's Vulkan SDK [Lun20] also comes with an HLSL/GLSL to SPIR-V compiler.

In 2018 Vulkan version 1.1 was released with new features such as API ray-tracing for more recent GPUs and multi-GPU support, and In 2020 version 1.2 came out with more extensions integrated into the base API.

3.1.2 GLSL

Graphics Library Shading Language (GLSL) [Khr20a] is a programming language tailored for programming the GPU graphics and compute pipelines. It uses a C-Style syntax and is supported across multiple platforms.

3.1.3 Dear ImGui

Immediate Mode Graphical User Interface (ImGui) are frameworks that provide instant updates when changes happen in the GUI. Dear ImGui [Cor20] is a fast, portable, renderer agnostic and self-contained, allowing for fast creation of debug tools. It supports many rendering APIs, including Vulkan at the time of this writing.

3.1.4 RenderDoc

Created by Karlsson, RenderDoc [Kar20] is a graphics and compute debugging tool for the GPU. It can replay draw calls sent to the GPU during a specific frame and supports shader debugging - except for Vulkan compute shaders at the time of this writing. This program proved to be a powerful tool during the development of this thesis.

3.2 Renderer

This renderer uses LunarG's Vulkan API for Windows, combined with GLFW [GLF20] for window management. The IDE used was Visual Studio 2019 [Mic20b], and shaders are written using GLSL.

3.2.1 Vulkan Abstraction Classes

An abstraction layer was built on top of the Vulkan API to make the development faster. All the classes related to the API have the *Vulkan* prefix - e.g. VulkanGraphicsPipeline. Internally, they make the necessary API calls to allocate and free buffers and images, pipelines, command pools, and so on.

3.2.2 Initialization

The first step when starting the renderer is to initialize GLFW and create a system window. Afterward, Vulkan is initialized along with resource acquisition from the system. Through the usage of a Vulkan extension, a render surface is acquired from the system. At this step the rendering resources, such as the deep shadow map 3D texture, are pre-allocated.

After the resources have been allocated, the swapchain is created with the graphics and compute pipelines provided by ImGui and the RenderTechnique classes. Lastly, all the synchronization semaphores and fences are created. With this last step the initialization is done and the renderer moves on to the rendering loop.

3.2.3 Render Loop

The main loop starts by polling window events and updating the time-related statistics. Secondly, it invokes ImGui specific commands to update the UI and receive user input. Afterward, a surface image is requested from the swapchain. Lastly, it proceeds to create the graphics draw commands for the UI Layer provided by ImGui, and queues the compute commands for the current RenderTechnique and sends the graphics and compute command buffers to the GPU.

The loop then repeats itself, only synchronizing when requesting a surface from the swapchain to prevent acquiring an image before the previous render is complete. Image 3.1 illustrates this process.

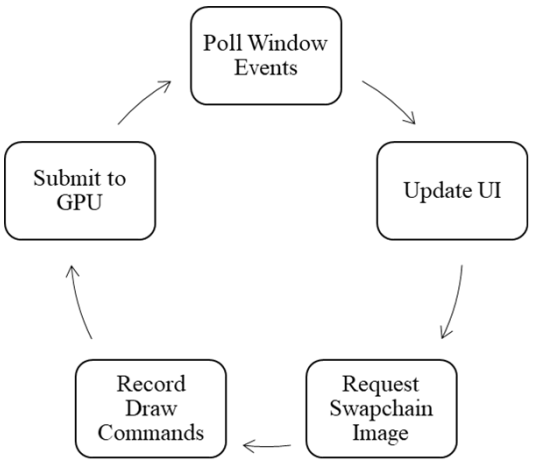


Figure 3.1: Render Loop

3.2.4 Render Techniques

The RenderTechnique class is an abstract base class for each of the algorithms implemented in this work. In theory, each technique could be swapped on the fly while the renderer is running, but due to time constraints and this feature not being a high priority, this was not implemented. Therefore, to change a rendering technique the program must be recompiled.

The following techniques exist:

- Path Tracing: *RenderTechniquePT*
- Deep Shadow Maps: *RenderTechniqueSV*
- Progressive Photon Mapping: *RenderTechniquePPM*
- Progressive Photon Beams: *RenderTechniquePPB* (Unfinished)

A RenderTechnique contains and maintains all the resources that are related to the specific technique - e.g. RenderTechniquePPM is the owner of its photon map, and RenderTechniqueSV allocates its 3D texture.

3.2.5 Host Data Structures

The class Grid3D holds a uniform 3D grid of floats, to load cloud density data from the disk and pass it on to the GPU. It contains also a GetMajorant() function to return the highest value in the grid for the delta tracking algorithm. The data

loaded from the disk is artificially scaled by a factor of 1000 to produce easier to read values without negative exponents. This is taken into consideration when using any scatter, extinction, or absorption coefficient.

3.2.6 User Interface

There are a couple of parameters that the user can tweak to get different render results:

- Henyey-Greenstein: the g parameter of the Henyey-Greenstein phase function.
- Position: the camera world position.
- Rotation: the camera world rotation.
- FOV: the camera horizontal field of view. The aspect ratio is locked at 16:9.
- Resolution: the render resolutions, either 800x600 or 1920x1080.
- Direction: the sunlight direction.
- Intensity: the sunlight intensity.
- Density: the cloud data density scaling.

Files with the .XYZ extension can be loaded from the "models" folder in the project's root folder using the "Cloud File" window. "Rendering Stats" shows the current Frame Count, the elapsed time since the beginning of the rendering, and the current ms/frame.

3.3 Path Tracing Implementation

The path tracing algorithm is used as an image reference for the other algorithms in this thesis. Its shader file is *PathTracer.comp*. A pseudo-code is shown in listing 3.1.

The algorithm starts by sampling a camera ray. Its start position is the center of the pixel and the direction is chosen based on the pinhole camera model. If a ray intersects the cloud grid, it starts the path tracing loop at the closest intersection point with the volume, otherwise, it directly samples the background.

Inside the volume, the next scatter point is calculated via delta tracking with scaled density values. If the ray leaves the volume, it samples the background in the given direction and exits the loop. If it is absorbed by the medium, it directly exits the loop. Otherwise, it proceeds to calculate the direct sunlight at the scatter point. Using the phase function, the radiance is accumulated and the next scatter direction is set. The loop repeats itself until the camera ray is either absorbed or leaves the medium.

The accumulated radiance is then averaged progressively with the existing values in the output image.

Listing 3.1 Path Tracing Implementation

```
Color radiance;
Ray cameraRay = getCameraRay(pixelCoordinates);
if(intersectCloud(cameraRay))
{
    sampleBackground(cameraRay);
}
else
{
    while(true) // Path Tracer Loop
    {
        getNextPosition(cameraRay);
        if(leftVolume(cameraRay)
        {
            radiance += sampleBackground(cameraRay)
            break;
        }

        radiance += SUNLIGHT_POWER * samplePhaseFunction(-cameraRay.
            dir, sunlightDirection) * sampleShadowMap(cameraRay.pos);

        getScatterDirection(cameraRay.dir);
    }
}

radiance = (oldRadiance * frameCount + radiance) / (frameCount + 1);
storeResult(radiance, pixelCoordinates);
```

3.3.1 Delta Tracking

Some minor changes have been added to the delta tracking algorithm to add support for varying density. Instead of directly using the sample from the cloud density 3D image, the scatter probability is scaled by the density user variable. This value is then divided by 1000 to compensate for the artificial scaling mentioned in the previous section *Host Data Structure*. The algorithm is shown in listing 3.2. Note that $\log(\zeta)$ is used instead of $\log(1-\zeta)$ - this works in this thesis use case,

but may cause problems in other scenarios.

Listing 3.2 Delta Tracking in getNextPosition(cameraRay)

```
do
{
    zeta = random();
    t += -log(zeta) / (majorant * densityScaling / 1000)

    cameraRay.pos = ray.pos + t * ray.dir;
    if( t >= dist )
    {
        return;
    }

    xi = random();
    if(xi < (extinction / majorant))
    {
        return;
    }
}
```

3.3.2 Deep Shadow Map

Since both the cloud data and the directional light are static in the scene, Deep Shadow Maps [LV00] were implemented to avoid recalculating the transmittance at every frame. The shadow map pass occurs only once before any path tracing pass using the RenderTechniqueSV and the *ShadowVolume.comp* shader. The deep shadow map is created as described by the pseudo-code in listing 3.3.

Listing 3.3 Deep Shadow Map Creation

```
1  float accumulatedDistance;
2  float accumulatedDensity;
3  for(uint z = 0; z < voxelAxisCount; z++)
4  {
5      position = calculateVoxelPosition(xyCoord, z);
6      if(isInCloud(position))
7      {
8          accumulatedDensity += sampleCloud(position) * densityScaling /
               1000;
9      }
10     accumulatedDistance += shadowVolumeProperties.voxelSize;
11     Color transmittance = exp( -accumulatedDistance *
           accumulatedDensity / (z + 1));
12     storeTransmittance(transmittance);
13 }
```

3.4 Progressive Photon Mapping Implementation

In the progressive photon mapping approach, one shader per pass is needed:

- *PPM_PT.comp*: the photon tracing shader.
- *PPM_PE.comp*: the radiance estimate shader.

The algorithm combines the probabilistic approach presented by [KZ11] and uses the original volumetric photon mapping from [JC98] as a black box. A 3D grid is used with spatial hashing to store the volumetric photons in a time-efficient manner.

Photon Tracing

The first pass deposits photons throughout the volume using a uniform 3D grid. to keep the number of photon tracing rays constant, Russian roulette is employed to decided between sunlight and ambient rays. Each photon ray power is scaled according to its probability. This pass is similar to the Path Tracing algorithm since both use the random-walk delta tracking to create paths through the cloud. At each vertex of this path, except for the first one, a photon is deposited. The reason behind this is that the direct radiance is better approximated in the second pass [JC98]. The pseudo-code in listing 3.4 represents the tracing algorithm.

Listing 3.4 Photon Tracing

```
float prob = 0.5f;
Ray photonRay;
if(random() < prob)
{
    generateSunRay(photonRay, pdf);
    currentColor = SUNLIGHT_COLOR * parameters.lightIntensity / (pdf *
        emittedPhotons);
    currentColor /= prob;
}
else
{
    generateAmbientRay(photonRay, pdf);
    currentColor = sampleBackground(-photonRay.dir) / (pdf *
        emittedPhotons); // Manual adjustment
    currentColor /= (1.0f - prob);
}

getNextPosition(cameraRay);
while(inVolume(photonRay))
{
    if(!firstScatter)
    {
        depositPhoton(cameraRay);
    }
    getScatterDirection(cameraRay.dir);
    getNextPosition(cameraRay);
}
```

Radiance Estimate

The gathering pass uses ray marching with constant discrete steps. Firstly, both intersection points with the cloud AABB volume are calculated. The marching starts at the furthest point and accumulates radiance moving towards the camera. At each step, the direct lighting from the sunlight and the indirect light from the photon map are sampled. Two approaches were tested for sampling the direct

ambient light, which is described in the section "Direct Lighting". The marching terminates after the ray exits the volume. An abstraction of this process is shown in listing 3.5.

Listing 3.5 Radiance Estimate

```
1  Ray cameraRay = getCameraRay(pixelCoordinates);
2  if(intersectCloud(nearIntersect, farIntersect))
3  {
4      maxDistance = distance(cameraRay.pos + nearIntersect * cameraRay.
           dir, cameraRay.pos + farIntersect * cameraRay.dir);
5      currentDistance = 0;
6      while(currentDistance < maxDistance) //Ray Marching through the
           volume
7      {
8          cameraRay.pos += stepSize * cameraRay.dir;
9
10         directRadiance = SUNLIGHT_POWER * samplePhase(cameraRay.dir,
               sunlightRay.dir) * sampleShadowMap(cameraRay.pos);
11         indirectRadiance = samplePhotonMap(cameraRay.pos);
12
13         transmittance = exp( -stepSize * sampleCloudDensity(cameraRay.
               pos));
14         extinction = sampleCloudDensity(cameraRay.pos) *
               densityScaling / 1000;
15
16         radiance *= transmittance;
17         radiance += sampleCloudDensity(cameraRay.pos) * (
               directRadiance + indirectRadiance) * stepSize;
18     }
19 }
20
21 radiance = (oldRadiance * frameCount + radiance) / (frameCount + 1);
22 storeResult(radiance, pixelCoordinates);
```

3.4.1 Spatial Hashing

Each photon inside the photon map is assigned a grid cell based on its position relative to the origin of the map in a similar manner to [Ped13]. The coordinates are assigned by dividing the relative position by the photon map cell size, and then flooring each component, as shown in Listing 3.6.

Listing 3.6 Spatial Hashing - Cell Assignment

```
1 vec3 normPos = relativePos / photonMapVoxelSize;
2 ivec3 idx3D = ivec3(normPos.x, normPos.y, normPos.z);
3 int idx = idx3D.x + (idx3D.y + idx3D.z * photonMapVoxelCount.y) *
      photonMapVoxelCount.x;
4 idx *= ELEMENTS_PER_CELL;
```

Two techniques were tested for photon storing. The first stored only one photon per cell, randomly chosen by the concurrency in the GPU, and kept track of the collisions using an atomic counter as in [HJ10]. In the radiance estimate, the surviving photon power would be scaled using Russian roulette. The probability of it being in the cell is $p(cell) = 1/n$ with n collisions in the cell. The second approach simply stores as many photons as it can in a cell. The maximum amount of photons per cell in the implementation was set to 32.

Both approaches share the same query process (Listing 3.9), differing only in how the radiance is calculated. Listings 3.8 and 3.7 show the difference mentioned in the previous paragraph.

Listing 3.7 Spatial Hashing - estimateRadiance(int collisionIdx); Many Photons

```
for(int i = 0; i < collisions[collisionIdx]; i++) {
    int idx = collisionIdx * ELEMENTS_PER_CELL + i;
    vec3 distVector = photons[idx].position - queryPos;
    if(dot(distVector, distVector) <= sqrRadius) {
        vec3 photonDir = dirFromPolar(photons[idx].theta, photons[idx].
            phi);
        return samplePhase(rayDir, photonDir) * photons[idx].power;
    }
}
```

Listing 3.8 Spatial Hashing - estimateRadiance(int collisionIdx); Single Photon

```
int idx = collisionIdx * ELEMENTS_PER_CELL + i;
vec3 distVector = photons[idx].position - queryPos;
if(dot(distVector, distVector) <= sqrRadius) {
    vec3 photonDir = dirFromPolar(photons[idx].theta, photons[idx].phi
        );
    return samplePhase(rayDir, photonDir) * photons[idx].power *
        collisions[collisionIdx];
}
```

Listing 3.9 Spatial Hashing - Query

```
int xMin = max((gridPosition.x - estimateRadius) / cellSize, 0);
int xMax = min((gridPosition.x + estimateRadius) / cellSize,
    photonMapVoxelCount.x - 1);

int yMin = max((gridPosition.y - estimateRadius) / cellSize, 0);
int yMax = min((gridPosition.y + estimateRadius) / cellSize,
    photonMapVoxelCount.y - 1);

int zMin = max((gridPosition.z - estimateRadius) / cellSize, 0);
int zMax = min((gridPosition.z + estimateRadius) / cellSize,
    photonMapVoxelCount.z - 1);

for(int z = zMin; z <= zMax; z++){
    for(int y = yMin; y <= yMax; y++){
        for(int x = xMin; x <= xMax; x++){
            int collisionIdx = x + (y + z * photonMapVoxelCount.y) *
                photonMapVoxelCount.x;
            radiance += estimateRadiance(collisionIdx);
        }
    }
}
```

3.4.2 Direct Lighting

Two approaches were tried for the direct ambient lighting: one using the photon map and the other with direct evaluation. The first one offered better performance by storing a photon in the first scatter, while the second one delivers an image closer to the path traced reference by randomly choosing a direction and taking 4 density samples at constant discrete steps. A comparison between the two methods is shown in the section Results.

3.5 Progressive Photon Beams Implementation

This is a proposed implementation for progressive photon beams. It is of great importance since it has the potential to remove the bottleneck from the ambient light radiance estimate in the progressive photon mapping implementation. The content presented here represents the intended final algorithm.

The progressive photon beams is substantially more complex than its previous counterparts. It requires 7 shaders per iteration:

- *PPB_PE.comp*: the photon beam radiance estimate.
- *PPB_PT.comp*: the photon beam tracing.
- *PPB_RadixSort_LocalSort.comp*: the local radix sort for the beam segments.
- *PPB_RadixSort_GlobalSort.comp*: the global radix sort for the beam segments.
- *PPB_RadixSort_PrefixSum.comp*: the prefix sum of the histograms.
- *PPB_GenerateHierarchy.comp*: the LBVH hierarchy tree generation.
- *PPB_CalculateAABB.comp*: the fitting process for the LBVH.

It starts by tracing the photon beams, then sorts the beams using a parallel radix sort, and fits a *bounding volume hierarchy* (BVH) to accelerate the final photon estimate pass.

3.5.1 Beam Tracing

The beam tracing works similarly to photon tracing. The major difference is that the entire beam is stored in the volume, even if it scatters or is absorbed [Jar+11a]. The implementation uses cylinders instead of cones due to simplicity, but integrating ray differentials should not be too difficult. Each beam is split into multiple segments of length equal to the current radius to maintain a unit aspect ratio when fitting the BVH.

3.5.2 Linear Bounding Volume Hierarchy

The BVH technique of choice is the *linear bounding volume hierarchy* instead of SAH as in the original paper from Jarosz et al. [Jar+11a]. The LBVH offers better performance due to its faster construction speed, but only if the number of rays traced remains under 8 million. Since only one sample is taken per pixel per pass,

for a resolution of 800x600 this results in 480 thousand rays and 2 million rays for 1080p per iteration. The LBVH is usually the tree of choice for collision detection in dynamic scenes.

To build an LBVH, first, the objects that will be contained in it need to be assigned a Morton code from a z-curve that goes through the entire scene. Afterward, they need to be sorted linearly according to their Morton code. Lastly, the BVH is built via binary splitting over each digit of the code. To avoid transferring data between the host and the device, the entire algorithm runs on the GPU.

Parallel Radix Sort

The parallel radix sort GPU implementation is based on the publication by Satish et al. [SHG09]. The algorithm works by performing b iterations of 1-bit splits [HSO07] on blocks of size $n * t$ from the total data, where t is the number of threads in a compute shader workgroup and n is the number elements assigned to a thread. After each workgroup finishes sorting, a local histogram of the b sorted digits is calculated and the locally sorted data is moved to global memory. A parallel prefix sum [HSO07] is performed over the histogram table to calculate the global offsets, where each element final index is calculated. The algorithm can be performed multiple times to sort more than b digits.

The process can be resumed in 4 steps:

1. Each workgroup loads a data block into local memory and performs b iterations of the 1-bit split.
2. Each workgroup calculates the 2^b digit histogram and copies local data to the global memory.
3. Calculate a prefix sum of the $p \times 2^b$ histogram table - stored in column-major order.
4. Use the histogram to calculate the offset of each sorted b-digit block and move the elements to their corresponding positions.

Tree Construction

With the elements linearly sorted, Karras presented an algorithm to build LBVHs on the GPUs in [Kar12b]. Given n unique Morton codes, The technique works by finding the covered ranges and split positions for each of the $n - 1$ internal nodes

of the tree in parallel. The image 3.2 illustrates this process. Given an internal node I_i at position i, the algorithm can be resumed to:

1. Find the sibling node with the highest common prefix and set the range search direction d towards it, with $d \in \{-1, 1\}$. In the case of the root node 0, the direction is always $d = 1$.
2. Using a binary search, find a node I_j in the direction d that has a common prefix smaller than the sibling I_{i-d} with the least common prefix. This represents an upper bound for the range covered by I_i. If the search goes out of the range $[0, n-1]$ supported by the nodes, the length of the common prefix is considered -1.
3. Using another binary search, find a node I_l between I_i and I_j (inclusive) that has the highest common prefix that is still smaller than the one with I_{i-d}. This sets the range covered by I_i to $[i, l]$ for $i < l$ or $[l, i]$ if $i \geq l$.
4. The last binary search simply finds the highest common prefix between all the nodes in the range covered by I_i and marks it as the split position.

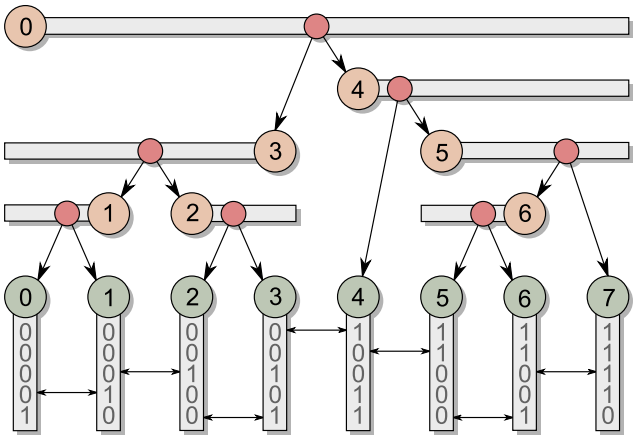


Figure 3.2: Binary Tree Construction. Source [Kar12a]

3.5.3 Radiance Estimate

Once photon beams have been traced and the LBVH is constructed, the radiance estimate can begin. For each camera pixel, a ray is traced into the scene to intersect photons. The estimation uses the Beam Gathering $\times$ Photon Beam with 1D blur from [Jar+11a]. If the ray intersects the beam cylinder, the exact collision point

with the photon beam's plane and the camera ray is calculated and used with the 1D blur formula.

4 Results

This chapter presents a comparison and analysis of the renderings produced by the path tracing and the Progressive photon mapping algorithms. The first section demonstrates a comparison of the two radiance estimate approaches for photon mapping with the reference path tracing images using a moderately complex cloud. The next section gives further examples of the combined radiance estimate approach and shows the differences from the reference images. Thirdly, a problem that was encountered using the single photon per cell is demonstrated. The last section shows a performance comparison between the used methods across the cloud data used.

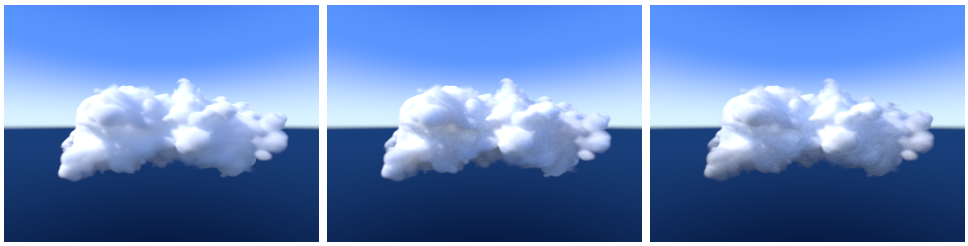
The metric used to compare the images is the *structural similarity index measure* (SSIM). It compares luminance, contrast, and pixel structure of the image to represent how similar two images are perceived by the human eye. A global average is given as well as the local SSIM values for each pixel in the form of a color map. The *root mean squared error* is also given per image to calculate the deviation from the reference image.

4.1 Dense Cloud with Moderate Protrusions

This cloud data was chosen for the implementation testing since it is a middle ground in terms of complexity compared to other clouds in the dataset. The rendering times are 6 minutes for all images presented in this section.

Full Radiance

Figure 4.1 shows a side by side comparison of the photon mapping combined radiance estimate (direct + photon map sampling) with just using the photon map. The first one produces a result closer to the reference. This is further confirmed by figure 4.2.



(a) Path Tracing Reference (b) Combined Estimate (c) Photons Only

Figure 4.1: Full Radiance - Cloud 1

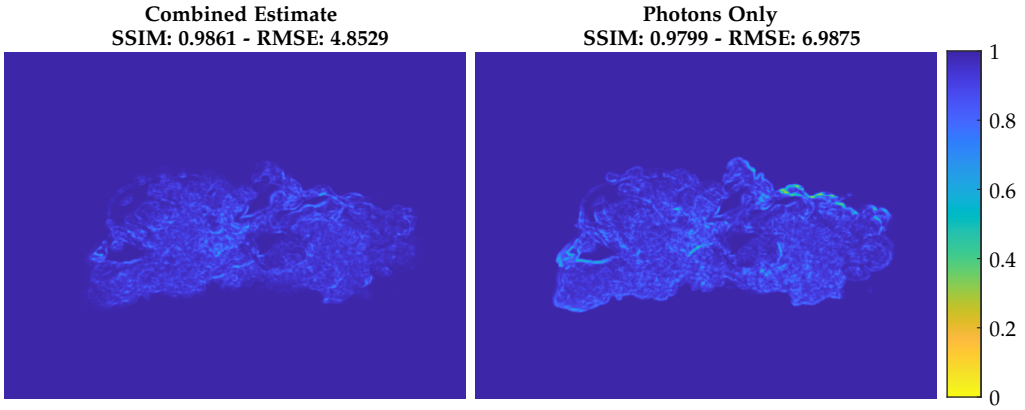
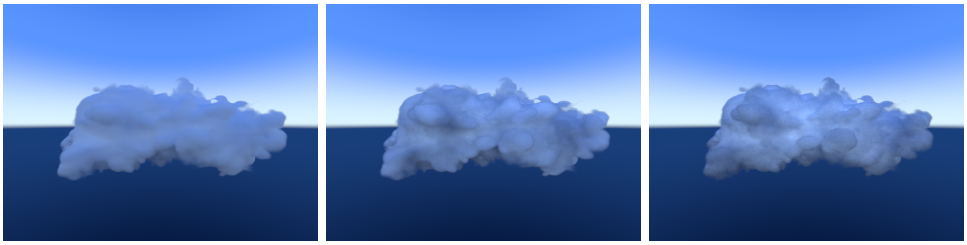


Figure 4.2: Full Radiance SSIM and RSME - Cloud 1

Ambient Light

Simulating the ambient radiance from the sky was a major challenge using photon mapping. As shown by figure 4.3, the borders of the cloud have a much lower variance than the reference, since photons tend to concentrate in the denser part in the center. This also causes the estimate using purely photon mapping to be overly bright in this region. Figure 4.1 illustrates this problem where the cloud edges and center have the greatest disparity in terms of structure similarity from the reference.



(a) Path Tracing Reference (b) Combined Estimate (c) Photons Only

Figure 4.3: Ambient Only - Cloud 1

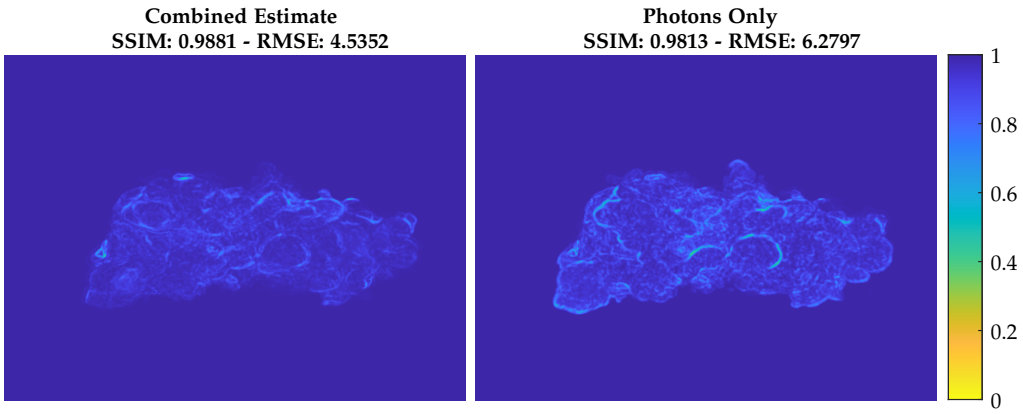
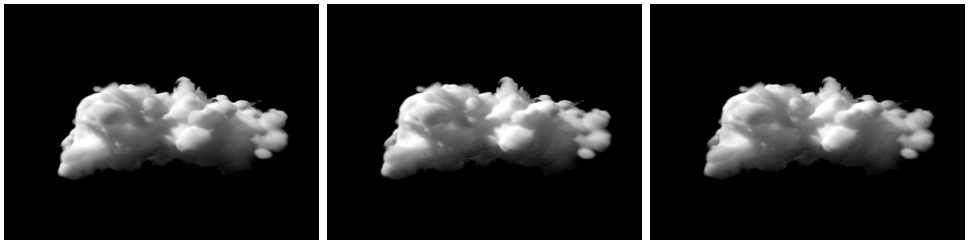


Figure 4.4: Ambient Radiance SSIM and RSME - Cloud 1

Direct Sunlight

Estimating radiance from the directional light modeling the sun illumination was straight forward due to the steady-state of the scene. Even though statistically

the error and structure similarity have values close to the ones from the ambient light comparison, visually the clouds look very similar to the reference. The differences tend to be high only at very specific parts of the cloud. A combined estimate also performs better than using only the photon map in this case. Figures 4.5 and 4.6 illustrate these comparisons.



(a) Path Tracing Reference (b) Combined Estimate (c) Photons Only

Figure 4.5: Sunlight Only - Cloud 1

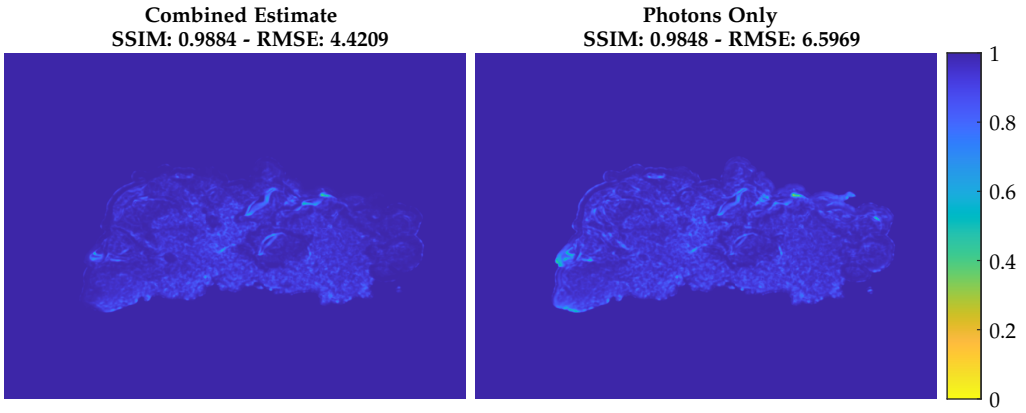
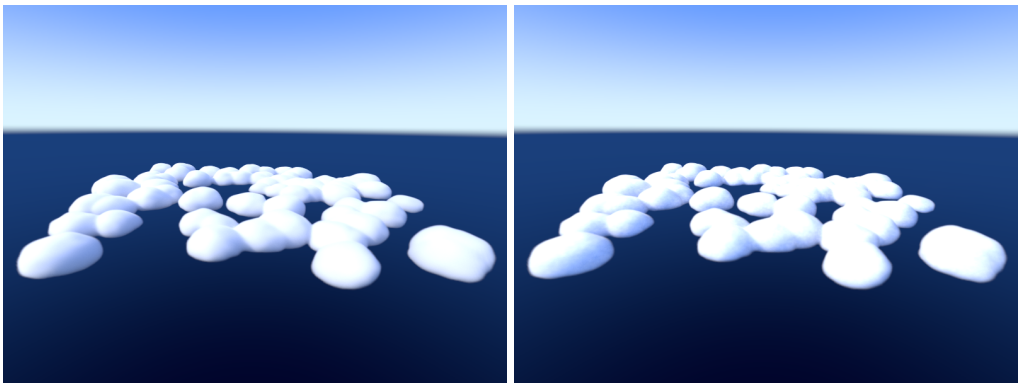


Figure 4.6: Sunlight Radiance SSIM and RSME - Cloud 1

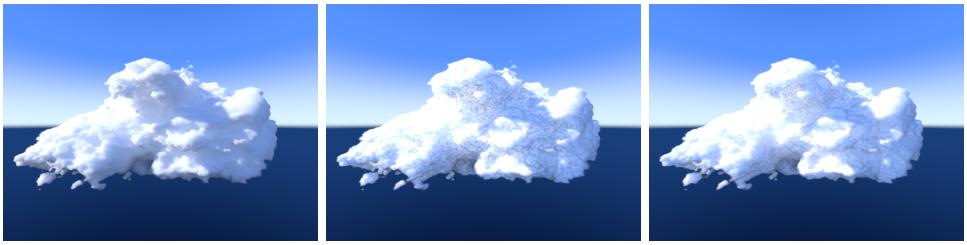
4.2 Simpler and More Complex Clouds

From the previous tests, the combined estimate showed better results. Therefore the next images compare only to this approach. Cloud 2 is composed of many simple small round clouds. This proved to be easily handled by the photon mapping, as seen in figure 4.7 (Rendering times: 60s). Cloud 3 on the other hand is far more complex, and the results did produce a lot of noise in some parts of the cloud. Figure 4.8 (Rendering times from left to right: 60s, 60s, and 360s) shows the progress and the artifacts. A statistical comparison is shown in figure 4.9.



(a) Path Tracing Reference (b) Combined Estimate

Figure 4.7: Full Radiance - Cloud 2



(a) Path Tracing Reference (b) Combined Estimate (c) Combined Estimate

Figure 4.8: Full Radiance - Cloud 3

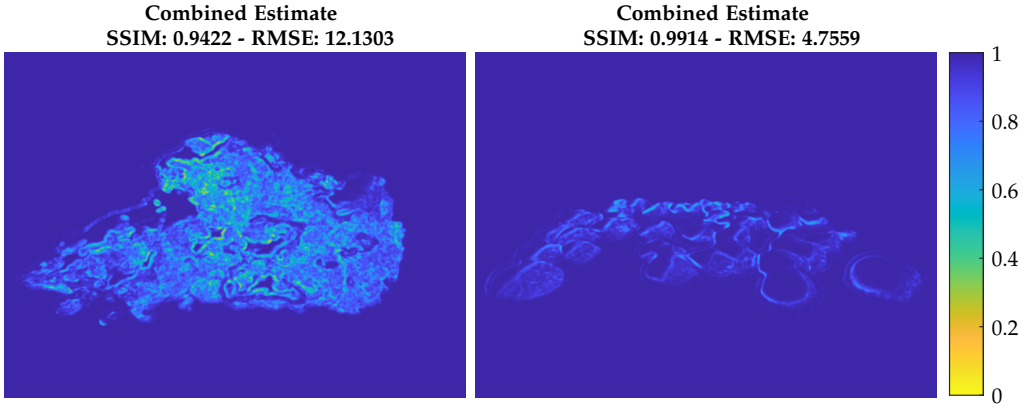
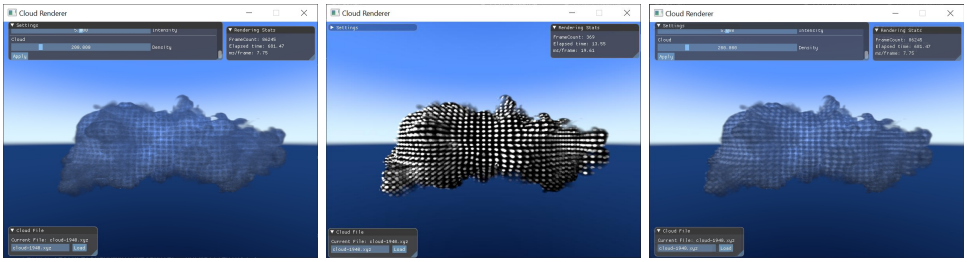


Figure 4.9: Full Radiance SSIM and RSME - Clouds 3 (left) and 2 (right)

4.3 One Photon vs. Many Photons per Grid Cell

A grid artifact was produced through the cloud when using a single photon per cell and scaling its power via Russian roulette. Even after extensive debugging the source of this was not found. Figure 4.10 shows an overlay of the grid artifact and the cell centers rendered as spheres. The artifacts appear to be aligned with cell centers. This grid artifact is not present when using multiple photons per cell.



(a) Grid Artifacts (b) Cell Centers as Spheres (c) Overlay of Both Images

Figure 4.10: Single Photon Grid Artifact

4.4 Performance

Comparing the performance of this works' progressive photon mapping implementation with the traditional path tracing implementation, the results were in favor of the second in terms of rendering time per frame. Both convergence times and framerate were better in path tracing. Table 4.1 shows the average framerate in ms/frame. C stands for *Combined Estimate* approach and P for *Photon Map Only* approach.

	PT	PPM- C	PPM-P	PPM-C Sunlight
Cloud 1940	12.6	174	26.3	31.8
Cloud 3	29.5	262	-	-
Cloud 2	3.65	31	-	-

Table 4.1: Performance Comparison

The major impact in performance for the combined estimate is the ambient term which requires ray marching to approximate the transmittance in an arbitrary direction. Secondly, the ray marching of the radiance estimate from the volumetric photon mapping technique is also relatively slower than the delta-tracking from path tracing. Sampling the deep shadow map introduces some marginal performance impact.

5 Conclusion

This thesis presented two functional implementations techniques to render clouds using a Vulkan engine with compute shaders. Insights on how to implement a third technique were also provided.

A path tracing algorithm was used as a reference to produce high quality physically based renderings in its basic backward tracing form. Additionally, a progressive photon mapping approach was implemented using the traditional volumetric photon mapping technique as a black box. This technique was applied with two variants: one using solely the photon map to estimate radiance, and the other one simulating the direct lighting at sample points explicitly. A single photon per grid cell was also tested but it produced grid-like artifacts in the final image. Lastly, a progressive photon beams implementation is proposed as an improvement to progressive photon mapping for the cloud rendering use case.

This work also presented a comparison between the techniques through rendered images and performance tables. The second implementation did not produce results that converge faster than the path tracing technique, and it does not have interactive framerates if an ambient term is simulated. The images produced by the progressive photon mapping implementation contain artifacts w.r.t. the reference for complex cloud formations and remain noisy after many progressive iterations.

In conclusion, the proposed implementation of progressive volumetric photon mapping did not perform better than the path tracing technique in interactivity terms. As stated by Knaus and Zwicker [KZ11], probabilistic progressive photon mapping uses other photon mapping approaches as a black box. Therefore, there is the possibility that another implementation or technique, such as photon beams, may perform better in this situation.

5.1 Outlook

There is a lot of room for improvement in the renderer presented. A couple the most significant are listed here:

Progressive Photon Beams

Even though this technique was not implemented, progressive photon beams [Jar+11b] should provide a substantial improvement over the progressive photon mapping with volumetric photon mapping as a black box. It eliminates the ray marching in the radiance estimate and replaces it by a ray-cone intersection. The additional BVH building time should still be marginal due to the low number of photon beams per pass, as seen in [Kar12b]. It also should solve the problem of estimating transmittance for the ambient term, since the beams originally use a deep shadow map to estimate the transmittance in heterogeneous media. Lastly, using ray differentials also adaptive sampling w.r.t. the photon density in certain regions of the cloud.

Beam Gathering

Using spheres to explicitly intersect the photons like in [JZJ08] should also be faster than trying to gather them at discrete steps. This approach may benefit from a BVH as well. One negative aspect of this approach would be that to have adaptive sampling an additional pass would be required to change the radius of each photon sphere regarding the local density.

Ambient Term Transmittance Caching

An approach similar to the one used by Jarosz in [Jar+08] to create a transmittance approximation field over the cloud could also improve the ambient term estimate in static scenes since deep shadow maps are not viable.

Other rendering frameworks

While the Vulkan API offers flexibility in terms of GPU programming, it is still relatively new compared to other APIs. This caused time to be spent on debugging instead of the research goal. Other more mature rendering frameworks may be

a better choice for doing academic research, which can provide better insight on results and facilitate solving programming related errors.

List of Figures

List of Tables

Listings

Bibliography

[Adv20] Advanced Micro Devices, Inc. *AMD RDNA 2 Architecture*. 2020. URL: https://www.amd.com/en/technologies/rdna-2 (visited on 06/23/2020).

[BJ17] B. Bitterli and W. Jarosz. "Beyond Points and Beams: Higher-Dimensional Photon Samples for Volumetric Light Transport." In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 36.4 (July 2017). DOI: 10/gfznbr.

[Cor20] O. Cornut. *Dear ImGui: Bloat-Free Immediate Mode Graphical User Interface*. 2020. URL: https://github.com/ocornut/imgui (visited on 06/23/2020).

[Den+19] X. Deng, S. Jiao, B. Bitterli, and W. Jarosz. "Photon surfaces for robust, unbiased volumetric density estimation." In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10/gf6rx9.

[Eck87] R. Eckhardt. "Stan Ulam, John von Neumann, and the Monte Carlo method." In: *Los Alamos Science* 15 (1987), pp. 131–137.

[Fon+17] J. Fong, M. Wrenninge, C. Kulla, and R. Habel. "Production volume rendering: SIGGRAPH 2017 course." In: *ACM SIGGRAPH 2017 Courses*. 2017, pp. 1–79.

[GLF20] GLFW. *GLFW - An OpenGL Library*. 2020. URL: https://www.glfw.org/ (visited on 06/23/2020).

[Gor+84] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. "Modeling the Interaction of Light between Diffuse Surfaces." In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 213–222. ISSN: 0097-8930. DOI: 10.1145/964965.808601.

[Hac+17] T. Hachisuka, I. Georgiev, W. Jarosz, J. Křivánek, and D. Nowrouzezahrai. "Extended Path Integral Formulation for Volumetric Transport." In: *Eurographics Symposium on Rendering - Experimental Ideas and Implementations*. Ed. by M. Zwicker and P. Sander. The Eurographics Association, 2017. ISBN: 978-3-03868-045-1. DOI: 10.2312/sre.20171195.

[HJ09] T. Hachisuka and H. W. Jensen. "Stochastic progressive photon mapping." In: *ACM SIGGRAPH Asia 2009 papers*. 2009, pp. 1–8.

[HJ10] T. Hachisuka and H. W. Jensen. "Parallel progressive photon mapping on GPUs." In: *ACM SIGGRAPH ASIA 2010 Sketches*. 2010, pp. 1–1.

[HOJ08] T. Hachisuka, S. Ogaki, and H. W. Jensen. "Progressive photon mapping." In: *ACM SIGGRAPH Asia 2008 papers*. 2008, pp. 1–8.

[HPJ12] T. Hachisuka, J. Pantaleoni, and H. W. Jensen. "A path space extension for robust light transport simulation." In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), pp. 1–10.

[HSO07] M. Harris, S. Sengupta, and J. D. Owens. *Parallel Prefix Sum (Scan) with CUDA*. 2007. URL: https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda (visited on 06/23/2020).

[Jar+08] W. Jarosz, C. Donner, M. Zwicker, and H. W. Jensen. "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.

[Jar+11a] W. Jarosz, D. Nowrouzezahrai, I. Sadeghi, and H. W. Jensen. "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.

[Jar+11b] W. Jarosz, D. Nowrouzezahrai, R. Thomas, P.-P. Sloan, and M. Zwicker. "Progressive Photon Beams." In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 30.6 (Dec. 2011). DOI: 10/fn5xzj.

[JC98] H. W. Jensen and P. 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*. 1998, pp. 311–320.

[Jen96] H. W. Jensen. "Global Illumination Using Photon Maps." In: *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Porto, Portugal: Springer-Verlag, 1996, pp. 21–30. ISBN: 3211828834.

[JZJ08] W. Jarosz, M. Zwicker, and H. W. 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.

[Kaj86] J. T. Kajiya. "The rendering equation." In: *Computer Graphics*. 1986, pp. 143–150.

[Kar12a] T. Karras. "Maximizing parallelism in the construction of BVHs, octrees, and k-d trees." In: *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. 2012, pp. 33–37.

[Kar12b] T. Karras. *Thinking Parallel, Part III: Tree Construction on the GPU*. 2012. URL: https://developer.nvidia.com/blog/thinking-parallel-part-iii-tree-construction-gpu/ (visited on 06/23/2020).

[Kar20] B. Karlsson. *RenderDoc*. 2020. URL: https://renderdoc.org/ (visited on 06/23/2020).

[Kel97] A. Keller. "Instant radiosity." In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 49–56.

[Kel98] A. Keller. "The Quasi-Random Walk." In: *Monte Carlo and Quasi-Monte Carlo Methods 1996*. Ed. by H. Niederreiter, P. Hellekalek, G. Larcher, and P. Zinterhof. New York, NY: Springer New York, 1998, pp. 277–291. ISBN: 978-1-4612-1690-2.

[Khr20a] Khronos Group. *Core Language (GLSL)*. Khronos Group Inc. 2020. URL: https://www.khronos.org/opengl/wiki/Core_Language_(GLSL) (visited on 06/23/2020).

[Khr20b] Khronos Group. *Vulkan Overview*. Khronos Group Inc. 2020. URL: https://www.khronos.org/vulkan/ (visited on 06/23/2020).

[Koc20] D. Koch. *Ray Tracing In Vulkan*. Khronos Group Inc. 2020. URL: https://www.khronos.org/blog/ray-tracing-in-vulkan (visited on 06/23/2020).

[Kři+14] J. Křivánek, I. Georgiev, T. Hachisuka, P. Vévoda, M. Šik, D. Nowrouzezahrai, and W. Jarosz. "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.

[KZ11] C. Knaus and M. Zwicker. "Progressive photon mapping: A probabilistic approach." In: *ACM Transactions on Graphics (TOG)* 30.3 (2011), pp. 1–13.

[Lun20] LunarG, Inc. *Vulkan ©SDK*. LunarG, Inc. 2020. URL: https://www.lunarg.com/vulkan-sdk/ (visited on 06/23/2020).

[LV00] T. Lokovic and E. Veach. "Deep shadow maps." In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 385–392.

[LW93] E. P. Lafortune and Y. 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.

[LW96] E. P. Lafortune and Y. D. Willems. "Rendering participating media with bidirectional path tracing." In: *Rendering techniques' 96*. Springer, 1996, pp. 91–100.

[Mar+18] J. Marco, A. Jarabo, W. Jarosz, and D. Gutierrez. "Second-order occlusion-aware volumetric radiance caching." In: *ACM Transactions on Graphics (Presented at SIGGRAPH)* 37.2 (Apr. 2018). DOI: 10/gdv86k.

[MB90] J. D. MacDonald and K. S. Booth. "Heuristics for ray tracing using space subdivision." In: *The Visual Computer* 6.3 (1990), pp. 153–166.

[Met+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. "Equation of state calculations by fast computing machines." In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[Mic20a] Microsoft Corporation. *DirectX Raytracing (DXR) Functional Spec*. 2020. URL: https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html (visited on 06/23/2020).

[Mic20b] Microsoft Corporation. *Visual Studio IDE*. 2020. URL: https://visualstudio.microsoft.com/ (visited on 06/23/2020).

[Nov+12a] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz. "Progressive Virtual Beam Lights." In: *Computer Graphics Forum (Proceedings of EGSR)* 31.4 (June 2012). DOI: 10/gfzndw.

[Nov+12b] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz. "Virtual Ray Lights for Rendering Scenes with Participating Media." In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 31.4 (July 2012). DOI: 10/gbbwk2.

[NSJ14] J. Novák, A. Selle, and W. Jarosz. "Residual Ratio Tracking for Estimating Attenuation in Participating Media." In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 33.6 (Nov. 2014). DOI: 10/f6r2nq.

[NVi20] NVidia Corporation. *NVidia RTX Ray Tracing*. 2020. URL: `https://developer.nvidia.com/rtx/raytracing` (visited on 06/23/2020).

[Ped13] S. A. Pedersen. "Progressive photon mapping on gpus." MA thesis. Institutt for datateknikk og informasjonsvitenskap, 2013.

[PH89] K. Perlin and E. M. Hoffert. "Hypertexture." In: *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. 1989, pp. 253–262.

[PJH17] M. Pharr, W. Jakob, and G. Humphreys. *Physically based rendering: from theory to implementation*. Morgan Kaufmann Publishers/Elsevier, 2017.

[PKK00] M. Pauly, T. Kollig, and A. Keller. "Metropolis light transport for participating media." In: *Rendering Techniques 2000*. Springer, 2000, pp. 11–22.

[SHG09] N. Satish, M. Harris, and M. Garland. "Designing efficient sorting algorithms for manycore GPUs." In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. 2009, pp. 1–10.

[Sil86] B. W. Silverman. *Density estimation for statistics and data analysis*. Vol. 26. CRC press, 1986.

[SSF08] L. Schjøth, J. Sporring, and O. Fogh Olsen. "Diffusion based photon mapping." In: *Computer Graphics Forum*. Vol. 27. 8. Wiley Online Library. 2008, pp. 2114–2127.

[Vea97] E. Veach. "Robust Monte Carlo methods for light transport simulation." PhD thesis. Stanford University, 1997.

[VG97] E. Veach and L. J. Guibas. "Metropolis light transport." In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 65–76.

[WH92] G. J. Ward and P. S. Heckbert. *Irradiance gradients*. Tech. rep. Lawrence Berkeley Lab., CA (United States); Ecole Polytechnique Federale, 1992.

[Wil78] L. Williams. "Casting curved shadows on curved surfaces." In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. 1978, pp. 270–274.

[WRC88] G. J. Ward, F. M. Rubinstein, and R. D. Clear. "A ray tracing solution for diffuse interreflection." In: *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. 1988, pp. 85–92.