FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Implementing Virtual Ray Lights

for Rendering Scenes with Participating Media

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Michael Oppitz
Matrikelnummer 01227129

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Károly Zsolnai-Fehér, B.Sc. M.Sc.

Wien, 7. August 2018

Michael Oppitz	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Implementing Virtual Ray Lights

for Rendering Scenes with Participating Media

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Michael Oppitz
Registration Number 01227129

to the Faculty of Informatics
at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Károly Zsolnai-Fehér, B.Sc. M.Sc.

Vienna, 7th August, 2018

Michael Oppitz	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Michael Oppitz
Blindengasse 5/25, 1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. August 2018

Michael Oppitz

Acknowledgements

First and foremost, I want to thank my advisor Károly Zsolnai-Fehér for his unwavering support and guidance through the work on this thesis. For all challenges that arose, he always trusted in me to solve them. He gave me the freedom and time to achieve something that I can be proud of.

Furthermore, I want to thank my family and friends. Especially through the steady support, motivation and trust of my parents this work was made possible.

Danksagung

Zu allererst möchte ich meinem Betreuer Károly Zsolnai-Fehér für seine Unterstützung und seinen Rat bei der Erstellung dieser Arbeit danken. Bei allen Herausforderungen vermittelte er mir immer die Zuversicht, diese bewältigen zu können. Er gab mir die Freiheit und die nötige Zeit, etwas zu schaffen, auf das ich stolz sein kann.

Weiters möchte ich meinen Eltern und Freunden danken. Speziell durch die durchgehende Unterstützung, die Motivation und das Vertrauen meiner Eltern wurde diese Arbeit möglich gemacht.

Abstract

This thesis documents the full implementation of the method Virtual Ray Lights for Rendering Scenes with Participating Media. As a basic understanding of the foundations of rendering and related approaches is necessary to understand this complex method, these foundations are discussed first. There, the rendering equation and the physical behaviour of light is described. Additionally, rendering approaches like Recursive Ray Tracing and Photon Mapping that do not consider participating media, as well as methods like Volumetric Photon Mapping, Virtual Point Lights and Photon Beams, which are able to render participating media, are evaluated.

For the discussion on Virtual Ray Lights, the evaluation takes place in three parts. First, the method is discussed in general with a mathematical analysis. Afterwards, implementation details are evaluated where pseudocode examples are provided. Lastly, the rendered results of the implementation are evaluated thoroughly. These results are also compared to provided images from various research papers.

The goal of this thesis is to provide an implementation of Virtual Ray Lights, as well as providing the tools to implement this method in other projects. We provide the well-documented source code for this project, with the scene settings to recreate the examples.

Kurzfassung

Diese Arbeit dokumentiert eine komplette Implementierung der Methode Virtual Ray Lights for Rendering Scenes with Participating Media. Um ein Grundverständnis für einige notwendige Prinzipien der Bildsynthese (Rendering) zu schaffen, werden zu allererst die wichtigsten Bereiche beschrieben, die für komplexere Methoden notwendig sind. In diesem Schritt werden auch die bekannte Rendering-Gleichung und einige physikalische Eigenschaften von Licht besprochen. Außerdem werden einige Methoden wie Recursive Ray Tracing und Photon Mapping untersucht, welche Partikelwolken noch nicht beachten. Im folgenden Schritt werden dann Methoden wie Volumetric Photon Mapping, Virtual Point Lights und Photon Beams diskutiert, welche in der Lage sind, Partikelwolken in die Berechnungen einzubeziehen.

Die Diskussion von Virtual Ray Lights wird in drei Teile aufgeteilt. Im ersten Abschnitt wird diese Methode allgemein besprochen, gemeinsam mit einer mathematischen Analyse. Anschließend werden Details zur Implementierung diskutiert sowie einige Codebeispiele aufbereitet. Im dritten Teil werden die Ergebnisse dieses Verfahrens analysiert und mit den Ergebnissen aus anderen wissenschaftlichen Arbeiten verglichen.

Das Ziel dieser Arbeit ist es eine Implementierung von Virtual Ray Lights zur Verfügung zu stellen sowie es zu ermöglichen, diese Methode in andere Projekte zu implementieren. Der gut dokumentierte Quellcode dieses Projekts wird gemeinsam mit den Beispielszenen zur Verfügung gestellt.

Contents

CHAPTER 1

Introduction

The field of rendering in computer graphics is, as computer science in general, a relatively new field that developed in the last decades. In these years, many important approaches were developed to create progressively better representations of real-world phenomena. The focus of this thesis is to evaluate and document the implementation of one of these methods, Virtual Ray Lights (VRL) for Rendering Scenes with Participating Media [NNDJ12b]. As this method is closely related to Photon Mapping, the first chapters of the thesis discuss foundations and similar methods, up to the explanation of Photon Mapping itself. The second part focuses on rendering methods for participating media, with a detailed discussion and evaluation of Virtual Ray Lights.

As a preparation of the evaluation of this method and the corresponding documentation of the implementation, a few principles must be discussed that are necessary for understanding the concept of this method and similar methods. In Chapter 2 the foundations for rendering, like the rendering equation, global illumination and the physical behaviour of light are discussed. Within this chapter, properties like the bidirectional reflectance distribution function, reflections and refractions are explained. These details are necessary to create methods that simulate the physical properties of the real world.

The discussed properties can then already be implemented into a well-known rendering method, Recursive Ray Tracing. The implementation of this approach is documented in Chapter 3. This chapter is of importance, as many complex methods use a similar framework, and it is therefore a good idea to separate the general functionality of most rendering methods from complex, implementation-specific functionality that is specific for each approach. The documentation in this chapter contains explanations for the construction of a camera ray, tracing that ray through the scene and the interaction with a few simplified materials. The figures in this chapter will show the first rendered results. They can already provide an understanding on how hard it is to create realistic results, as these results are not close to convincing simulations of the real world.

The next chapter presents an important rendering method that not only improves the results immensely, but which was also expanded through many approaches that are discussed later on. This method is called Photon Mapping (PM), and as Virtual Ray Lights is one of the methods that expand Photon Mapping, the thorough evaluation of this method is necessary and is provided in Chapter 4. This two-pass approach is evaluated with a mathematical analysis, as well as code examples and figures. The shortcomings are discussed and approaches to get rid of them are discussed in Section 4.2 and 4.3.

Up to this point, to simplify the explanations and examples, participating media are not considered. Therefore, a thorough evaluation of different properties of participating media, as well as a mathematical analysis of these properties is provided in Chapter 5, to explain the knowledge necessary for the discussion on related rendering methods. Many of the equations of this chapter are needed in the implementation details of Virtual Ray Lights later. For readers that do not have experience with these topics, it is therefore advisable to make oneself familiar with the mentioned terms in this chapter. The last section of this chapter (Section 5.6) explains how the tracing of photons can be expanded to participating media, as the approaches that are explained in the following chapters rely on it.

As the foundation for the rendering of participating media was built in Chapter 5, Chapter 6 discusses different techniques that can render participating media. This chapter is the related work to Virtual Ray Lights, as it describes many methods that expand Photon Mapping into participating media. There are two types of algorithms for this task. The first type are algorithms that trace photons through the scene and evaluate them. Some examples of these algorithms are Volumetric Photon Mapping [JCKS02], the Beam Radiance Estimate [JZJ08] and Photon Beams [JNSJ11]. The second type create virtual lights at the position of the traced photons. Virtual Point Lights [Kel97] and Virtual Beam Lights [NNDJ12a] belong to this type of algorithms and are discussed in this chapter along other methods. This second type is especially important, as Virtual Ray Lights belongs to this type as it expands Virtual Point Lights.

The following chapter is solely focused on the explanation of Virtual Ray Lights. Chapter 7 therefore explains the methodology of this method first, followed by a detailed discussion of the equations for volume-to-surface and volume-to-volume radiance. Furthermore, the importance sampling techniques that are used and developed for this method are provided with much mathematical depth. Additionally, Appendix A provides a simple introduction to importance sampling, as the techniques that were developed for Virtual Ray Lights are quite advanced. Readers that are not experienced in the topic of importance sampling are therefore advised to read this appendix first to be able to evaluate these complex importance sampling functions of this chapter properly.

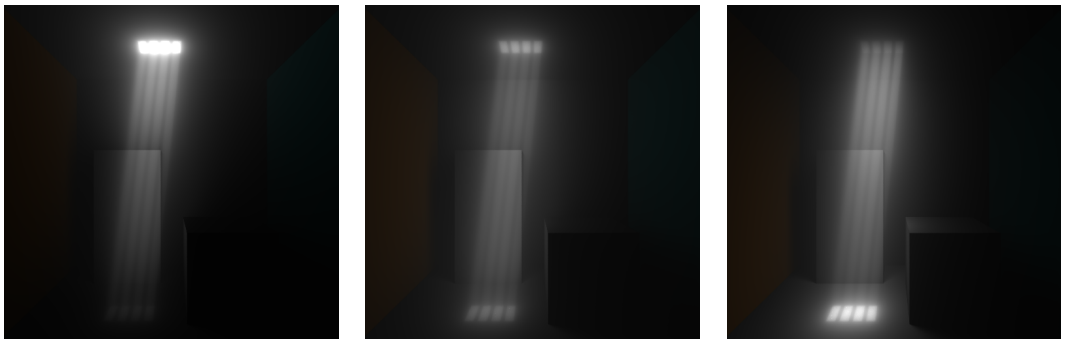
The documentation of the implementation of Virtual Ray Lights into an educational path tracer [ZF18b] is provided in Chapter 8. Although the examples are only provided with pseudocode, the functions and custom data structures are explained thoroughly in this chapter. It is therefore possible to implemented Virtual Ray Lights by using this chapter as a guide. Some simplifications that are made in the pseudocode examples in this chapter can be reread in previous chapters, which is mentioned on a case-by-case basis. Especially the functions for importance sampling are kept rather similar to the original implementation, as they provide the most difficulty. To understand this chapter properly, it is necessary to have read the previous chapter, as many of the explanations and naming is based primarily on the explanations of that chapter.

In Chapter 9, the resulting images that were created with the implementation of this method are compared and analysed. There are different example scenes for which many tests are provided. Theses scenes entail point lights in infinite media, as well as Cornell box scenes. An example for a Cornell box scene that was rendered with this implementation can be seen in Figure 1.1. This small example shows the convergence of the scene with isotropic media (middle) and anisotropic media (left and right). For all scenes, tests with a low sample count are provided to compare the early convergence. Furthermore, examples that were created in equal time are provided as well as fully converged results. Not for all examples fully converged results can be provided, as some methods are highly unlikely to converge. For the statistical evaluation of the convergence, the root-mean-square error and the peak signal-to-noise ratio are provided for the evaluation. For some specific scenes, further specialized tests and examples are provided. Additional to all these tests, recreations of figures that are provided in Virtual Ray Lights [NNDJ12b] and Joint Importance Sampling [GKH$^+$13] are provided to show comparisons of the results. Although the recreation cannot be exactly equal, as some scene properties are not known, it is the goal to show similar convergence of the same sample count as well as similar properties that were highlighted for these figures.

The last chapter (Chapter 10) will provide a conclusion of this thesis. The main focus will be on the implementation of Virtual Ray Lights and the results that are provided.

In summary, this thesis provides:

- a thorough introduction to light transport and global illumination,
- detailed descriptions and analysis of multiple fundamental rendering techniques,
- a full implementation of Virtual Ray Lights with a detailed mathematical derivation (Equations 7.21 - 7.26) of the advanced joint distribution sampling, which was absent from the original paper,
- an analysis of the results of our implementation,
- the source code for the implementation in two versions (a command line program and an integration into an educational path tracing framework).



Anisotropic (H-G, g = -0.45) Isotropic (H-G, g = 0.45) Anisotropic (H-G, g = +0.45)

Figure 1.1: A Cornell box scene with three types of participating media. On the left, backwards scattering media (H-G, g = -0.45), in the middle, isotropic media and on the right, forward scattering media (H-G, g = 0.45). All scene are rendered with the implementation of VRL.

CHAPTER 2

Light Transport

A good basic knowledge of the key concepts of light transport is necessary to understand the complex approaches that follow. In this chapter, the mathematical foundations that are crucial for the realistic representation of light transport are explained in Section 2.1. The results that are the outcome of implementing a realistic lighting model are compared to a simpler model, that does not include all these complex calculations (Section 2.2). Furthermore, the behaviour of light when interacting with a surface is discussed in different cases in Section 2.3. This leads to a basic understanding of the necessary theory. The approaches, that are explained in the following chapters, build on the theorems and equations that are discussed in this chapter. It is therefore vital to understand the basics of rendering and slowly build a foundation to understand complex theories. Most of the knowledge for the following theory is provided in multiple excellent lectures [ZF18a, Mar12], books [PH10] and theses [Jar08, Nov14].

2.1 The Rendering Equation

There are different ways to calculate the transport of light in a scene. The key concept, for approaches that want to simulate the realistic behaviour of light, is the Rendering Equation [Kaj86, ICG86]. In comparison to other representation of the behaviour of light, the rendering equation is physically correct. The result is called Global Illumination [Dor95], which is a key in creating the realistic looks of renderings.

The rendering equation was introduced by James Kajiya [Kaj86] and David Immel et al. [ICG86] in 1986 and is still, beyond a doubt, one of the most important equation in computer graphics. The rendering equation can be written as

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega L_i(x,\vec{\omega}')f_r(x,\vec{\omega},\vec{\omega}')\,\cos\theta\,d\vec{\omega}', \tag{2.1}$$

with all terms of this equation being explained in Table 2.1.

Symbol	Description
$L_o(x, \vec{\omega})$	the exiting radiance at point x in the direction $\vec{\omega}$
$L_e(x, \vec{\omega})$	the emitted radiance at point x in the direction $\vec{\omega}$
$\int_\Omega ... d\vec{\omega}$	the sum of the incoming radiance from all directions $\vec{\omega}'$
$L_i(x, \vec{\omega}')$	the incoming radiance at the point x from the direction $\vec{\omega}'$
$f_r(x, \vec{\omega}, \vec{\omega}')$	the BRDF

Table 2.1: The terms of the rendering equation.

Unfortunately, the integral $\int_\Omega ... d\vec{\omega}$ is not solvable in a closed form. The exiting radiance $L_o(x, \vec{\omega})$ at a point x in the direction $\vec{\omega}$ depends on the incoming radiance $L_i(x, \vec{\omega}')$ of all other directions $\vec{\omega}'$ to this point x. The incoming radiance $L_i(x, \vec{\omega}')$ also depends on x, which therefore makes it not solvable.

This leads to the interesting fact, that approximations of this equations must be made to receive results. There are many methods that approximate the rendering equation, with each of them having different advantages and disadvantages. Some of the most important approaches are Recursive Ray Tracing [Gla89, Whi79], Path Tracing [LW93], Metropolis Light Transport [VG97] and Photon Mapping [Jen01]. An introduction into Ray Tracing is given in Chapter 3. Although this version of Ray Tracing does not approximate global illumination, it is important to understand the concept, as it is used in a majority of all other approaches. Photon Mapping is explained in detail in Chapter 4, as it provided the foundation of multiple algorithms that are discussed in the following chapters.

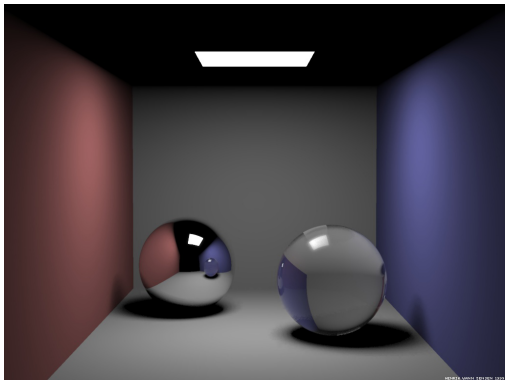
An important group of algorithms tries to solve this problem with Monte Carlo integration. This process approximates an integral with an estimator, the Monte Carlo estimator. The approximation convergence to the actual solution by evaluating the estimator with many samples. This Monte Carlo estimator of the integral $F(x)$ of a function $f(x)$ can be denoted as

$$F(x) \approx \frac{1}{N} \sum_{N}^{i=1} \frac{f(x)}{\mathrm{pdf}(x)}, \tag{2.2}$$

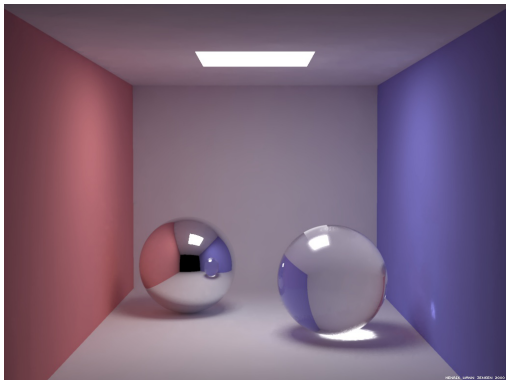
where N is the number of samples, $f(x)$ the result of the function with the current sample x and $\mathrm{pdf}(x)$ the probability of choosing this sample. A complete introduction in Monte Carlo sampling and importance sampling is provided in Appendix A. Along many others, Virtual Ray Lights is a method that relies on this technique for convergence and is therefore called a Monte Carlo method.

2.2 Global Illumination

To show the improvements, that global illumination brings to the overall realistic appearance of a rendered scene, an image that was created by only using the Recursive Ray Tracing algorithm from Chapter 3 is compared to a scene, that used the Photon Mapping approach from Chapter 4. In Figure 2.1 a scene that was raytraced is shown



Recursive Whitted-Style Ray Tracing



Photon Mapping

Figure 2.1: A comparison of a scene that was rendered with Recursive Whitted-Style Ray Tracing [Whi79] to a scene that was rendered with Photon Mapping. Source: [JCKS02]

in the left image and a scene that was rendered using Photon Mapping is shown on the right. There are many differences in these images that lead to a much more realistic appearance of the image to the right.

The reason for this realistic appearance is, that light bounces around in a scene, which can be simulated with Photon Mapping. How light bounces in a scene, and the differences in behaviour, that are caused by different materials, are explained in Section 2.3.

Just from a visual comparison, one difference that is immediately visible is the bright light disturbance on the floor and on the wall. This phenomenon is called caustics. Caustics are created, when many light rays are reflected or refracted in to a specific region. They can often be seen when light travels through glass or water. The mathematical theory behind reflections and refractions can be seen in the following sections.

Another discrepancy between these images is that a color transfer from areas that are close to each other can be seen in the image to the right. Especially on the floor and the ceiling, the colors from the adjacent walls can be seen. This phenomenon is also the reason, why the image to the right is overall brighter than the image to the left. There are multiple names for these appearances like radiosity, color bleeding and indirect illumination. In the next chapters, this effect is only called indirect illumination, to prevent misunderstandings.

2.3 Light Behaviour

As mentioned in the previous sections, the behaviour of light, when interaction with different kinds of materials and media is an important factor, when it comes to the realistic representation of objects in renderings. As discussed in Section 2.1, illumination is not just the direct contribution of light on a point, but also the incoming radiance from other points. An easy example would be a room on a sunny day. Even though the sun does not illuminate the whole room directly, the room is still bright. In the real world there is rarely ever a place that is completely dark. This is contributed to the fact that light bounces after its first intersection, which produces indirect illumination and caustics. The behaviour of light, when interacting with a surface is discussed in this section, but light can also be affected by participating media, which is discussed in Chapter 5.

The first thing to discuss is in what direction light can be reflected in a scene. In Figure 2.2 a comparison between basic materials is shown. To the left the behaviour of light on a completely diffuse surface is shown. In this case light scatters in all direction of a hemisphere that is oriented to the surface normal of the interaction point. The case in the middle shows a completely specular surface. This material is a perfect mirror and reflects light in a direction that can easily be specified. How this direction can be calculated can be seen in Section 2.3.2. The last case is a combination of the methods mentioned above, which is much more realistic, as in nature a perfectly diffuse surface only exists in theory.

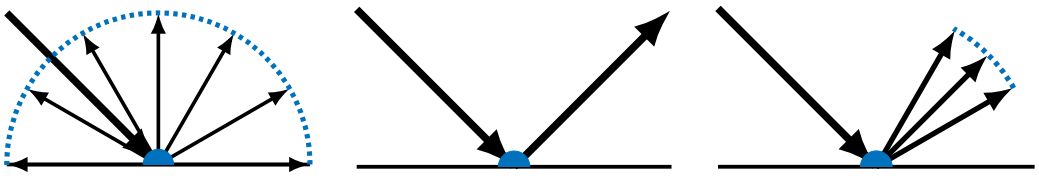


Figure 2.2: The different distributions of light rays when interacting with various surface types.

2.3.1 Bidirectional Reflectance Distribution Function

Different materials have different properties. These properties can be defines with a bidirectional reflectance distribution function (BRDF) [Nic65]. The mathematical notation of the BRDF is

$$f_r(x, \vec{\omega}, \vec{\omega}'), \tag{2.3}$$

where f_r is the probability of an outgoing direction $\vec{\omega}'$ for an incoming direction $\vec{\omega}$ at a point x. This distribution function is a powerful tool when it comes to rendering realistic light behaviours, as even very specific materials can be defined by it.

Although the BRDF provides a certain degree of freedom when defining a material, it must have the following properties:

- Helmholtz-Reciprocity: The ray direction can be reversed.

$$f_r(x, \vec{\omega}, \vec{\omega}') = f_r(x, \vec{\omega}', \vec{\omega})$$

- Positivity: The probability of an exiting direction cannot be negative.

$$f_r(x, \vec{\omega}, \vec{\omega}') \geq 0$$

- Energy Conservation: A material, that does not emit light, cannot have a higher exiting energy, than incoming energy.

$$\int_\Omega f_r(x, \vec{\omega}, \vec{\omega}') \cos\theta \, d\vec{\omega} \leq 1$$

The energy conversation property displays another important issue. While the energy of an exiting ray cannot be greater than the energy of an incoming ray, it can be less than the original energy. This means that a part of the energy can also be absorbed. If this would not be the case, everything would be as bright as the light source.

There many different types of BRDFs [PJH16] and countless different implementations. One of the most important, but also one of the simplest ones is the Lambertian BRDF. A Lambertian surface corresponds to the behaviour of light in Figure 2.2 (left) and is therefore simulating a perfectly diffuse surface. Although this behaviour is not possible in a real-world scenario, it is a suitable approximation for highly diffuse surfaces at a low performance cost. It is also possible to separate the BRDFs into a diffuse and a specular component. This approach is taken in the Cook-Torrance Model [CT82]. The idea of separating the BRDF into a diffuse part and a specular part is nowadays used in a variety of models.

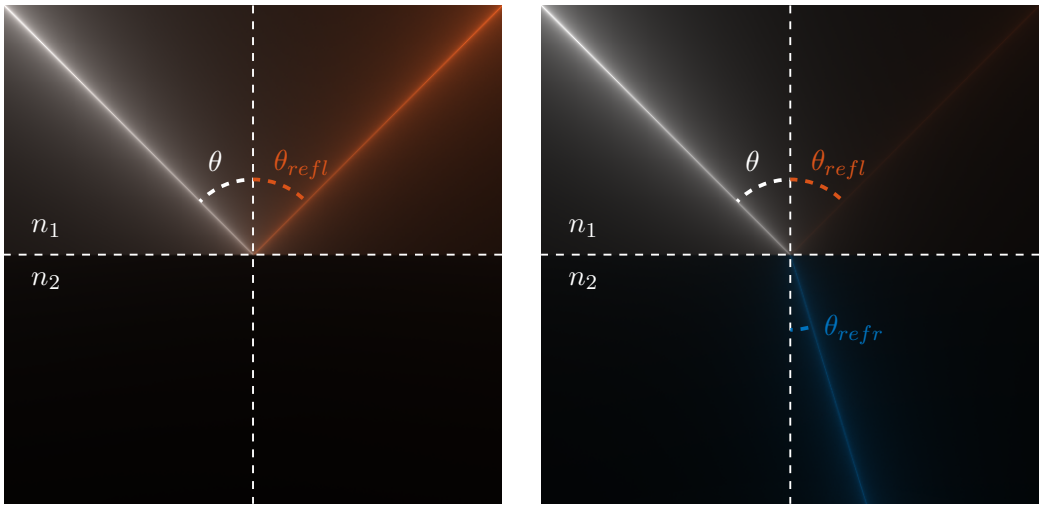
The behaviour of light when entering an object or medium is called a refraction. To be able to define what happens with refracted rays, the BTDF was introduced. It is an additional help to define the behaviour of light when passing through a medium. Additional information on the behaviour of light in participating media is provided in Chapter 5.

2.3.2 Specular Reflection

The calculation of a perfect specular reflection direction is simple. It was already introduced in ancient Greek [Hea21]. The calculation is

$$r = d - 2(d \cdot n)n, \tag{2.4}$$

where d is the direction of the incoming ray, while n is the surface normal, which must be normalized in advance. An illustration of a specular reflection is shown in Figure 2.3 (left).



Reflection Refraction

Figure 2.3: The behaviour of light when hitting a specular object and a refractive object. The light ray is traveling in a vacuum ($n_1 = 1$). The refractive object has the refractive index of a diamond ($n_2 = 2.4$).

2.3.3 Refraction

To calculate the direction and the probability of the refraction ray, a few basic principles must be explained. In Figure 2.3 (right) a light ray is shown, that is split into two. A part of the ray is reflected in a specular direction and another part of the ray is refracted. This happens, because different media have different refractive indices. The index of refraction indicates the difference in light speed in a medium. The speed of light is approximately $3 \cdot 10^8$ meters per seconds in a vacuum. The index of refraction in a vacuum is therefore 1. Water has an index of refraction of 1.333, which means that light travels 1.333 times slower in water than in a vacuum.

The indices of refraction of an entering and an exiting medium are needed to calculate the probability of a reflection in contrast to a refraction. This probability can be calculated with the Fresnel equation [Hec87]. The Fresnel equation has been formulated as

$$R_s(\theta) = \left| \frac{n_1 \cos\theta - n_2 \sqrt{1 - (\frac{n_1}{n_2} \sin\theta)^2}}{n_1 \cos\theta + n_2 \sqrt{1 - (\frac{n_1}{n_2} \sin\theta)^2}} \right|^2 . \tag{2.5}$$

This equation is expensive to compute, as there are square roots involved. Therefore, it is in most cases suitable to approximate the Fresnel equation with the Schlick approximation [Sch94] which is

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5, \tag{2.6}$$

with

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2. \tag{2.7}$$

In both of these equations, the angle θ of the incoming direction is used, as well as the refrative indices of the entering medium n_1 as well as the refractive index of the exiting medium n_2.

The most common case in computer graphics is, that one of the media (the entering or exiting) is air which has a refractive index of 1.000293 at 0°C [Hec87]. This makes it possible to further approximate the Fresnel equation by substituting the refractive index of air with the refractive index of a vacuum, which is 1.

The last thing that is needed, is the actual direction of the refraction. It is possible to calculate the angle of a refraction with Snell's Law [Hen01] and use this angle to calculate the direction of the ray. Transforming the equation for Snell's law,

$$\frac{\sin\theta_{\text{refl}}}{\sin\theta_{\text{refr}}} = \frac{n_2}{n_1}, \tag{2.8}$$

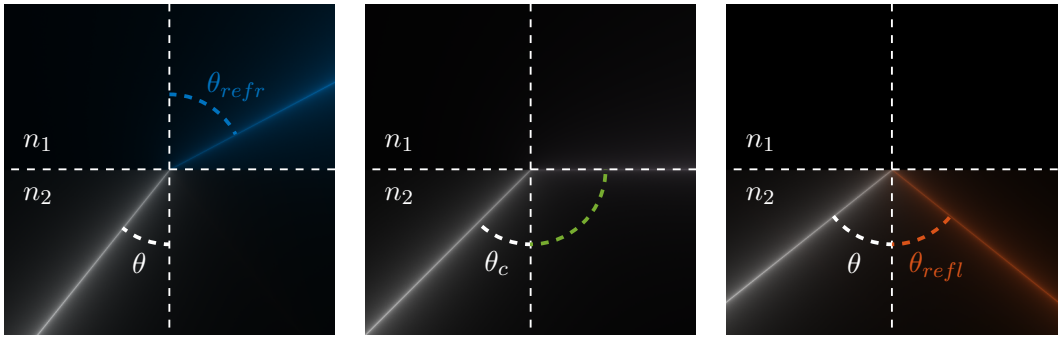
makes it possible to calculate both angles, just with the indices of refraction.

An important phenomenon, that can be seen when a light ray traverses from a medium with a higher refractive index to a medium with a lower refractive index is shown in Figure 2.4 (right). This behaviour of light is called total inner reflection. This phenomenon only occurs, when the angle of the incident light is greater than a critical angle θ_c. The critical angle can be calculated with

$$\theta_c = \sin^{-1}\left(\frac{n_2}{n_1}\right) \tag{2.9}$$

and can be seen in the middle of this figure.

The practical implementation of all formulae above into a recursive ray tracer can be seen in the next chapter.



Refraction | Critical Angle | Total Internal Reflection

Figure 2.4: A light ray traveling from water ($n_2 = 1.333$) to air ($n_1 = 1$).

CHAPTER 3

Recursive Ray Tracing

The first Ray Tracing algorithm was introduced in 1968 by Arthur Appel [App68]. It is an important algorithm that is the basis for many rendering frameworks that rely on it. Nowadays, there are many different implementation and different advancements that add different functionalities. One of these advancements is Recursive Ray Tracing (RRT) [Whi79], which includes the computation of objects that are see-through and the computation of shadows. This section focusses on RRT, as it contains important components that are used later for methods that trace rays in a similar fashion.

Many of the ideas and equations of the previous chapter are implemented in pseudocode. This pseudocode shows a detailed description of how Ray Tracing works, as just a written description would not be sufficient as an explanation. This pseudocode does not have the intent to be as efficient as possible, but should just be an educational tool in understanding the principle. It would therefore be necessary, to make appropriate changes, when implementing this algorithm with the intent to get an optimal performance. These necessary procedures for a performance enhancement are not in the scope of this thesis and are therefore not discussed.

To make the notation of all variables clear, directions are denoted with a vector above them, i.e., $\vec{x}$. Directions are assumed to be normalized automatically on initialization and to store the original length within their structure.

3.1 Pseudocode

The main intent of the Ray Tracing algorithm, as it is for all rendering algorithms, is to compute a color value for every pixel. This is where the pseudocode starts with Algorithm 3.1. As can be seen from this pseudocode snippet, there is no scene setup. This part is not included into this explanation, as it can fundamentally vary for different implementations.

Algorithm 3.1: RayTracing()

```
1 for every pixel do
2 |   d⃗ = ComputeCameraRayDirection(p_pixel)
3 |   color = Trace(o, d⃗, 0)
4 end
```

This algorithm starts with the main function which computes a color c for every pixel p_{pixel}. A loop that iterates through every pixel first computes a direction $\vec{d}$ which determines the camera ray which consist of the position o of the camera and the direction $\vec{d}$. Afterwards, it calls the Trace-function to compute the pixel color.

Algorithm 3.2: ComputeCameraRayDirection(p_{pixel})

```
1 fov_x = π/4
2 fov_y = (h/w) · fov_x
3 d⃗.x = (2 · p_pixel.x − w)/w · tan(fov_x)
4 d⃗.x = (2 · p_pixel.y − h)/h · tan(fov_y)
5 d⃗.z = −1
6 return d⃗
```

The ComputeCameraRayDirection-function (Algorithm 3.2) takes the pixel coordinates p_{pixel} in the image plane as input parameters and outputs a direction $\vec{d}$. The first computation that is made in this function is the computation of the field of view in the horizontal and vertical direction. Then the position of the pixel on the image plane gets computed in Lines 3 and 4. The image plane is one unit away from the camera origin, which is why the direction in the z-axis is -1.

The Trace-function in Algorithm 3.3 describes the heart of the Ray Tracing algorithm. It is therefore explained in detail as it is the key concept that should be understood when learning about Ray Tracing. The input parameters for this function are the origin o of a ray, its direction $\vec{d}$ and a depth parameter which is a simple counter value. This method has function calls in it, in which it can call itself, therefore it is called a recursive function.

The first step in this algorithm is the intersection of the objects in the scene with the ray (Algorithm 3.4). If the ray does not intersect with an object in the scene, it returns. In most cases a color value for a background would be returned here. Alternatively, an environment map could be used for a look-up. In the case that an intersection event occurred, the hit point p_{hit} is computed. The computation is shown in Line 3 and is a simple vector operation. Additionally, the object on which the intersection occurs, is saved. This can be done differently and can vary for different implementation as the scene setup may be different.

Algorithm 3.3: TRACE($o, \vec{d}$, depth)

```
1  intersection = INTERSECTSCENE(o, d⃗)
2  if no intersection then return
3  p_hit = o + d⃗ · intersection.dist
4  obj = intersection.obj
5  if depth ≤ threshold then
6      if obj is specular then
7          d⃗_refl = SPECULARDIRECTION(d⃗, obj.n⃗)
8          return TRACE(p_hit, d⃗_refl, depth +1)
9      end
10     if obj is refractive then
11         d⃗_refl = SPECULARDIRECTION(d⃗, obj.n⃗)
12         color_refl = TRACE(p_hit, d⃗_refl, depth +1)
13         d⃗_refr = REFRACTIVEDIRECTION(d⃗, obj.n⃗, obj.refr_i)
14         color_refr = TRACE(p_hit, d⃗_refr, depth +1)
15         prob = FRESNELTERM(d⃗, obj.n⃗, refr_i)
16         return (1 − prob) · color_refl + prob · color_refr
17     end
18 end
19 s⃗ = p_light − p_hit
20 intersection_s = INTERSECTSCENE(p_hit, s⃗)
21 if intersection_s or
22   intersection_s.dist > s⃗.length then
23     return obj.color · light.intensity
24 else
25     return obj.color
26 end
```

In Line 5 the depth value makes its first occurrence. As this is a recursive function, the depth values indications, at which depth of the recursion the algorithm is currently on. It is necessary to keep track of the depth, as it may be possible that there is an infinite number of recursions and that is not possible to compute. This can be prevented by keeping track of the depth and aborting the algorithm, if the recursion gets deeper than a specified threshold. It should be considered that a higher depth value can lead to a performance decrease, that may not be acceptable, as the increase in quality of the results could be minimal after a certain threshold. While just checking if the depth value is above a threshold is a straightforward approach, there are also approaches that handle this situation superior. One of these approaches is Russian Roulette [WJZ95] which is used in most implementations, as it is still simple to use and provides statistical correct results. In this implementation, after a predefined threshold, the object is considered as a diffuse object and therefore returns a color without a further recursion.

The next step it to separate the type of object that was intersected with the ray. For a simplification an object can only be diffuse, specular or refractive, but not multiple types at once. This means that an object can only have the perfect properties, that only exist theoretically. The theory for this was already discussed in Section 2.3. The case in which the object is specular is the simplest one. First, the reflection direction is computed which is described in Algorithm 3.5. In the second step a recursion of the Trace-function is performed. As an input for this function, the previously computed hit point p_{hit}, the reflection direction $\vec{d}_{\text{refl}}$ and an increase value are provided. The results of this recursion are returned directly as a result of this function. To imagine the process, it is sufficient to imagine a perfect mirror. When a camera ray hits this mirror, it does not show any property of the mirror itself, but just the reflection that can be seen in the mirror. This reflection is computed with the recursive call of the Trace-function.

For the case of a refractive object, the first part is similar to the specular case. Again, the reflection direction $\vec{d}_{\text{refl}}$ gets computed, but this time the color of the recursive trace function is saved to a temporary value. Afterwards, the refractive direction $\vec{d}_{\text{refr}}$ is computed (Algorithm 3.6). Like in the specular case, the previously computed hit point p_{hit}, the refractive direction $\vec{d}_{\text{refl}}$ and the increased depth value are the input for the recursive Trace-function. The return value of this function is saved to another value as it is needed later.

The theory of refractive objects was discussed in Section 2.3.3. Line 15 computes the Fresnel Term 3.7 which is the probability of a reflection. This probability is used to linearly interpolate the colors for the reflection direction and the refraction direction that were previously computed. In contrast to the specular case, there is not one recursion, but two recursions. This can decrease the performance of the algorithm rapidly and it is therefore important to adjust the depth value accordingly when rendering refractive objects.

In this implementation, the computation of shadows is added. First, a shadow ray is computed by creating a ray that starts at the hit point and ends at the position of the light p_{light} (Line 19). Afterwards, the scene is intersected with the shadow ray. This is a simple addition to the framework, with a performance cost, that is not insignificant due to the intersection routine.

At the end of the algorithm, in Line 21, the computation of the color of a diffuse object starts. With the information of the shadow computation, it now must be determined, if the ray is in the shadow or is illuminated by the light. In the case that there is no intersection with a scene object or the distance to the nearest intersected object is greater than to the light, this point is not in a shadow. The color of the object is now multiplied with the intensity of the light and the result is returned. If the object is in the shadow, only the color of the object is returned, without the increase of the intensity of the light.

Algorithm 3.4: IntersectScene$(o, \vec{d})$

```
1 for every obj in the scene do
2 |   obj.Intersect(o, d⃗)
3 end
4 return the closest intersection or ∞
```

The computation of if a ray intersects an object in a scene, is one of the components, from which the performance suffers the most. Therefore, in modern rendering engines, different acceleration structures are used to increase the performance. The basic principle, as shown in Algorithm 3.4, is that for every object in the scene, the nearest intersection, if there is one, is computed. For the whole scene, the closest intersection of all objects is returned, as it is, in this simple case, the only relevant one. It is important to note, that intersection that are in the other direction of the ray should be discarded, as they are behind the point of interest. Furthermore, the distance of the nearest intersection should always be greater than zero to avoid an intersection with the point from which the ray is cast.

Algorithm 3.5: SpecularDirection$(\vec{d}, \vec{n})$

```
1 d⃗ = d⃗ − 2 · (n⃗ · d⃗) · n⃗
2 return d⃗
```

The computation of the reflection direction was already explained in Section 2.3.2. There is no difference from the actual theory to this implementation in Algorithm 3.5.

Algorithm 3.6: RefractiveDirection($\vec{d}, \vec{n}, \text{refr}_i$)

```
1  refr_{i_in} = refr_i
2  refr_{i_out} = 1
3  cos θ_1 = d · n
4  if cos θ_1 > 0 then
5  |   n = −n
6  |   Swap(refr_{i_in}, refr_{i_out})
7  else
8  |   cos θ_1 = − cos θ_1
9  end
10 refr_ratio = refr_{i_out} / refr_{i_in}
11 cos θ_2 = 1 − refr_ratio^2 · (1 − cos θ_1^2)
12 if cos θ_2 < 0 then return
13 d = d · refr_ratio + n · (refr_ratio · cos θ_1 − √(cos θ_2))
14 return d
```

To compute the refraction direction (Algorithm 3.5), a few topics that were previously discussed in Section 2.3.3 must be considered. First, in this implementation, there is never a refractive interaction, where one of the media is not air. One of the two media is always air, while the other one is defined by the object properties. For the computation of Snell's law, it is necessary to determine which of those two media is the entering and which is the exiting one. Therefore, the angle between the ray direction and the surface normal is made according to the results. This is shown in Lines 4 to 9, where the starting point would be in the medium, if the angle is greater than zero.

With these changes the ratio $\text{refr}_{\text{ratio}}$ of the refractive indices refr_i can be computed, as well as the second angle. A visual example of this phenomenon can be seen in Figure 2.3. After computing both angles, and with the ratio of the refractive indices available, the direction of the refraction can be computed. This can be seen in Line 13.

Algorithm 3.7: FresnelTerm($\vec{d}, \vec{n}, \text{refr}_i$)

```
1 cos θ = d · n
2 R_0 = (1 − refr_i)/(1 + refr_i)
3 return R_0^2 + (1 − R_0^2) · (1 − cos θ)^5
```

The Fresnel term can be computed as it was explained in Equation 2.5. Instead of implementing the complex Fresnel equation, the Schlick approximation is chosen as a suitable replacement. As is shown in Algorithm 3.7, the implementation does not differ from the mathematical theory that was discussed before.

3.2 Results

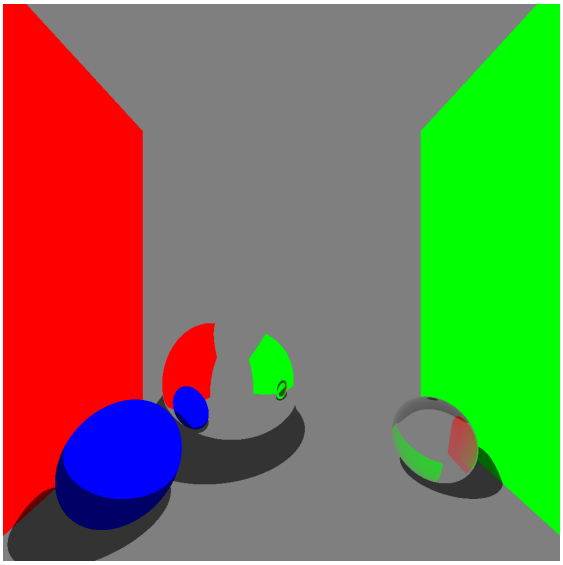


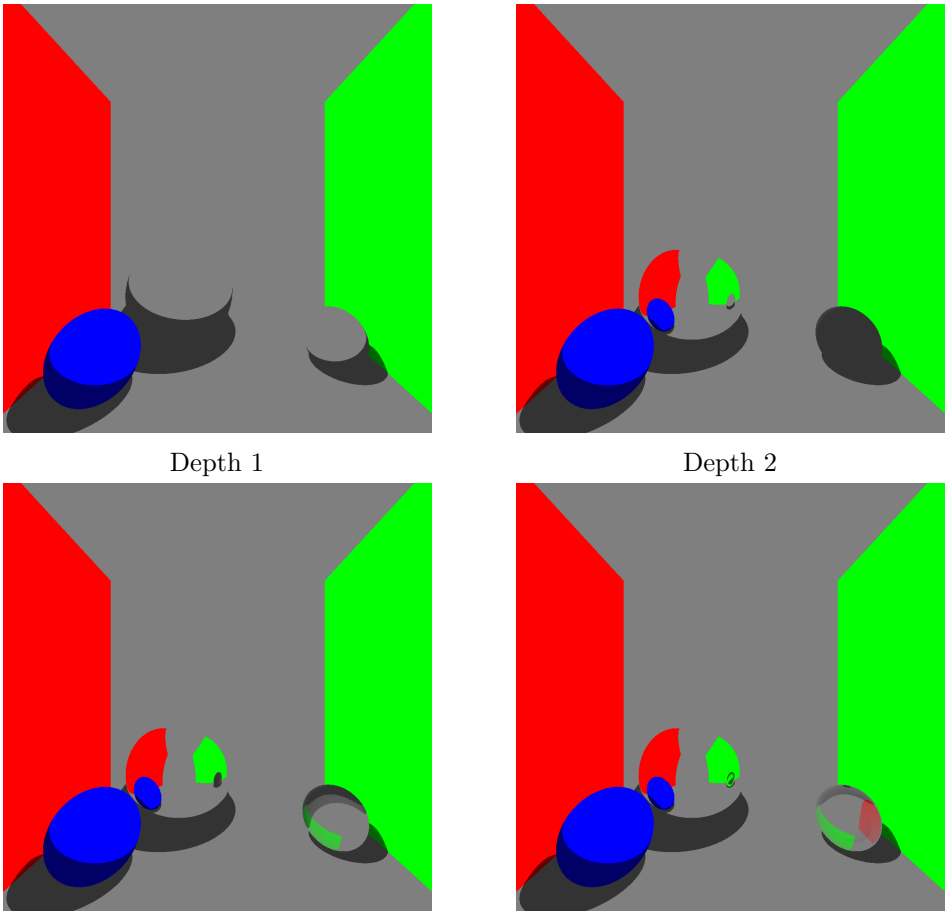
Figure 3.1: The result of rendering a scene with RRT.

In Figure 3.1 the result of RRT can be seen. In this Cornell box, the left most sphere has a completely diffuse surface. The sphere in the middle has a perfect specular surface and the sphere to the right is a refractive object. This refractive sphere has a refraction index of 1.5. The shadows are visible and a light source that is in the middle, top half of the image illuminates the scene.

3.2.1 The depth value

The depth value has a great impact on the outcome of the image. A comparison of the scene with different depth values can be seen in Figure 3.2. The first image shows a scene with a depth value of 1. This does not affect the walls and the diffuse sphere, as there is not additional recursion needed to compute these values. The specular and refractive sphere are both displayed as diffuse spheres, as it is the way that it is implemented in this case. There can be different fallback methods, for when the depth value has been reached and a specular or refractive object has been hit.

The different depth values lead to different results in the other images. In the second image the depth value is 2, therefore the specular surface can already produce a reflection. A depth value of 3 generates the first results of the refractive sphere, as is shown in the corresponding figure. The last image of Figure 3.2 shows the results of an image with a depth of 4. Not only can a part of the red wall be seen in the refractive sphere, but there is also a difference in the reflection of the refractive sphere in the specular sphere.



Depth 1 Depth 2

Depth 3 Depth 4

Figure 3.2: The result of rendering a scene with RRT and different depth values.

To further evaluate different depth values, a different scene setting was chosen in Figure 3.3. In this case, four specular spheres are positioned in an equal distance around a white diffuse sphere in the middle. The specular spheres have the diffuse color green as a fallback when a predefined depth value is reached. While this would certainly not be a good choice in a scene that tries to approximate a real-life scenario, it is a good tool to evaluate this case. The first image shows what happens at a depth value of 1. All spheres are displayed as diffuse surfaces, as there are no further traces possible. In next image, at a depth value of 2, the first reflections of the white sphere are already visible. The reflections of the other specular spheres are in their diffuse color, green. This trend continues in the next images. In the final image the depth value is increased to 100 and the diffuse color of the specular surface is not visible anymore. The pattern, that emerged from repeatedly reflecting the white sphere is a fractal [Man77]. Fractals are an interesting field in science and some can be generated with Ray Tracing [HSK89], as can be seen in this example.

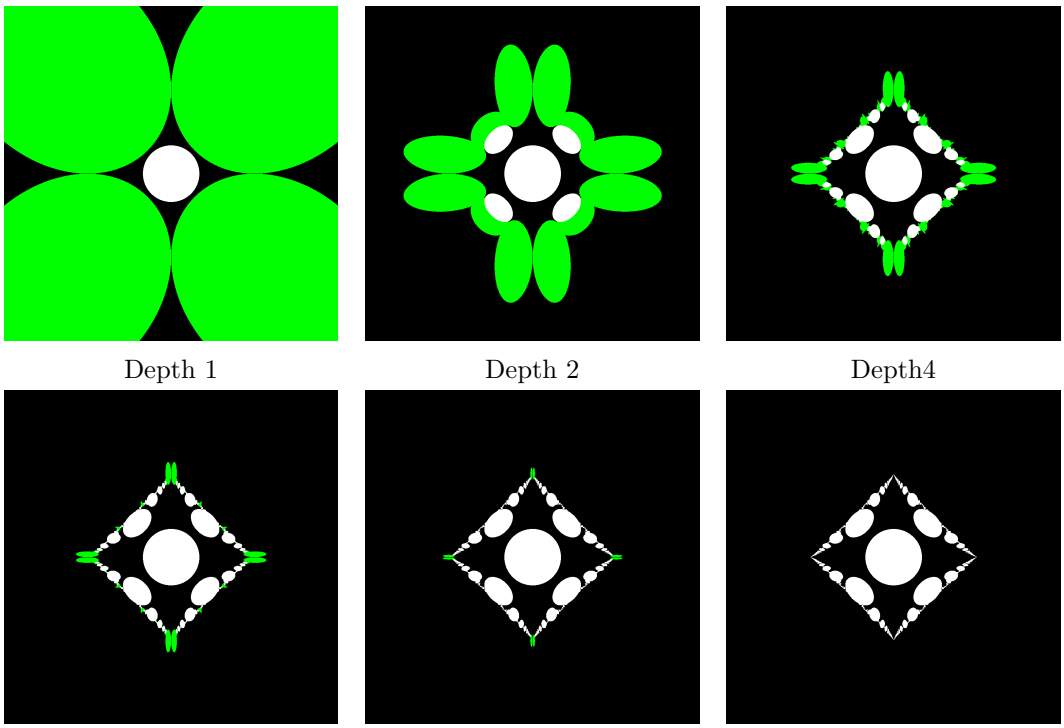


Figure 3.3: The generation of fractals by rendering a scene with RRT and different depth values.

CHAPTER 4

Photon Mapping

After the examination of Ray Tracing, it is possible to evaluate Photon Mapping (PM). This chapter discusses the theory behind PM in detail, as well as the visual improvements that are created by it. The main focus is on the photon tracing and the photon gathering steps. Furthermore, there are descriptions of the extensions PPM and SPPM which are explained in Sections 4.2 and 4.3, respectively.

Photon Mapping is an extension of the previously explained Ray Tracing algorithm. It has the same possibilities as Ray Tracing, but extends them to make the rendering of global illumination effects possible, which make it that much more powerful. Global illumination and its influence on the realistic representation of a scene was already covered in Section 2.2 and is therefore not discussed further.

4.1 Photon Mapping - The Original Approach

The idea for PM is based on the physical concept of a photon, a particle of light. The name originates from the fact that these photons are traced throughout the scene and are stored in a photon map, which is used for the calculation of the illumination. This approach was developed from 1993 to 1994 by Henrik Wann Jensen in his PhD thesis [Jen96b] and was first published in 1995 [Jen96a]. During the time of development, there were general issues in rendering, that no algorithm could solve yet. The main problems besides indirect illumination and caustics, which were already discussed in Section 2.2, was the transportation of light from a specular surface to a diffuse surface and again to a specular surface. This kind of light path is called SDS-path and can, for example, be seen as the light patterns at the floor of a swimming pool. SDS-paths are difficult to compute as the probability that a ray takes exactly the desired path is marginally small. A person without a general knowledge of computer graphics can often not describe the issues, but scenes that were rendered with algorithms that do not feature these effect, often seem unrealistic to a high percentage of viewers.

This was and still is a major issue in computer graphics, as the computation of global illumination is expensive. PM was one of the first algorithms to solve this problem efficiently. Nowadays, PMg is a well-known approach in computer graphics and is documented well [Jen01, JCKS02] for extensive studies.

4.1.1 Methodology

The PM approach consists of two passes. The first pass consists solely of the construction of the photon map, while the second part estimates the illumination with the previously traced photons with Ray Tracing.

The definition of a photon can vary for different implementations, but it must contain at least elements described in Table 4.1.

Symbol	Description
p	the position of the photon
power	the power of the photon for all color channels
$\vec{d}$	the incident direction of the photon

Table 4.1: The structure of the Photon data type.

An illustration of photons in a scene is shown in Figure 4.1. The black dots are the photon positions from which the incident direction of the photon is displayed. For visualization purposes, the distance of the incident direction vector was adjusted by the average power of the photon to give a better understanding of the behaviour of a photon.

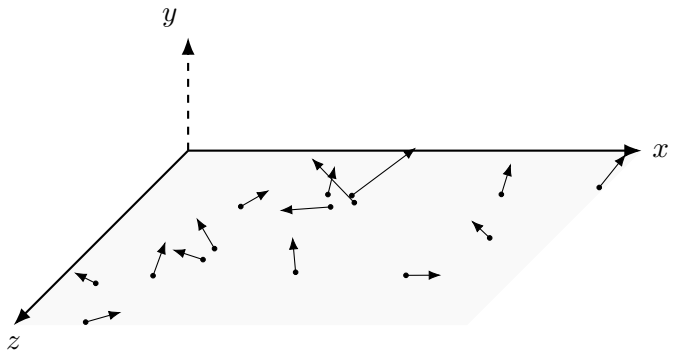


Figure 4.1: A visualization of photons with the incident direction vector which is scaled by the average power of the photon.

4.1.1.1 First Pass - Photon Map Creation

Algorithm 4.1: FillPhotonMap()

```
1 for every light in the scene do
2 |   for the amount of photons for this light do
3 |   |   p = ComputePhotonStartingPoint(light)
4 |   |   d⃗ = ComputePhotonStartingDirection(light)
5 |   |   power = light.power /(amountof photons forthis light)
6 |   |   TracePhoton(o, d⃗, power, 0)
7 |   end
8 end
```

For the creation of the photon map, a predefined number of photons must be traced through the scene. Algorithm 4.1 describes the journey of a photon through the scene. The path of every photon begins at a light source. If there are multiple light sources, the number of photons can be split up. Brighter light sources should cast more photons than dimmer light sources. This should be considered when defining the overall number of photons, so that even from dimmer light source, a large enough number of photons is cast. The starting position of the photon must be computed according to the type of light. In the trivial case of a point light source, the position of the light source is the position from which the photon is traced.

The starting direction of the photon that is traced through the scene depends on the type of light. Again, for the trivial case of a point light source, the position can be computed as random direction from the light source. In the case, that there is not much geometry in the scene, this behaviour would lead to the tracing of many photons that would never hit an object. Therefore, projection maps can be introduced to pre-filter the possible directions of the photons in a way that they are hitting something. The starting power of the photon is the power of the light divided by the number of photons that are traced from this light. This means that if we combine all the power of the photons that are traced from one light source, the result is the original power of the light source.

The TracePhoton-function (Algorithm 4.2) is very similar to the trace function of a Ray Tracing algorithm. Therefore, some parts just reference the algorithm from Chapter 3 to avoid redundancy. Again, to keep things simple, there are only three different types of objects. An object can either be completely diffuse, specular or refractive.

The important changes start at Line 2, where the storing of the photons begins. At this point in the algorithm, the object is either completely diffuse or the depth values prohibits the photon from further tracing, which means that the current object is viewed as a completely diffuse object at this depth level. Photons that hit a specular surface or a refractive object are just redirected into a different direction, but nothing is stored to the photon map at this point. At a diffuse surface, the reflection probability is computed first by finding the maximum of the different color channels from the object.

Algorithm 4.2: TracePhoton($o, \vec{d}$, power, depth)

```
1 Algorithm 3.3 Lines 1 - 18
2 prob = max(obj.color.r, obj.color.g, obj.color.b)
3 power = power ·(obj.color / prob)
4 StorePhoton(Photon(p_hit, d, power))
5 if power > ξ then
6 |   d = DiffuseDirection(obj.n)
7 |   TracePhoton(p_hit, d, power, depth +1)
8 end
```

This means the decision, of if a diffuse reflection should occur, is solely based on the maximum energy at a point. With the probability of reflection, it is possible to compute the power of the photon at the current point. With these values the photon can be stored into the photon map.

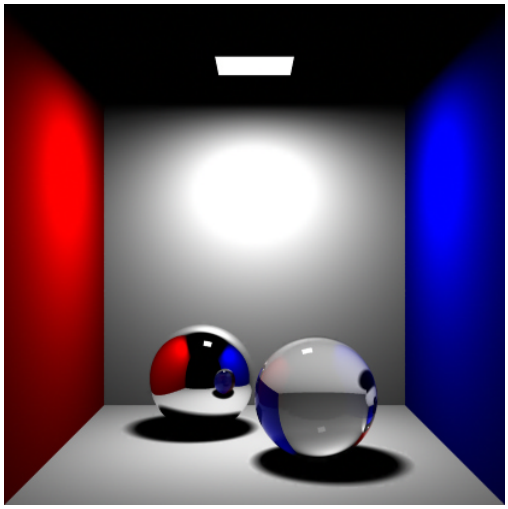
In the last step, the power of the photon determines if it is traced further. Therefore, the power is compared to a random value $\xi \in [0, 1)$. If the power is greater than ξ, the photon is traced again. The direction at this diffuse intersection is computed by determining a sample on a hemisphere, which is directed into the direction of the normal at the current position. This is equivalent to a Lambertian BRDF.

The storage of the individual photon is an important factor for the overall performance of the renderer. For the second pass, the photon map is traversed many times, therefore an efficient data structure is necessary. In most cases, balanced kd-trees [Ben75] are used. These trees guarantee a worst-case performance of $O(logN)$. After all photons are stored inside this data structure, the kd-tree must be balanced to allow this performance.

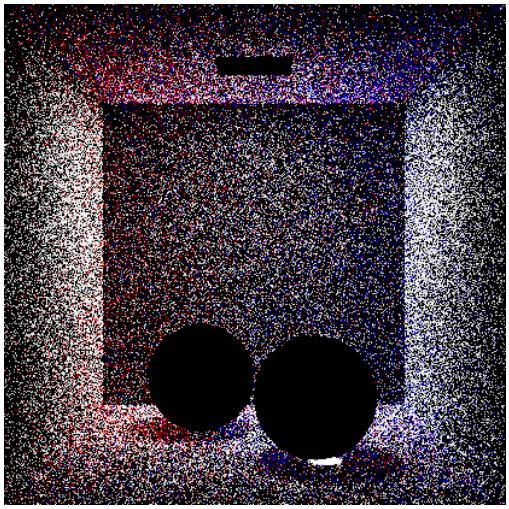
Another important addition, that is made in most frameworks, is to store the caustic photons into a separate map. A caustic photon must have hit at least one specular surface or refractive object, before hitting a diffuse surface. Caustics are an important factor in the representation of realistic lighting and consist often out of very fine details. It is therefore, in many cases, necessary to increase the number of photons for the caustic photon map, to be able to approximate these lighting patterns in an efficient way.

4.1.1.2 Second Pass - Ray Tracing

To understand, how the photons behave in a scene, every photon of the photon map can be displayed as a point, with its power as the color of this point. The results of this process is shown in Figure 4.2 (right) with the ray traced scene on the left. The contribution of the color of the walls to the surrounding area is the reason, the final rendering has indirect illumination in it. Another important factor are the photons that approximate the caustics from the sphere in the foreground.



Ray Tracing



Photon Map

Figure 4.2: Visualization of how photons are distributed in the scene. On the left the ray traced scene and on the right the photons that are stored in the photon map. Source: [JCKS02]

In the second pass, the first steps are very similar to the Ray Tracing example from Chapter 3. The differences are only in the TRACE-function, therefore only this algorithm is evaluated in Algorithm 4.3. Even the beginning of this function is identical to the previous TRACE-functions, only afterwards, important changes are necessary.

Algorithm 4.3: TRACE$(o, \vec{d}, \text{depth})$

```
1 Algorithm 3.3 Lines 1 - 18
2 photons = SEARCHFORPHOTONS(p_hit)
3 for every photon in photons do
4 |   color = color +(obj.color · photon.power)
5 end
6 color = color /(πr²)
7 return color
```

The first thing to do is to search for photons that are in the area of the hit point. There are different methods on how this can be implemented. In the general case, a sphere is expanded from the hit point, until enough photons are inside the sphere. This process can be seen in Figure 4.3 where a sphere is expanded until four photons are inside of it. In the first step (blue, inner sphere) no photons are inside the sphere, therefore it must be expanded further, which results in in the larger sphere (green). It is possible to use other shapes than a sphere, but this is only practical in specific use cases.

This step is the performance bottle neck of the algorithm, which makes it that much more important, that the data structure, that is going to be used for the storing of photons, is as efficient as possible. The SearchForPhotons-function is not discussed further, as it is heavily reliant on the data structure that is used. The radius term r that is used at Line 6 is the radius of the final sphere that surrounds the correct number of photons.

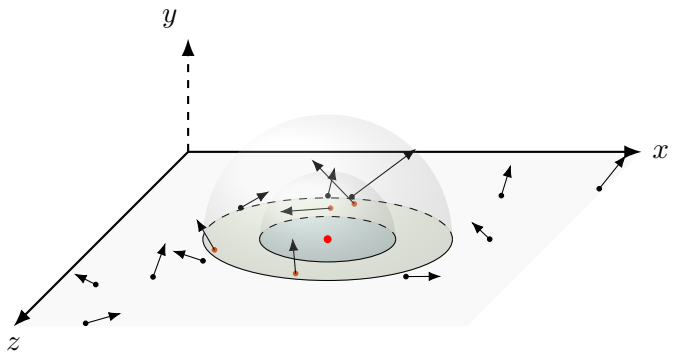


Figure 4.3: To search for a specific number of photons (int this case four) around a point. The sphere is expanded until enough photons are in enclosed by it. In the first step (blue, inner sphere) no photons are found within the area. The sphere is therefore expanded (green, outer sphere) to find four photons inside of it.

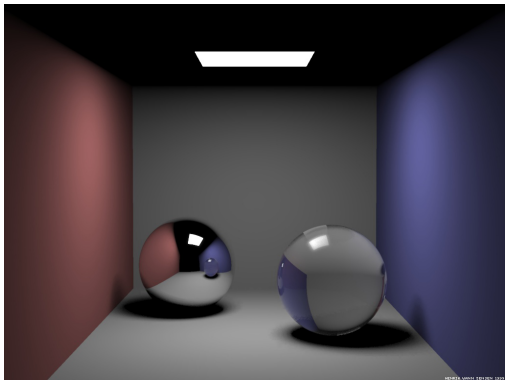
After gathering the specified number of photons, the radiance at this point must be calculated with

$$L_r(x,\vec{\omega}) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x,\vec{\omega}_p,\vec{\omega}) \Delta\Phi_p(x,\vec{\omega_p}). \tag{4.1}$$

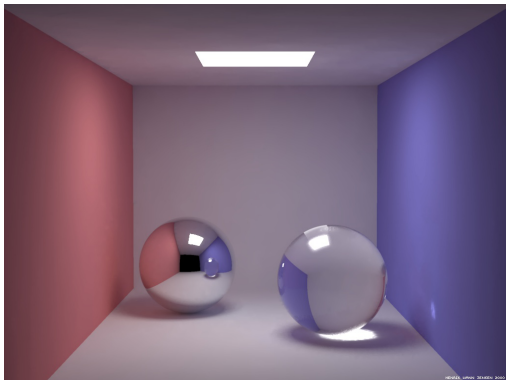
The radiance L_r at a point x into a direction $\vec{\omega}$ is the sum of the BRDF f_r at this point x which is multiplied by the power of every photon $\Delta\Phi_p$ that is selected before. This whole term is than divided by the square of the radius of the sphere r and by π.

This computation can be seen in the Algorithm 4.3 at Lines 3 - 6. In this simple case the BRDF is just the diffuse color of this object.

As can be seen with the code segments that are specified above, PM can use a lot of the structure of Ray Tracing. Therefore, the PM technique is in most cases a possible extension to an existing Ray Tracing framework, without having to sacrifice much of the underlying structure of the original algorithm. The Trace-functions of both passes can be combined into one, that just depend on an additional boolean value that can be provided as an input. This boolean value can separate the different code segments and can specify which pass is active at the time.



Recursive Whitted-Style Ray Tracing



PM

Figure 4.4: The comparison of a scene that was rendered with Recursive Whitted-Style Ray Tracing to a scene that was rendered with PM. Source: [JCKS02]

4.1.2 Discussion

A comparison of a scene that uses PM compared to a scene that uses Recursive Whitted-Style Ray Tracing is shown in Figure 4.4. The noticeable differences are the illumination changes and the caustics, which both add a realistic appeal to the scene. These are also the advantages of the algorithm, as caustics and indirect illumination progress are visible in a sufficient quality even at a lower number of photons. Previews, and lower quality images that are still displaying global illumination, can therefore be rendered in a quick and efficient way. Another advantage is that shadows do not have to be computed in an additional step, as this is a side benefit of all global illumination techniques. This effect can be seen in Figure 4.2 (right) where the photons are displayed directly as points and the areas with a lower density of photons will be darker in the final image.

PM is a biased and only theoretically consistent approach. For an infinite number of photons, PM would be consistent, but this can never be the case. This is one of the shortcomings of PM, as there is always an error that can never be eliminated. The conclusion of this is, that it is better to have as many photons as possible. Having as many photons as possible leads to another problem, as storing that many photons means that the memory consumption increases, too. The storage of a photon can be minimalized to a certain degree, but at some point, storing a number of photons that is that big leads to memory problems that have to be addressed.

The usage of a photon map that holds a large number of photons, naturally lead to an increase in rendering time. While the overall increase in rendering time must be accepted, if a high quality must be achieved, it is drawback, that no temporary results can be viewed. This means that if an error occurs at an early stage of the rendering progress, this error cannot be detected until the whole image is finished. Even with the mentioned short comings, PM is still a respected method in computer graphics.

4.2 Progressive Photon Mapping

The next step in the evolution of PM is Progressive Photon Mapping (PPM) [HOJ08]. This approach tries to eliminate some of the disadvantages that are problematic when using the original approach. The main idea is to progressively add photons that decrease the error over time. Therefore, it is also made possible to create images even while rendering.

4.2.1 Methodology

PPM is a multi-pass approach, compared to the original PM method, which is just a two-pass approach. The first change is doing the Ray Tracing first and multiple photon tracing passes later. The first pass only stores the hit points into a data structure, so that characteristics of them can be adjusted after every pass. The other passes progressively add photons to the previously stored hit points. Therefore, this multi-pass approach can be explained by just evaluating two passes, where the second pass is repeated multiple times. The important adaptions are made at the end of every photon tracing pass, where the radius and flux must be corrected, while adding more photons to every hit point.

4.2.1.1 First Pass - Ray Tracing

The first pass is very similar to the original Ray Tracing implementation. There is only one key difference in the algorithm. Instead of computing a color when hitting a diffuse surface, this hit point is stored into a data structure. In this data structure, the values of Table 4.2 are stored for every hit point.

Symbol	Description
p	the position of the hit point
$\vec{n}$	the surface normal at this position
$\vec{d}$	the incident direction of the ray
p_{pixel}	the x and y position on the image plane
r	the current radius at this point
n	the current number of accumulated photons at this point
τ	the current reflected flux at this point

Table 4.2: The structure of the HitPoint data type.

The values in this data structure, that are important for the progressive updates of the radiance, are the radius r, the number of photons n and the flux τ. As in the original PM approach, the photons are gathered at a point, by evaluating a sphere, and gathering the photons inside of it. With an increasing number of photons, that are added after every photon tracing pass, this radius must be adjusted over time. The same principles hold for the number of photons, that are gathered for this point and the accumulated flux. The correction of theses variable during the rendering process is discussed in the next section.

4.2.1.2 Multiple Passes - Photon Tracing

The photon tracing pass can be executed for just one time or multiple times. It traces photons through the scene exactly as described in the original PM approach. The number of photons that are traced in this pass are in most cases separated into equal parts of the desired number of photons that should be in the final image. Either for every photon individually while tracing, or for all photons collectively at the end of the pass, for every hit point in the scene the photon is evaluated if it is in the current radius. After temporarily adding all photons, a few adjustments must be made. These adjustments are discussed for every pass and not for every individual photon.

Radius Correction

The radius in which each hit point is searching for photons is a predefined value, that depends on the scene and the resolution of the image. While adding multiple passes, the radius should decrease, as more and more photons are added to this hit point. The first step is to calculate the corrected radius $\hat{r}$ with the help of other values that must be calculated. The data structure that is being used for the storing of hit points has also the ability to store the current radius r and the number of photons n, which is useful for this calculation.

The new local density estimation

$$\hat{d} = \frac{n+m}{\pi r^2}, \tag{4.2}$$

can simply be calculated by adding the new number of photons m, to the number of photons that were already added in previous passes n. The radius r stills stays the same for the calculation of the density, as the density is to be assumed constant within the radius.

The new number of photons $\hat{n}$ can be calculated in two ways, which are needed to get the adjusted radius. The first way to calculate $\hat{n}$ is by simply rearranging the equation of the local density estimation to $\hat{n} = \pi \hat{r}^2 \hat{d}$. In this case, the calculation already considers the adjusted radius $\hat{r}$. Another way to calculate $\hat{n}$ is to add only the number of photons to n that should be in the final calculation. This can be done by introducing a new variable $\alpha \in (0, 1)$ which is predefined. α determined the number of photons that are added to n at every pass. The calculation of $\hat{n}$ with α is $\hat{n} = n + \alpha m$. The previously discussed equations can now be combined and rearranged to calculate the corrected radius $\hat{r}$ where

$$\pi \hat{r}^2 \hat{d} = n + \alpha m, \tag{4.3}$$

which can be written as

$$\hat{r}^2 \frac{n+m}{r^2} = n + \alpha m, \tag{4.4}$$

that can be reformulated as

$$\hat{r} = r \sqrt{\frac{n + \alpha m}{n+m}}. \tag{4.5}$$

Flux Correction

The correction of the flux at a point can simply be seen as adding the combined flux τ_m of all photons that are added in this pass to the flux τ_n the is received before this pass. The flux of a number of photons can be calculated by the sum of photon power Φ which has been multiplied by the BRDF f_r. This is the same equation as for the calculation of the radiance in the original PM algorithm (Equation 4.1).

The flux τ_m of the photons that are added in this pass could just be added to the flux τ_n that was received before this pass if the radius would stay constant. By adjusting the radius, the flux has to be adjusted, too. The corrected flux $\tau_{\hat{n}}$ can be calculated with

$$\tau_{\hat{n}} = (\tau_n + \tau_m)\frac{\pi\hat{r}^2}{\pi r^2}, \tag{4.6}$$

where

$$\hat{r} = r\sqrt{\frac{n + \alpha m}{n + m}}. \tag{4.7}$$

This can also be written as

$$\tau_{\hat{n}} = (\tau_n + \tau_m)\frac{n + \alpha m}{n + m}. \tag{4.8}$$

Radiance Evaluation

The evaluation of the radiance can be done after every pass. This means, that updates of the rendering process can be provided progressively. The radiance L at a hit point can be evaluated with

$$L = \frac{1}{\pi r^2}\frac{\tau}{k}. \tag{4.9}$$

The equation only needs the radius r and the flux τ that are stored with each hit point for this calculation. The value k is the number of photons that are emitted throughout the whole rendering process. By dividing the flux τ through this amount k the flux is normalized. This is necessary, because the power of the photons cannot be divided by the number of photons at the beginning, as progressive updates would not be possible anymore.

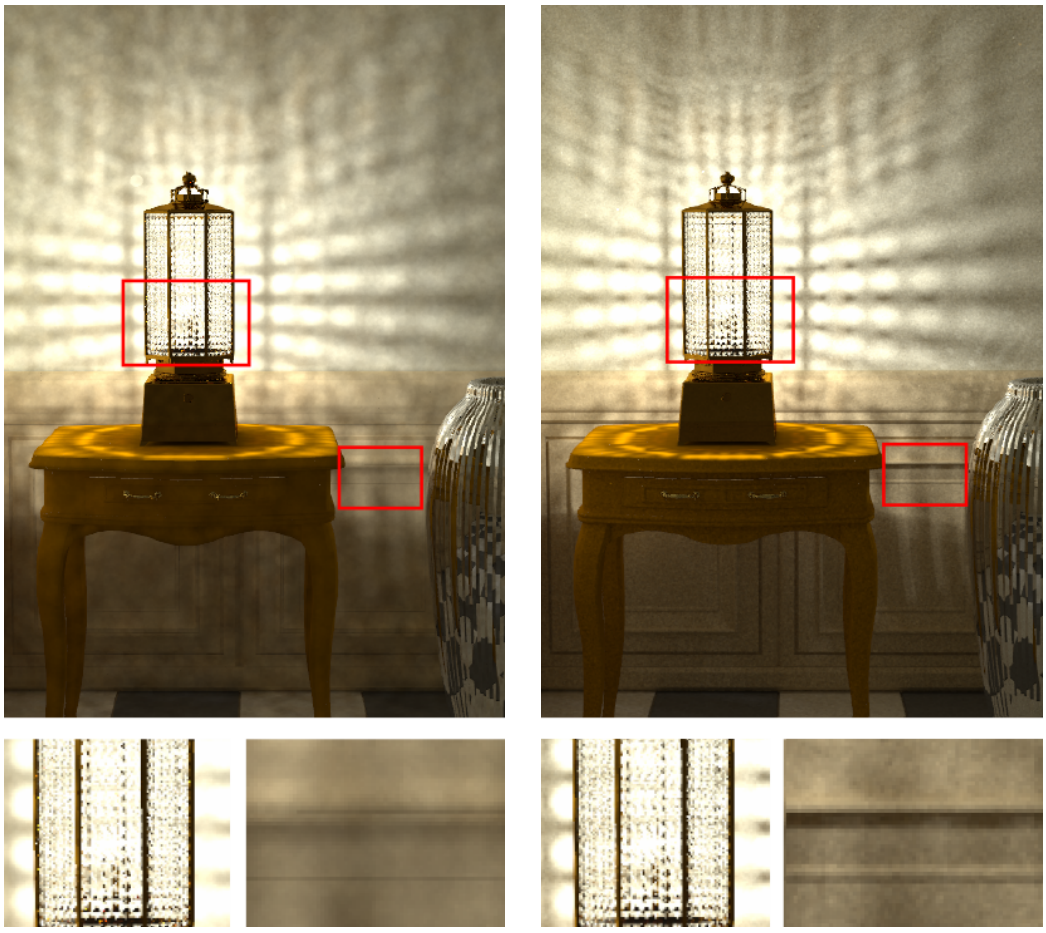
4.2.2 Discussion

PPM is in many ways an improvement to the original PM algorithm: By being a consistent algorithm, not only in the theoretical sense, PPM can produce more detailed results. A comparison is shown in Figure 4.5, where the light pattern on the wall is more detailed in the image that uses PPM (right), compared to the image that uses PM (left). This can also be contributed to the fact, that PPM can avoid using big amounts of memory for its computations, by splitting the photon tracing passes. Not only can the end results therefore be computed with amounts of photons that are not possible with PM due to memory restrictions, PPM is also able to produce progressive update while rendering.

This can be practical to detect errors early in the rendering process or being able to abort it when being satisfied with the results after an earlier photon tracing pass and not having to wait for the final image.

Although PPM is an improvement to the original Photon Mapping algorithm, it still produces artifacts in specific cases. Theses artifacts can be eliminated by using SPPM instead, which is briefly discussed in the next section.

An implementation of this method that was created during the process of this theses is provided in Appendix B.



PM PPM

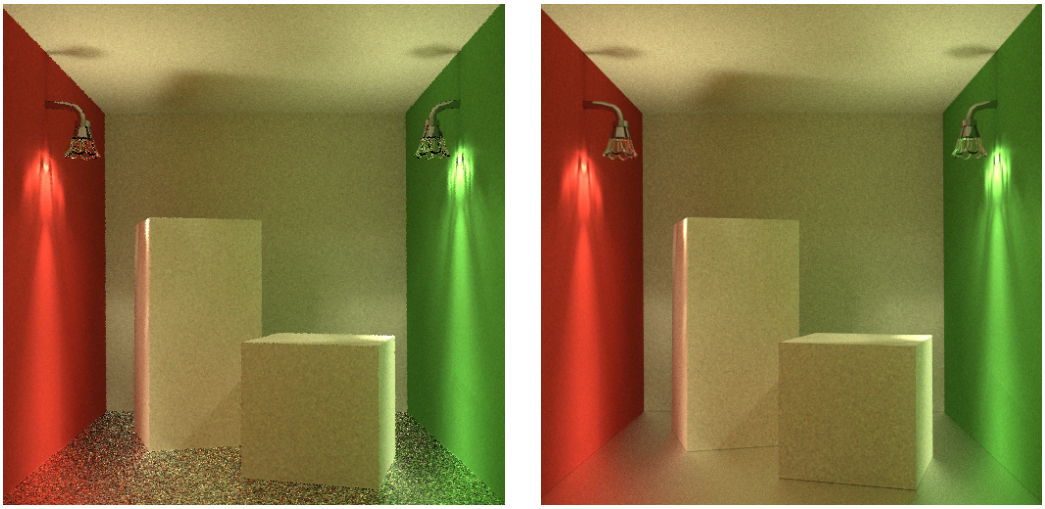
Figure 4.5: The comparison of a scene that was rendered with PM to a scene that was rendered with PPM. The scene that was rendered with PM has light patterns on the wall that were unintentionally blurred and which remain sharp for PPM. Source: [HOJ08]

4.3 Stochastic Progressive Photon Mapping

Stochastic Progressive Photon Mapping (SPPM) [HJ09] is an extension to PPM. The theory behind PPM has a flaw, that can create artefacts in some cases, which SPPM provides the solution for. In PPM, the evaluation of the radiance is correct for every point, which is in most cases enough to create visually correct results. In some cases, the correct evaluation of the radiance at a point is not enough, as the correct evaluation of the radiance for a whole region is needed. Some of these cases may be for anti-aliasing or depth of field, and in general for all effects of Distributed Ray Tracing [CPC84].

SPPM solves this problem by reformulating the PPM algorithm. After every photon tracing pass, the photons are not added to the exact hit point, but to a point that is chosen randomly within a region of the hit point by Distributed Ray Tracing. This is the only change to the algorithm. Even with this change, the algorithm still stays consistent. The advantages of PPM compared to PM are still available as before, while increasing the quality in most scenes.

Compared to PPM, SPPM reduces the noise in difficult settings significantly. Especially on objects, where the variations of the photon contributions can be significant, SPPM produces marginally better results. In Figure 4.6, a comparison of PPM (left) and SPPM (right) can be seen. The glossy floor in this Cornell box is marginally less noisy in the scene that uses SPPM within the same rendering time.



PPM SPPM

Figure 4.6: The comparison of a scene that was rendered with PPM to a scene that was rendered with SPPM. The scene that was rendered with PPM has a lot of noise on the glossy floor. Source: [HJ09]

CHAPTER 5

Participating Media

Before it is possible to evaluate rendering methods that compute the behaviour of light in a medium, it is important to know which properties define media. This chapter covers the basics of participating media. The theory in this chapter includes the evaluation of the properties of a medium 5.1 for emission, absorption, out-scattering and in-scattering and how these effects can be evaluated in applications. Furthermore, the differences between homogeneous and heterogeneous media, as well as single and multiple scattering are explained. The observations that are made in this chapter apply to homogeneous media if not mentioned otherwise. In the end of this chapter, the application of these theories to the tracing and scattering of photons is discussed in a general form.

5.1 Participating Media Properties

The main properties of a medium are separated into four different areas. These areas are emission, absorption, out-scattering and in-scattering. All of these properties are defined for one unit that is used in the scene. A visual representation of these properties is shown in Figure 5.1.

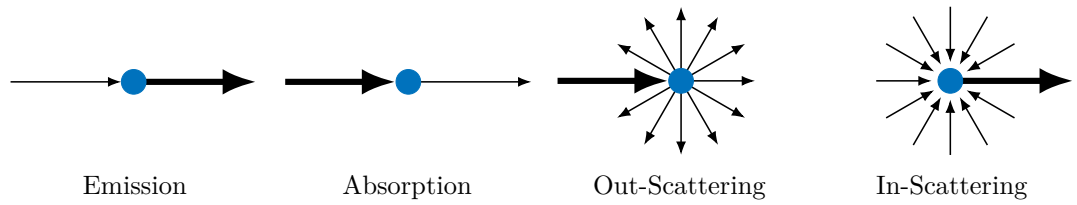


Figure 5.1: The four properties of participating media.

5.1.1 Emission

The emissive property defines the amount of light that is emitted by the participating media per unit and is defined by $\varepsilon(x)$, the emittance of light at a point x. In most cases this property is not used, as there are not many media that emit light. In the following chapters, this property is in most cases ignored, as the algorithms for the computation of participating media focus on media that does not emit light. One example of a medium that emits light is fire.

5.1.2 Absorption

How much light a participating media absorbs, can be expressed with the absorption property. This property is defined with the absorption coefficient $\sigma_a(x)$. If the absorption coefficient has a value of 0.5 that means that fifty percent of the light is lost after the light travels the length of one unit distance.

5.1.3 Out-Scattering

The property of out-scattering defines how much light scatters into other directions and does not continue the path of the light ray. This property is also called the scattering coefficient $\sigma_s(x)$.

The absorption coefficient $\sigma_a(x)$ and the scattering coefficient $\sigma_s(x)$ can be combined to the extinction coefficient $\sigma_t(x) = \sigma_a(x) + \sigma_s(x)$. This simplification can be made, as both coefficient represent a loss of power of the light either because of absorption or because of scattering into other direction. For the calculation of the power of the light ray after a certain distance, this still means that both of these properties act the same. The calculation of the loss of light over a distance t can be done with the transmission function

$$T_r(x, x_t) = e^{-t\sigma_t(x)}. \tag{5.1}$$

Some of the most important properties of the transmission are, that the results of the calculation always have to be in the interval $[0, 1]$. Furthermore, the transmission from a point to itself is always 1.

5.1.4 In-Scattering

The last of these four properties is the amount of light that is added to the power of the light from other directions. This property is called in-scattering and is very closely related to the out-scattering property. While a certain amount of energy is lost due to out-scattering at the current point, the light that is scattered at other points is added to the current position.

5.1.4.1 Phase Functions

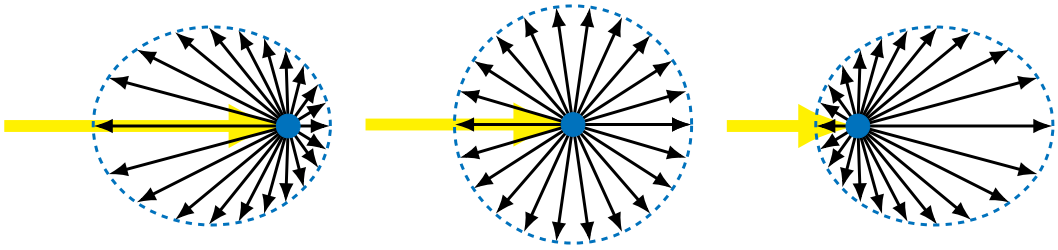
The distribution of in-scattering from different directions can be evaluated with phase functions. The mathematical notation of a phase functions is

$$p(x, \vec{\omega}, \vec{\omega}'), \tag{5.2}$$

which describes the distribution of the scattering of light at a point x for an incoming direction $\vec{\omega}'$ into a direction $\vec{\omega}$. Phase functions have two important properties:

1. Helmholtz-Reciprocity: The ray direction can be reversed. This property can also be seen for the BRDF.
$$p(x, \vec{\omega}, \vec{\omega}') = p(x, \vec{\omega}', \vec{\omega})$$
2. Normalization: Over all directions of a sphere, the phase function always has an integral of 1.
$$\int_{4\pi} p(x, \vec{\omega}, \vec{\omega}')\, d\vec{\omega}' = 1$$

The most important phase functions are evaluated in the following paragraphs.



Backward (g < 0) Isotropic (g = 0) Forward (g > 0)

Figure 5.2: The possibilities of scattering in a participating media.

Isotropic Phase Function

The easiest case is, that light scatters into this position from all directions evenly. In this case it is called isotropic scattering and can be evaluated with an isotropic phase function which is always

$$p(x, \vec{\omega}, \vec{\omega}') = \frac{1}{4\pi}. \tag{5.3}$$

The isotropic phase function is therefore a constant and a visual representation of it can be seen in Figure 5.2 (middle).

Henyey-Greenstein Phase Function

The Henyey-Greenstein Phase Function [HG41] is one of the most well-known phase functions. Compared to the isotropic case, the distribution of the scattering may not be even. In anisotropic scattering events, the main distinction is made between forward and backward scattering. Both of these scattering behaviours is shown in Figure 5.2 (left and right).

The Henyey-Greenstein phase function approaches this behaviour with the introduction of a variable $g \in [-1, 1]$. A g-value smaller than 0 defines backward scattering, while a g-value greater tahn 0 defines forward scattering. The Henyey-Greenstein phase function can be calculated with

$$p(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\ \cos\theta)^{1.5}}. \tag{5.4}$$

Not only can this phase function be used in the anisotropic case, but for a g-value of 0 it results in $\frac{1}{4\pi}$, the result of the isotropic phase function.

This phase function is used in the implementation of the Virtual Ray Lights implementation in Chapter 7.

Schlick Phase Function

The Henyey-Greenstein phase function would be ideal if it were not for the high computation time that is needed due to the power of 1.5 in the denominator of the second multiplicand. Therefore the Schlick Phase Function [BSS93] is needed to still get the advantages of the Henyey-Greenstein phase function without having to calculate the power of 1.5 of a term.

For the Schlick phase function the additional value k is introduced. The k-value can be calculated with

$$k = 1.55g - 0.55g^3, \tag{5.5}$$

where the g-value that was introduced for the Henyey-Greenstein phase function is remapped. The properties of the g-value for the definition of backward scattering and forward scattering still hold true. Although this simplification of the Henyey-Greenstein phase function may provide acceptable results in some cases, it does not provide good approximations in anisotropic media.

The calculation of the Schlick phase function can be done with

$$p(\theta) = \frac{1}{4\pi} \frac{1 - k^2}{(1 + k\ \cos\theta)^2}. \tag{5.6}$$

Other Phase Functions

There are many different phase functions. Some of the most important phase functions are the Legendre Polynomials [AA92], the Mie-Debye Phase Function [FW59], the Rayleigh Phase Function [Buc95] and the δ-Two-Term Approximation [JWW76]. The explanation of these phase functions is not in the scope of this thesis.

5.2 Radiative Transfer Equation

After evaluation of the four main properties of participating media, it is possible to formulate an equation to combine these attributes. This equation is called the Radiative Transfer Equation [Cha13] and can be calculated with

$$(\vec{\omega} \cdot \nabla)L(x,\vec{\omega}) = -\underbrace{\sigma_a(x)L(x,\vec{\omega})}_{\text{absorption}} - \underbrace{\sigma_s(x)L(x,\vec{\omega})}_{\text{out-scattering}} + \underbrace{\varepsilon(x)}_{\text{emission}} + \underbrace{\sigma_s(x)\int_{4\pi} p(x,\vec{\omega},\vec{\omega}')L(x,\vec{\omega}')d\vec{\omega}'}_{\text{in-scattering}}. \tag{5.7}$$

The radiative transfer equation calculates the change in the power of light at a point x and a direction $\vec{\omega}$. The first and the second term of the equation is the amount of light that is subtracted from the overall amount due to absorption $\sigma_a(x)$ and out-scattering $\sigma_s(x)$ at this point x. These two properties can be combined to the extinction coefficient $\sigma_t(x)$, as it is mentioned before. The third term adds the emission $\varepsilon(x)$ at this point x. The fourth and final term adds the amount of in-scattered light. All directions are evaluated and the light from every direction $L(x,\vec{\omega}')$ is multiplied with the phase function $p(x,\vec{\omega},\vec{\omega}')$. The result of the integral over the sphere is then multiplied with the scattering coefficient $\sigma_s(x)$ at this point.

5.3 Volume Rendering Equation

The radiative transfer equation only evaluates the power of the light at a point. To calculate the power of a light ray after it has traveled through a participating medium the Volume Rendering Equation [DCH88] is needed, which can be seen here:

$$L(x,\vec{\omega}) = \int_0^s \underbrace{T_r(x,x_t)}_{\text{extinction}} \underbrace{\sigma_s(x_t)\int_{4\pi} p(x,\vec{\omega},\vec{\omega}')L(x_t,\vec{\omega}')d\vec{\omega}'dt}_{\text{in-scattering}} + \underbrace{T_r(x,x_s)}_{\text{extinction}} L(x_s,\vec{\omega}) \tag{5.8}$$

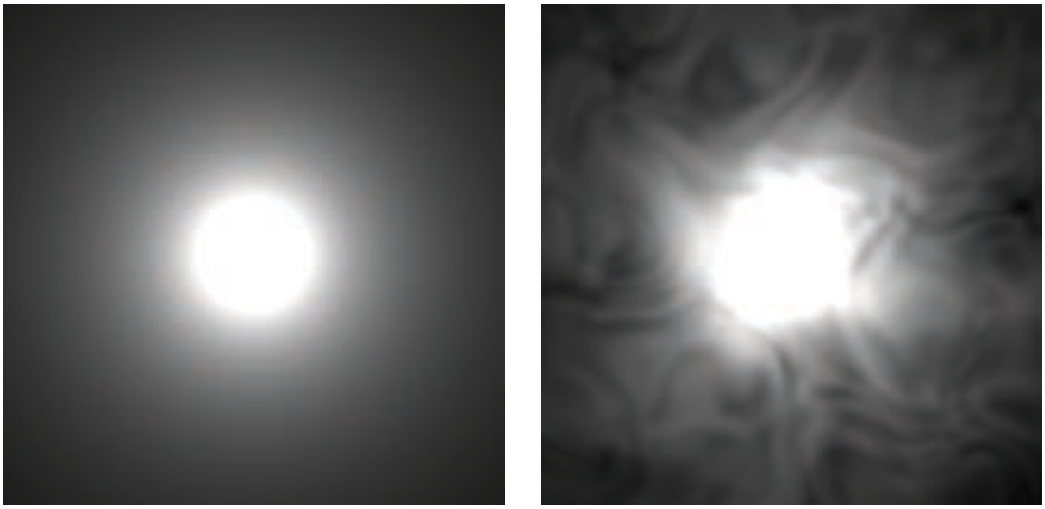
In this equation the distance t is the distance the ray has traveled to reach the point $x_t = x_s + t\vec{\omega}$. The ray itself has a distance of s and has its starting point in x_s. This means that the integral from s to 0 evaluates the ray over its entire distance. The light that is extinct over the entire distance of the ray is calculated with $T_r(x,x_s)$, therefore, at every evaluation step in the integral from s to 0, the amount of light that is extinct up to this point $T_r(x,x_t)$ has to be added again. The term for the in-scattered light is exactly the same as before.

As with the original rendering equation, the volume rendering equation cannot be solved in a closed form. The volume rendering equation is even more complex than the rendering equation. For every point in a medium the radiance depends on the radiance of all other points in the medium, as well as the surface radiance of all points in the scene.

This leads to the interesting fact that the volume rendering equation is hard and costly to solve. Therefore, many different approaches have been developed to solve this equation as efficiently as possible. A few of these approaches are discussed in Chapter 6.

5.4 Homogeneous and Heterogeneous Media

In the previous sections one important simplification is made. The media that is evaluated has always constant properties, that do not change inside the medium. This means that the media that are evaluated until now are all assumed to be homogeneous. In the real world there are not only homogeneous media, but also heterogeneous media. In a heterogeneous medium the properties of the previously discussed attributes can change. For example, this means the absorption coefficient of a heterogeneous medium is not a constant for all points in a medium, like it is in the homogeneous case, but it is different for every point in the medium. This statement is not only true for the absorption, but also for the emission, in-scattering and out-scattering. A comparison of homogeneous and heterogeneous media is shown in Figure 5.3.



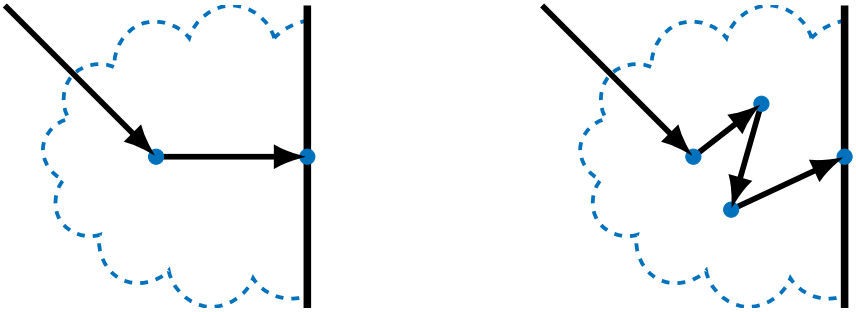
Homogeneous Media Heterogeneous Media

Figure 5.3: A comparison of homogeneous and heterogeneous participating media. Source: [Jar08]

This change in properties for heterogeneous media must be accounted for in many ways. The transmission, for example, cannot be simply evaluated as in the homogeneous case. For heterogeneous media, ray marching or a similar technique must be performed to evaluate the absorption and out-scattering properties at different step sizes, to make it possible to approximate the transmission. The problems that heterogeneous media incorporate are not only limited to transmission, but also to many other areas.

5.5 Single and Multiple Scattering

The properties for single and multiple scattering are an important factor, because the results can be more realistic, but the performance can decrease for multiple scattering events. The term single scattering means, that when light reaches a point in the medium, there were no other scattering events in the medium. The light ray may come from a light or a surface, but not a scattering event that was created due to the medium. As can be expected, multiple scattering means, that there have been previous scattering events inside the medium before the light reaches this point. A visual comparison of these tow terms can be seen in Figure 5.4.



Single Scattering Multiple Scattering

Figure 5.4: A visual comparison of single and multiple scattering.

Single and multiple scattering can produce very different results: While multiple scattering produces more realistic results, it is also a lot slower in computation. It is important to note, that the amount of multiple scattering has to be restricted. Even when changing from single scattering to tow scattering events, the number of samples needed for convergence increases marginally. Therefore, a threshold after which no more scattering events can occur must be chosen carefully.

5.6 Photon Scattering

For scenes with participating media, the algorithm for the tracing of photons must be adapted accordingly. The core of the algorithm as it is described in Section 4.1 stays the same, only a few adaptions in the TracePhoton-function must be made. As can be seen in Algorithm 5.1, the computations for the starting position, the staring direction and the power of the photon stay the same. Although it is not mentioned here specifically, the data structure for a photon and the storing of these data does not have to be adapted in any form or way. These structures can be used exactly as they are described in Section 4.1.

The important changes start at the beginning of the TracePhoton-function (Algorithm 5.2). The important change in this function is, that scattering event can also occur at any position within the participating medium and not only at surfaces.

Algorithm 5.1: FillPhotonMap()

```
1 for every light in the scene do
2     for the amount of photons for this light do
3         p = ComputePhotonStartingPoint(light)
4         d⃗ = ComputePhotonStartingDirection(light)
5         power = light.power /(amount of photons for this light)
6         TracePhoton(o, d⃗, power, 0)
7     end
8 end
```

For simplification purposes, the medium is assumed to be in the whole scene, as the distinction that have to be made otherwise are simple, but decrease the readability.

Algorithm 5.2: TracePhoton($o, \vec{d}$, power, depth)

```
1  dist_ff = − ln(ξ/σ_t)
2  intersection = IntersectScene(o, d⃗)
3  if intersection.dist ≤ dist_ff then
4      if no intersection then return
5      Algorithm 3.3 Lines 3 - 18
6      Algorithm 4.2 Lines 2 - 8
7  else
8      prob = σ_a/σ_t
9      if ξ < prob then return
10     p_scatter = o + d⃗ · dist_ff
11     pdf_scatter = exp(−σ_t · dist_ff)
12     power = power ·(Transmission(dist_ff)/ pdf_scatter)
13     d⃗ = ComputeScatteringDirection(p_scatter, d⃗)
14     TracePhoton(p_scatter, d⃗, power, depth +1)
15 end
```

To determine the position of a scattering event in a participating medium, the transmission function has to be reformulated to generate a distance for a uniform random value. This reformulation can be done with

$$d = -\ln\left(\frac{\xi}{\sigma_t}\right), \tag{5.9}$$

where the distance d is the length that a ray travels in a medium dependent on the uniform random value $\xi \in [0, 1)$ and the extinction coefficient σ_t. The corresponding PDF can be easily evaluated with

$$\text{pdf} = e^{-\sigma_t d}. \tag{5.10}$$

An introduction into importance sampling which includes explanations for PDFs can be found in Appendix A.

The implementation of this algorithm is shown in the first line of the TracePhoton-function. The important distinction that must be made is, if a scattering event in the medium occurs before the ray would hit a surface. If there is no scattering event in the participating medium before the ray reaches a surface, the algorithm equals exactly the prior version. If a scattering event occurs before the ray hits a surface, a part of the volume rendering equation must be evaluated.

An evaluation has to be made if the ray should be absorbed at this point or scattered into another direction. The probability of an absorption can be calculated by $\frac{\sigma_a}{\sigma_t}$. If the decision is made to continue, the point of the scattering event and the PDF for the distance, that was already mentioned before, has to be computed. The power of the photon must be adjusted by the transmission that affected the photon on the way to this point and the PDF.

The new direction of the ray depends on the scattering properties of the medium. For an isotropic medium, the distribution is even, so a new scattering direction can be sampled over a sphere. In an anisotropic medium, the direction must be evaluated by sampling the phase function at this point. With the direction of the ray, the photon can be traced further trough the scene until the depth value reaches the predefined threshold. The depth value can influence the performance even more than in a scene without participating media. Therefore, the choice could be made to create different depth values and thresholds for participating media scattering events and surface scattering events.

CHAPTER 6

Volumetric Photon & Virtual Light Techniques

The rendering of participating media is a task that many approaches try to solve efficiently. Approaches that use photon tracing provide some of the most promising results, therefore these algorithms are discussed in detail. The basis for this is a solid understanding of the rendering equation and volume rendering equation which were discussed in the previous chapter.

The first method that is evaluated in this thesis is Volumetric Photon Mapping (VPM) [JC98]. This method is a simple extension of Photon Mapping that serves as the base for most of the approaches that follow. Compared to PM, VPM is able to trace photons trough participating media and store the positions of the photons inside the media. Apart from that change, VPM is very similar to PM. An important extension of VPM is the Beam Radiance Estimate (BRE) by Jarosz et al. [JZJ08]. The BRE gathers photons along a beam. This means the noise and the speed of convergence can be significantly reduced.

Another method to use photon tracing is to create light sources at the positions that were traced. This method is called Virtual Point Lights (VPL) [Kel97] as it creates point light sources at the positions that would be photons. The illumination is calculated for every point light independently and the results are combined for the final image. Due to singularities that occur because of the intensity of the light sources, Virtual Spherical Lights (VSL) [HKWB09] are created. These spherical lights blur the point lights over an area to distribute the intensities of the point lights more evenly.

The next development of these algorithms is not to evaluate photons just at a single point, but over a beam [Nov14, NGHJ18]. Jarosz et al. [JNSJ11] introduce the concept of photon beams that use information that was gathered during the photon tracing process already, but was discarded until now. This means that the density of the data increases with this representation of photons as beams. Furthermore, Jarosz et al. classify nine different radiance estimators. These estimators include different techniques that were already created at that time and different options of using photon beams. The close connection between these nine different estimators is of major importance for the research that followed. Progressive Photon Beams (PPB) [JNT+11] improve this method by making it possible to render photon beams progressively by adjusting the radius of the photon beam at every pass. Although PPB provides great results in many scenarios, some other methods are superior in different scene settings. Unifying Points, Beams and Paths (UPBP) [KGH+14] creates a multiple importance sampling (MIS) strategy to combine different radiance estimators with an unbiased path integral estimator like Bidirectional Path Tracing (BPT) to achieve the best results in the least amount of time.

As photons can be turned into point lights, the same principle can be applied to photon beams. Virtual Ray Lights (VRL) [NNDJ12b] create a light ray that is geometrically similar to a photon beam, but has the same properties as a point light over its length. For the calculation of the radiance along a camera ray, single points along the light ray are sampled and evaluated. This can be seen as evaluating the contribution of one point light to the camera ray. As there are still singularities visible due to the intensities of the light rays, Virtual Beam Lights (VBL) were introduced by Novák et al. [NNDJ12a]. VBL extend VRL in the same way that VSL extend VPL, by blurring the light ray over an area to distribute the radiance.

As in the progression of most of these methods, the increase in dimensionality has been an important factor, Bitterli and Jarosz [BJ17] introduce photon planes and volumes. In their work, they prove that the representation of photons as beams instead of points decreases the variance of the results, and that the representation of photons in an even higher dimensionality further reduces the variance.

6.1 Volumetric Photon Mapping

6.1.1 Original Approach

Although, the introduction to PM focuses only on scenes without participating media, the algorithm can easily be adapted [JC98, JCKS02] and is called Volumetric Photon Mapping (VPM). How photons can be traced in a scene with participating media was already discussed. The photon gathering process is harder than in PM, as the photons are not only stored on surfaces, but also within the medium. An efficient data structure, like a balanced kd-tree, is necessary to reduce the gathering time to a minimum. Additional to these changes, the radiance estimation must be adapted to participating media. The in-scattered light L_i for photons that are stored in a participating media is calculated with

$$\begin{aligned} L_i(x,\vec{\omega}) &= \int_{\Omega_{4\pi}} f_s(\theta) L(x,\vec{\omega}')\, d\vec{\omega}' \\ &\approx \frac{1}{\sigma_s(x)\frac{4}{3}\pi r^3} \sum_{i=1}^{N} f_s(\theta_i)\Delta\Phi_i(x,\vec{\omega}_i). \end{aligned} \tag{6.1}$$

The extinction and emission properties are already taken care of. The power of the photon at a position in the medium is already adjusted by these properties, which makes the evaluation on the in-scattered light the only thing that must be calculated. The radiance estimation for photons on surfaces, which is shown in Equation 4.1, must be adapted by the scattering coefficient $\sigma_s(x)$ and the fact that the evaluation must take place within a sphere and not a disc. The whole term $\frac{4}{3}\pi r^3$ is the volume of the sphere, compared to the term πr^2 which is the area of the disc.

Only one further adaptation is necessary to make the rendering of participating media within a PM framework possible. The rays that were previously cast from the camera must now be evaluated at step sizes to gather the radiance of the medium. This process is called ray marching and can be expressed as

$$L(x,\vec{\omega}) \approx T_r(x,x_s)L(x_s,\vec{\omega}) + \left(\sum_{t=0}^{S-1} T_r(x,x_t)\sigma_s(x_t)L_i(x_t,\vec{\omega})\Delta t\right). \tag{6.2}$$

In this equation $T_r(x,x_s)$ is the transmission for the entire ray which adjusts only the radiance from a surface point $L(x_s,\vec{\omega})$. The sum that is added to this radiance is the sum of all the sample points $x_0,...,x_s$ which is again adjusted by the transmission to this point $T_r(x,x_t)$ and in this case also the scattering coefficient at this point $\sigma_s(x_t)$. The step size that is taken between the sample points is expressed as Δt. The in-scattered radiance into these sample points $L_i(x_t,\vec{\omega})$ as explained in Equation 6.1 is the most expensive part of this calculation.

The disadvantages of the PM algorithm still exist for this variant. There are multiple other shortcomings due to ray marching that are discussed in the next sections as different algorithms try to solve them.

6.1.2 The Beam Radiance Estimate

The previously discussed approach for VPM has a few flaws that can mainly be contributed to ray marching. The step size in ray marching is very important, since step sizes that are too large lead to noise and a step size that is too small leads to performance increases. The Beam Radiance Estimate (BRE) [JZJ08] solves these problems by reformulating the photon gathering technique. With ray marching, the photons are gathered at multiple steps, in contrast to the BRE, where photons are gathered over a ray. This can be expressed with

$$L(x, \vec{\omega}) = \int_0^s T_r(x, x_t)\sigma_s(x_t)L_i(x_t\vec{\omega})\, dt. \tag{6.3}$$

Not only solves this the problem of choosing an appropriate step size, it additionally solves the problem of gathering photons more than once. In ray marching, it is possible that the sampled regions overlap and include the same photons multiple times.

As the gathering of photons over a ray, as mentioned in Equation 6.3, is not realistically possible as the probability of a photon lying directly on a ray is zero, the ray must be expanded to a beam, which can be done with

$$\begin{aligned} L(x, \vec{\omega}) = \int_{\mathbb{R}} \int_0^{2\pi} \int_{\mathbb{R}} \int_{\Omega_{4\pi}} & K(t, \theta, r)T_r(x, x')\sigma_s(x') \\ & f_s(\theta')L(x', \vec{\omega}_t)\, d\vec{\omega}_t\, dr\, d\theta\, dt. \end{aligned} \tag{6.4}$$

This equation may seem very complex at first, but can be easily explained, as the description of a point by its cylindrical coordinates t, θ, r around the ray is the reason for the increased number of variables. The most important change is made by adding the smooth kernel $K(t, \theta, r)$. With this addition bias is introduced.

To be able to compute this equation it can be reformulated to

$$\frac{1}{N}\sum_{i=1}^{N} K(t_i, \theta_i, r_i)T_r(x, x_i)\sigma_s(x_i)f_s(\theta_i)\alpha_i. \tag{6.5}$$

This is simply an approximation of Equation 6.4. The new parameter α_i is the weight of the point $x_i = (t_i, \theta_i, r_i)$.

Additionally, the size of the volume which is surrounding the ray can be fixed or adaptive. The adaptiveness of the beam volume is essential to produce the correct results in some scenes.

The advantages of the BRE are noticeable in the speed of the convergence and the noise of the results. A comparison can be seen in Figure 6.1. On the left is the reference of this scene, in the middle the scene that was rendered with VPM and on the right the scene which was rendered with the BRE technique. Both versions were rendered within the same render time. The images produces by the BRE are noise-free, while the images that were created with the VPM still have a significant amount of noise in them.

Reference 02:38:57 VPM 00:04:21 BRE 00:04:15

Figure 6.1: A comparison of VPM and the BRE to a reference solution. The reference solution was rendered with VPM and a small step size. The render time is provided in hours:minutes:seconds. Source: [JZJ08]

6.2 Virtual Point Lights

6.2.1 Original Approach

A similar method to PM was introduced in 1997 by Alexander Keller [Kel97]. Compared to PM, in the tracing pass, a point light is created instead of a photon. This method can be explained as rendering the scene with just one of the point lights at a time. The sum of these images is the final result of the computation. The radiance at a point can therefore be calculated with

$$L(x,\vec{\omega}) \approx L_e(x,\vec{\omega}) + \sum_{i=1}^{N} V(x_i)G(x_i)L_i(x_i,\vec{\omega}_i). \tag{6.6}$$

The term $L_e(x,\vec{\omega})$ represents the radiance that reaches this point from a non-virtual light source. Added to this is the sum of all point lights that are not occluded. The terms $V(x)$ and $G(x)$ are the visibility and geometric terms at a point x that define the occlusion of this point light. The in-scattered radiance can be calculated with

$$L_i(x,\vec{\omega}) \approx \frac{\Phi f_s(\theta_p) f_s(\theta_u) T_r(w)}{w^2}. \tag{6.7}$$

The term Φ is the power of the light source, which has to be adjusted by the phase function $f_s(\theta_p)$ at this point to get the intensity. The other phase function at the location x_u represents the scattering at the position of evaluation. The term w is the distance of the point light at x_u from the position of evaluation. Therefore, $T_r(w)$ is the transmission along this distance.

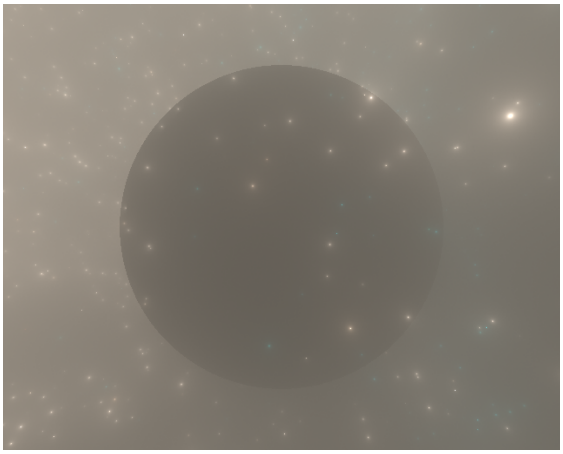


Figure 6.2: A scene rendered with VPL. The singularities in the medium can be seen as there are not enough point lights to illuminate the scene evenly. Source: [NNDJ12b]

Some of the disadvantages of VPL have a noticeable effect on the outcome of some scenes. Due to the nature of this method, local singularities with a high intensity can occur, as the distance ω can get arbitrarily small. These singularities are shown in Figure 6.2. Additionally, hard shadows can be displayed at places where there should not be any. These issue can be resolved by either clamping or blurring, as described in the next section.

6.2.2 Virtual Spherical Lights

The singularities that occur when rendering with VPL must be accounted for. These spikes in illumination are caused by the sampling of specific positions of the point light. Therefore, an idea to solve this problem is to not evaluate single points, but spheres instead [HKWB09]. This means that the single point light is blurred into spherical light. A visual comparison of these ideas is shown in Figure 6.3.

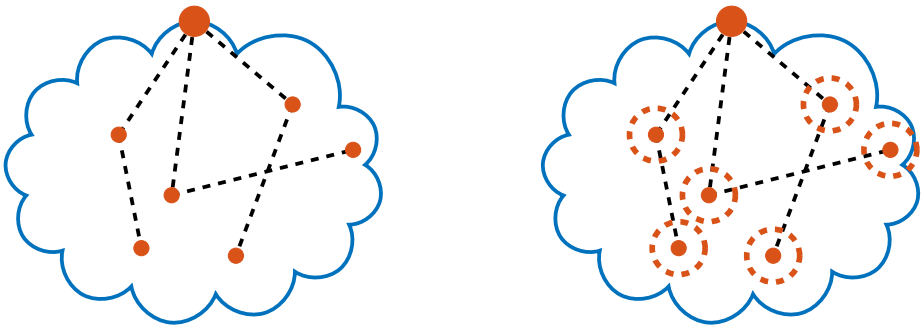


Figure 6.3: A visual comparison of the ideas behind VPL and VSL.

All points within a sphere, share the same properties that the origin of the sphere, the actual point light, has. This means that, even for the calculation of normals, the BRDF and phase functions, only the information from the original point light is used.

The size of the sphere depends on the density of the point lights that were created. In the impossible case that an infinite number of point lights is created, the radius of the spherical lights converges to zero. It is therefore a consistent algorithm, in the same sense that PM is a consistent algorithm. This approach is only consistent with an infinite number of spherical lights. This means that consistency can only be achieved in theory. Another disadvantage is, that this approach is biased due to blurring.

6.2.3 Iterative Importance Sampling of Virtual Point Lights

There are multiple additions to the VPL algorithm which focus on the distribution of the point lights by importance sampling [WBS03], [SIMP06], [SIP07]. One of the simpler, but effective ideas was introduced in 2010 by Georgiev et al. [GS10]. This approach makes use of the fact that the initial tracing of the point lights is computationally inexpensive. This preprocessing step is extended with just little overhead.

The main problem with the distribution of the point lights in the original approach, is that most of the created point lights only have no or just a very low contribution to the overall scene. Therefore, the calculation of the illumination of these point lights does not add much additional information, while still using viable rendering time. This make it sufficient to extend the tracing step of this approach to just sample the point lights that are vital for the computation of the scene.

The best way to distribute the point lights would be to sample them according to their contribution to every pixel that is computed individually. As this would defeat the efficiency of computational coherence, the image is seen as a just one pixel and the contribution of the point lights to this one-pixel image is used in the computation of the sampling probability. The importance of a point light can simply be estimated by the total image contribution. This makes it possible to calculate a probability of acceptance for a point light.

6.3 Photon Beams

6.3.1 Original Approach

One of the main contributions that led to the development of many of the approaches that follow, was made by Jarosz et al. [JNSJ11]. In this work, the connection between multiple approaches that already existed at the time is made. Furthermore, a new data representation which is called photon beam is introduced.

Previous to this approach, photons were only stored as points. Although, this makes them relatively easy to evaluate, a lot of information is lost, that was already available during tracing. The idea is now to use beams instead of points to store the information that is gathered during the tracing step.

Query	Data	Blur Possibilities
Point	Point	3D
Beam	Point	3D, 2D
Point	Beam	3D, 2D
Beam	Beam	3D, $2D_1$, $2D_2$, 1D

Table 6.1: The summary of different radiance estimates. Four different groups can be distinguished, where there are different blur possibilities for every one of these groups.

With the additional information that is stored within the beams, the density of the data in the scene is higher. This means that in areas where data was sparse previously, the beams produce additional information sources. This phenomenon can be seen in Figure 6.4, where the original VPM approach does not produce photons inside the sphere radius. In contrast, with the new Photon Beams (PB) method, there are two beams in the area that can be used as information sources. The number of traces does not change, while this approach still increases the density of the data in the scene.

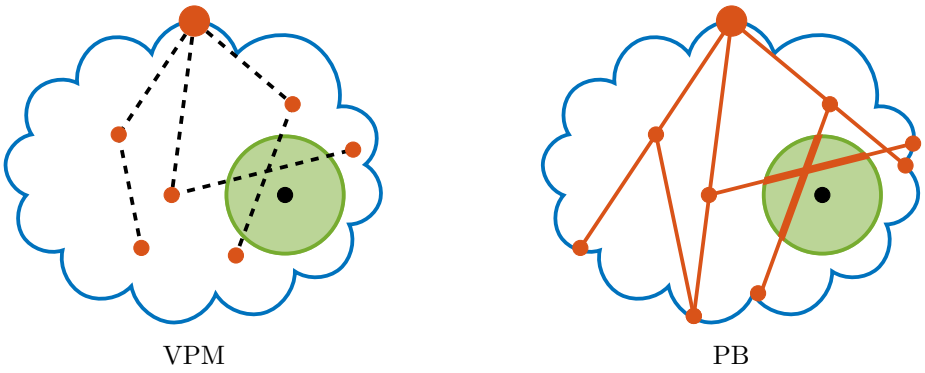


Figure 6.4: A visual comparison of the ideas behind VPM and PB.

Another contribution of Jarosz et al. is to summarize different representations into nine different radiance estimates (Table 6.1). Some of these radiance estimates have already been developed in publications, but this is the first time they are connected methodologically. The first row in Table 6.1 shows the radiance estimate for a point query and point data with a 3D blur. This is exactly the description of VPM as it was discussed previously.

For a beam query and point data, there are two blurring possibilities. The method for a 2D blur was already evaluated as it describes the BRE. The radiance estimator that blurs in 3D would also be possible. For the next group, the beam representation of photons is used. The group of estimators that use beam data, can be separated into estimators that evaluate the radiance at a point, or at a beam. For estimators that only evaluate the radiance at a point, a 3D blur can be seen as the evaluation of a sphere around this point. For a 2D blur, the same principle is applied with a disk.

The last group consists of radiance estimators for photon beams that evaluate the radiance at a beam segment. The version with a 3D blur is only of theoretical relevance, as it involves the integral of a 3D convolution which is hard to solve for different conditions. The versions with a 2D or 1D blur are far more practical. As is shown in Table 6.1, there are two versions of a 2D blur. Both of these versions are very similar. The first 2D blur evaluates the photon beam as continuous for the in-scattered radiance at a point. Therefore, the integral over these points leads to the final radiance estimate. In contrast to that, the second 2D blur starts with the BRE for different points on the photon beam. The 1D blur is a simplification of both of these approaches. In this case the power of the photon beam is blurred into only one direction which depends on the direction of the camera ray and the photon beam.

The mathematical formulation of the approach with a 1D blur can be done with

$$L_m(x, \vec{w}, r) \approx \Phi k_r(u)\sigma_s(x_w)T_r(w)T_r(v)\frac{f_s(\vec{w}\cdot\vec{v})}{\sin(\vec{w},\vec{v})}, \tag{6.8}$$

where the term k_r is the 1D blur kernel that is centered on the ray. In this equation, u, v and w are the scalars along the three axes to the point closest to the camera ray.

The results of rendering with PB can be seen in Figure 6.5 which is a recreation of the lighthouse scene from Jarosz et al. [JZJ08]. The approach that is used in the right image is the radiance estimate for a beam query and a photon beam with a 1D blur. The image in the middle is rendered with the BRE and the image to the left is the reference solution. The PB method shows clear improvements with the same tracing steps and similar render time. Compared to the reference solution, there is still some low frequency noise visible.

6.3.2 Progressive Photon Beams

The PB approach that was introduced by Jarosz et al. [JNSJ11] shares a few disadvantages with the original PM approach and some related methods. The main problem is, that it needs an infinite amount of beams to be unbiased. For an infinite amount of beams the blur kernel gets infinitely small which therefore eliminates the bias. This means that an error is introduced by the blur kernels that must be eliminated.

Progressive Photon Beams (PPB) [JNT+11] solves these problems with the implementation of PB into a progressive framework as it is used for Progressive Photon Mapping (PPM). This algorithm starts from the beam x beam variant with a 1D blur. For the convergence of an algorithm with progressive updates, the average variance and the expected value (bias) must converge to zero for an infinite number of passes. This can be achieved for this method by adjusting the radius of the blur kernel appropriately. There are two different properties for the size of the blur radius: One is the global scaling factor and the other one a minimum and maximum value for every pass. The convergence can be achieved by creating a specific ratio of the radius between passes.

Reference

BRE 00:00:31

PB BxB 1D 00:00:25

Figure 6.5: A comparison of BRE and the PB in the beam x beam version and a 1D blur, to a reference solution. The render time is provided in hours:minutes:seconds. Source: [JNSJ11]

This ratio and the calculation of the radius to achieve this ratio is done with

$$\frac{R_{i+1}}{R_i} = \frac{\mathrm{Var}[\epsilon_i]}{\mathrm{Var}[\epsilon_{i+1}]} = \frac{i+\alpha}{i+1}. \tag{6.9}$$

The variance $\mathrm{Var}[\epsilon]$ of the error at a pass ϵ_i has to be evaluated with the variance of the next pass $i+1$. This equals the inverted ratio of the radius of the next pass R_{i+1} to the radius of this pass R_i. To evaluate this for convergence, these ratios must be equal to the term $\frac{i+\alpha}{i+1}$ which includes the user defined variable $\alpha \in [0,1]$. The actual radius can be derived from

$$R_i = \left(\prod_{k=1}^{i-1} \frac{k+\alpha}{k} \right) \frac{1}{i}. \tag{6.10}$$

With this radius adjustment per pass, the bias vanishes for an infinite number of passes, too. Therefore, the properties for a progressive algorithm are satisfied. The value α, that can be defined by the user, influences how much the radius changes after every pass. The higher this value is, the smaller is the reduction of the radius per pass.

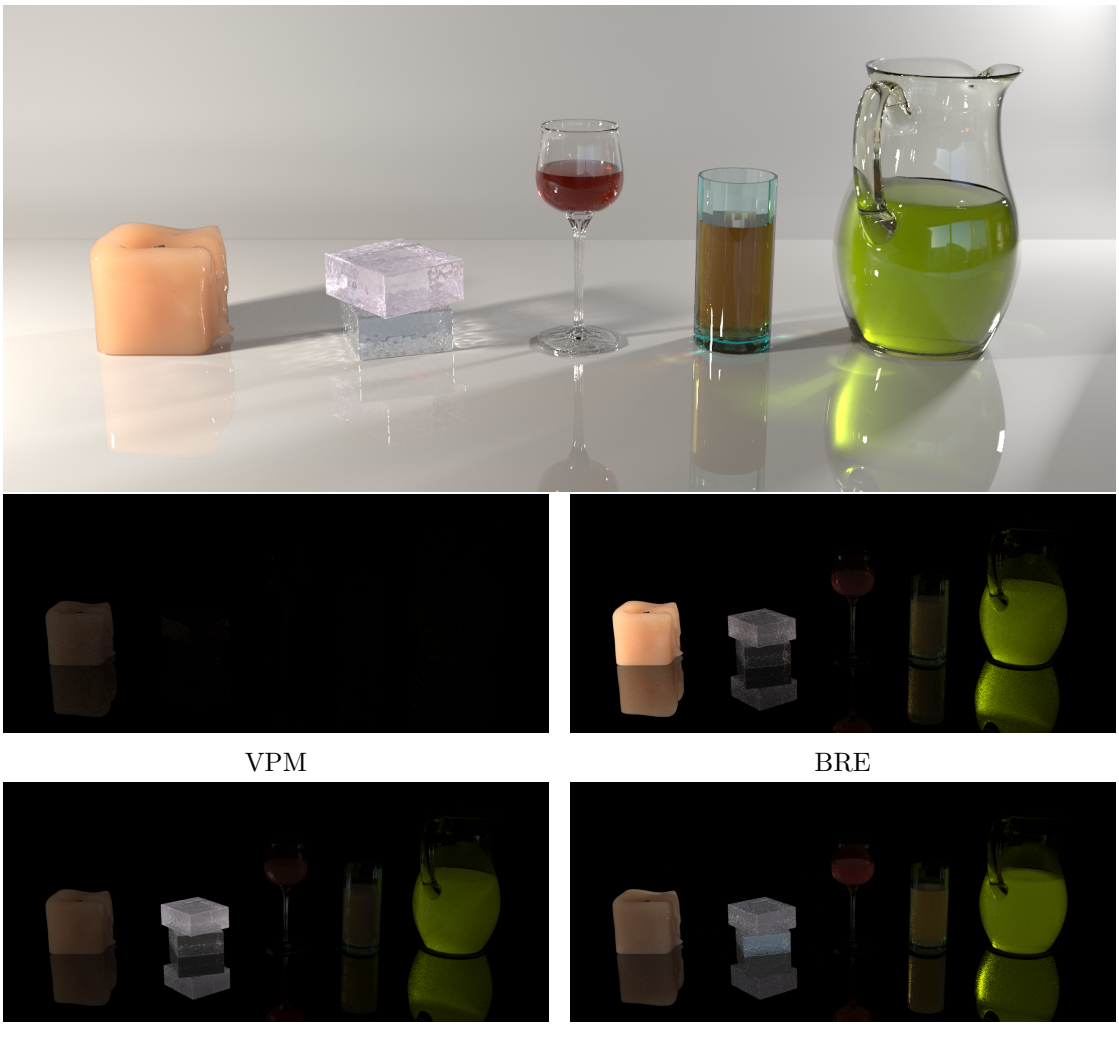


Figure 6.6: A comparison of UPBP, VPM, BRE, PB and BPT. The images for VPM, BRE, PB and BPT are weighted by their contribution to the overall scene. Source: [KGH+14]

6.3.3 Unifying Points, Beams and Paths

There are many approaches explained by Jarosz et al. [JNSJ11] in their summary of the nine radiance estimates. Although it is suggested to use the beam x beam approach with a 1D blur, this may not be the best approach for every scenario. As discussed in Unifying Points, Beams and Paths (UPBP) [KGH+14, Vév15], the suggested approach to use the beam x beam 1D blurring radiance estimator may not be the best choice. In dense media, beams actually lead to results that are more noisy than results from photons that were evaluated as points. Conversely, photons that are evaluated as points produce noisier results in thin media, compared to beams.

UPBP chooses from three different radiance estimators, for which a MIS strategy is created to combine these estimators with an unbiased path integral estimators like BPT . This strategy evaluates the efficiency of every estimator and chooses an appropriate one. These three radiance estimators are the point x point estimator with a 3D blur (VPM), the beam x point estimator with a 2D blur (BRE) and the beam x beam estimator with a 1D blur (PB). The results of each of theses estimators, and the combination of them with UPBP is shown in Figure 6.6.

6.4 Virtual Ray and Beam Lights

6.4.1 Virtual Ray Lights

The contributions that were made to not just consider the point locations of photons, but an entire beam, lead to Virtual Ray Lights (VRL) [NNDJ12b]. This approach is discussed in detail in the next chapters, but a short explanation is provided here for completeness of the evaluation of the approaches. This approach combines the beam estimations with VPL. This means that this time, instead of photon beams, simple light rays are created, that act similar to the point lights of the VPL approach. A comparison of VRL and VPL can be seen in Figure 6.7.

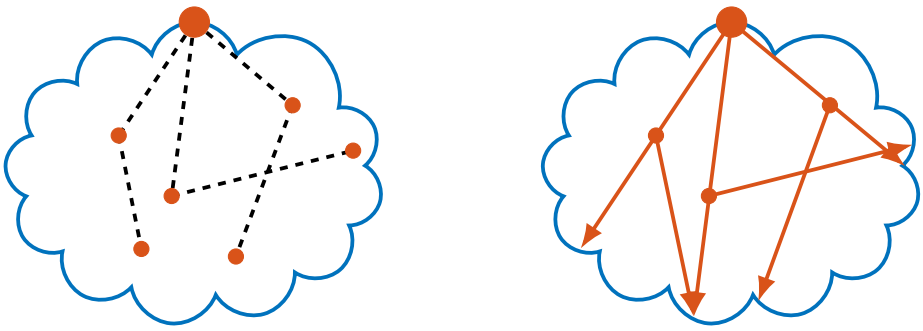


Figure 6.7: A visual comparison of the ideas behind VPL and VRL.

One of the disadvantages of the VPL method is that singularities are created, as the single point light creates a high intensity over a small area. By distributing the energy along a light ray, the singularities can be significantly reduced. The radiance at a point can be calculated with

$$L(x, \vec{\omega}) = T_r(s) L_s(x_s, \vec{\omega}) + L_m(x, \vec{\omega}). \tag{6.11}$$

The calculation of the surface radiance L_s and the radiance from the medium L_m can be achieved with

$$L_s \approx \Phi \int_0^t \frac{\sigma_s(v) f_s(\theta_v) f_r T_r(w_u(v)) T_r(v) V_u(v)}{w_u(v)^2} \, dv, \tag{6.12}$$

and

$$L_m \approx \Phi \int_0^s \int_0^t \frac{\sigma_s(u) \sigma_s(v) f_s(\theta_u) f_s(\theta_v) T_r(u) T_r(v) T_r(w) V}{w(u, v)^2} \, dv \, du, \tag{6.13}$$

respectively.

A visual explanation of the terms in these equations is shown in Figure 6.8.

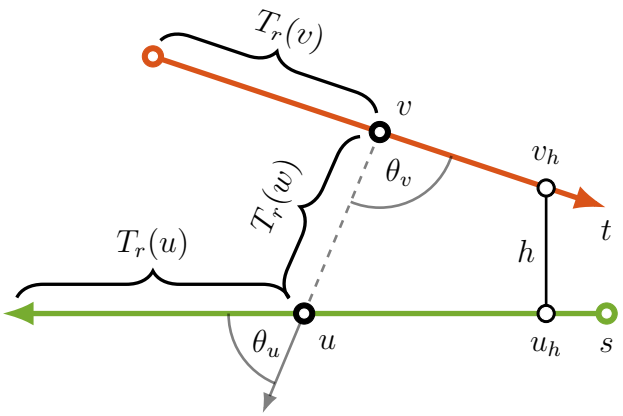


Figure 6.8: A visualization of the terms of Equation 6.13. The green ray is the camera ray that has the length s. The orange ray is the light ray with the length t. In this case, the contribution of the point light at the position v on the light ray is evaluated for the position u on the camera ray.

Another contribution of Novák et al. is a product importance sampling technique to integrate over the camera and light ray. This makes it possible to handle difficult cases, i.e., anisotropic media, robustly. The importance sampling can be viewed as first sampling a point on the light ray and calculating the contribution of this point to the camera ray. The chosen points can be interpreted as point lights. Therefore, this technique works as a final gather over the light ray. Another great benefit from VRL is that it is an unbiased method. This means that the rendering with progressive updates can be trivially implemented.

VPL PPB VRL

Figure 6.9: A comparison of VPL, PPB and VRL. All of these images were created within the same render time (600 seconds). Source: [NNDJ12b]

Although, VRL creates images with significantly less noise than VPL, other approaches are still superior in specific areas: The creation of volume caustics is better created with PPB and surface caustics are better implemented with PPM. For volume to volume and volume to surface events VRL outperforms these approaches. A comparison between VPL, PPB and VRL is shown in Figure 6.9. All of these images were created in 600 seconds. As can be seen in this figure, the image with VRL has a lot less noise compared to VPL and additionally produces acceptable results faster than PPB.

6.4.2 Virtual Beam Lights

VRL already reduce the occurrences of singularities significantly, but some singularities are still visible. This is due to the fact that intensities along a light ray are still very bright compared to the surrounding area. This property is especially visible at glossy surfaces and the contribution from a volume to a surface. In these cases, only a single integral is used for the computation which takes away the element of the final gathering that is possible for volume to volume events. A way to solve this problem is to blur the light ray into a light beam [NNDJ12a]. The expansion from a light ray to a light beam is very similar to the expansion from a point light to a spherical light. In both cases the data structure to gather the data from is expanded to eliminate occurring singularities. As with the conversion from point light to spherical light, the expansion from light ray to light beam introduces bias. The progressiveness of this method can still be maintained, as the radius of the beam can be reduced for passes as it has been explained for PPB. In some cases the introduction of bias may be an acceptable trade off, as VBL produce acceptable images significantly faster than VRL.

A comparison of VRL and VBL can be seen in Figure 6.10. The singularities that occur on the glossy surfaces of the image that was created with VRL vanish in the image that was created with VBL. Furthermore, the beams of light that are clearly visible when rendering with VRL are not visible anymore in VBL due to blurring.

VRL

VBL

Figure 6.10: A comparison of VRL and VBL. Both of these images were created within the same render time (186 seconds for volume to volume events, 363 seconds for volume to surface events). Source: [NNDJ12a]

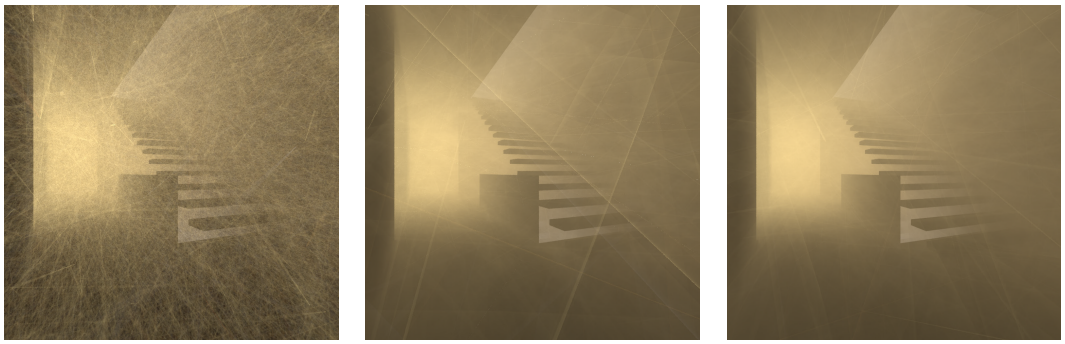
6.4.3 Joint Importance Sampling

Similar to the case of VPL, Georgiev et al. [GKH+13] propose a method the improves the scattering of the light in a scene for light paths, which is called Joint Importance Sampling (JIS). The idea is that previous methods scatter the light according to local scattering events without taking into account the global view of the path. This leads to significant variance which can be reduced with this new approach.

One contribution is the formulation of multiple techniques as a general method with different connection strategies. It is therefore possible to show the advancements of this technique on various approaches, as the sampling of longer subpaths has been suboptimal in most cases. Furthermore, especially the results in anisotropic media are improved tremendously. As this is a highly complex technique, a further evaluation is omitted due to the scope of this thesis.

6.5 Higher-Dimensional Photon Representations

The progression of most of the approaches has been to increase the dimension of the data to be able to avoid singularities and noise. Therefore, photon points are expanded to beams. There is also an idea to create higher dimensional data [BJ17] such as photon planes and volumes. Although the increase in dimension for the representations of the photon data does not always lead to an improvement of the results. In cases where beams are already superior to points, the further increase in dimensionality further increases these advantages. A comparison of different representations with different dimensionalities is shown in Figure 6.11. As the dimensionality of the representation increases, the variance reduces. Especially, the changes from photon beams to photon planes are significant.



PB PP PV

Figure 6.11: A comparison of photon beams, photon planes and photon volumes. All of these images were created within the same render time. Source: [BJ17]

CHAPTER 7

Virtual Ray Lights

Many approaches that were developed prior to Virtual Ray Lights (VRL) [NNDJ12b] lead to the development of this method. As the Beam Radiance Estimate advanced the gathering of photons at points, to gathering them along a line, the Progressive Photon Beams extend the photon data to beam data. Therefore, radiance estimators for beam queries and beam data were created. VRL extends these principles into the context of virtual lights.

Virtual Point Lights had been developed years ago and some of the issues that were surfaced were still not solved properly. As the VPL create point lights with high intensities, these point lights lead to singularities, that must be handled. One way of doing this, is by blurring the point light to create a spherical light. Unfortunately, this process introduces bias. As this unsatisfactory situation had to be solved, Novák et al. [NNDJ12b] decided to solve this problem by developing Virtual Ray Lights.

7.1 Methodology

The representation of the light data is one of the main changes in VRL compared to VPL. As can be seen in Figure 7.1, the entire path segments are stored compared to just the point data. Therefore, the data is distributed more evenly and the singularities can be eliminated inside the medium.

The radiance at a point can be calculated with

$$L(x, \vec{\omega}) = T_r(s)L_s(x_s, \vec{\omega}) + L_m(x, \vec{\omega}). \tag{7.1}$$

The terms L_s and L_m in Equation 7.1 are the terms for the surface radiance and the radiance from the medium. The calculation of these terms is shown in Equation 7.2 and Equation 7.3, respectively.

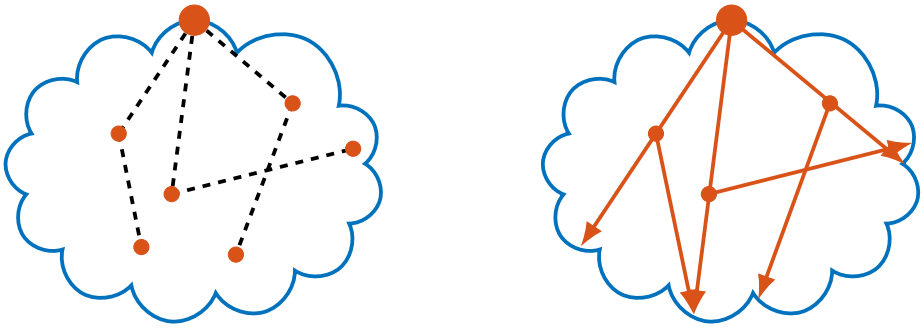


Figure 7.1: A visual comparison of the ideas behind VPL and VRL.

The surface radiance

$$L_s \approx \Phi \int_0^t \frac{\sigma_s(v) f_s(\theta_v) f_r T_r(w_u(v)) T_r(v) V_u(v)}{w_u(v)^2}\, dv, \tag{7.2}$$

is an integral over the light ray, as the camera ray only has to be evaluated at its end point. This equation consists out of the following variables of Table 7.1.

Symbol	**Description**
v	a sampled point on the light ray
θ_v	the angle of the light ray and the direction from v to the surface point
Φ	the power of the light ray
$\sigma_s(v)$	the scattering coefficient at the position v
$f_s(\theta_v)$	the phase function of the angle θ_v
f_r	the BRDF at the surface point
$T_r(w_u(v))$	the transmission from the point v to the surface point
$T_r(v)$	the transmission from the start of the light ray to the point v
$V_u(v)$	the binary visibility from the point on the surface to the point v
$\frac{1}{w_u(v)^2}$	the inverse squared distance from the point v to the surface point

Table 7.1: The terms of the equation for the surface radiance.

The medium radiance

$$L_m \approx \Phi \int_0^s \int_0^t \frac{\sigma_s(u)\sigma_s(v) f_s(\theta_u) f_s(\theta_v) T_r(u) T_r(v) T_r(w) V}{w(u,v)^2}\, dv\, du, \tag{7.3}$$

is a double integral over the light ray and the camera ray, as in this situation, the camera ray has to be evaluated over its entire length. This equation consists out of the terms described in Table 7.2.

A visual explanation of the terms in these equations can be seen in Figure 7.2. The only terminology that is not explained yet are the symbol h which is the closest distance between both rays and the symbols v_h and u_h which are the closest points along each ray.

Symbol	Description
v	a sampled point on the light ray
u	a sampled point on the camera ray
θ_v	the angle of the light ray and the direction from v to u
θ_u	the angle of the camera ray and the direction from v to u
Φ	the power of the light ray
$\sigma_s(u)$	the scattering coefficient at the position of the point u
$\sigma_s(v)$	the scattering coefficient at the position of the point v
$f_s(\theta_u)$	the phase function of the angle θ_u
$f_s(\theta_v)$	the phase function of the angle θ_v
$T_r(u)$	the transmission from the start of the camera ray to the point u
$T_r(v)$	the transmission from the start of the light ray to the point v
$T_r(w)$	the transmission between the point v to the point u
V	the binary visibility from the point v to the point u
$\frac{1}{w(u,v)^2}$	the inverse squared distance from the point v to the point u

Table 7.2: The terms of the equation for the radiance from the medium.

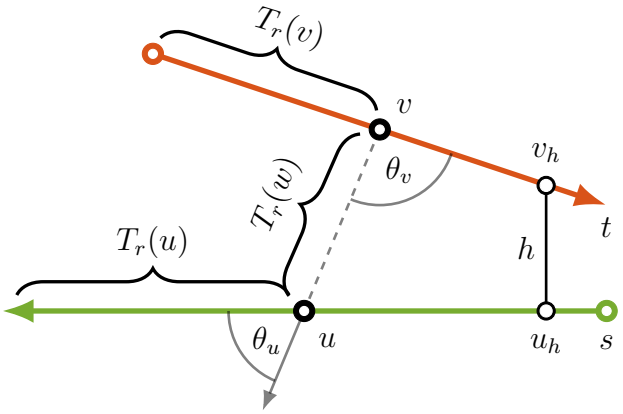


Figure 7.2: A visualization of the terms of Equation 7.3. The green ray is the camera ray that has the length s. The orange ray is the light ray with the length t. In this case, the contribution of the point light at the position v on the light ray is evaluated for the position u on the camera ray.

7.2 Algorithm

The basic concept for this method is to incorporate a process similar to photon tracing, as was discussed in Section 4.1. In this case, the data is stored as a ray, therefore the name Virtual Ray Lights, which are, for simplification, just called light rays in this thesis. For a camera ray, all contributions of all light rays that have been created have to be evaluated. Therefore, a pair of the camera ray and one light ray is formed, for which either the volume-to-surface contribution is evaluated, as was described in Equation 7.2 or the volume-to-volume contribution is computed, as was discussed with Equation 7.3. As these equations are not solvable in a closed form, an approach with a Monte Carlo integration has to be taken for the computation. A basic introduction into Monte Carlo intergration and importance sampling is included in Appendix A where these terms are explained with simple examples.

As for the volume-to-surface contribution only a single integral has to be solved, whereas for the volume-to-volume contribution, a double integral has to be solved, the volume-to-volume contribution is harder to compute. Therefore, a closer look will be taken into the approach for the volume-to-volume contribution, as the approach for the volume-to-surface contribution can be derived from it.

Appendix A already explains how to convert an integral into a Monte Carlo estimator in a simple case. In this more challenging case with a double integral, the procedure is similar. The integrand

$$g(u,v) = \frac{\Phi\sigma_s(u)\sigma_s(v)f_s(\theta_u)f_s(\theta_v)T_r(u)T_r(v)T_r(w)V}{w(u,v)^2} \tag{7.4}$$

of Equation 7.3 will now be simply called $g(u,v)$. This change in notation is in the interest of brevity.

In the unbiased Monte Carlo estimator

$$L_m \approx \frac{1}{N}\sum_{i=1}^{N}\frac{g(u_i,v_i)}{\mathrm{pdf}(u_i,v_i)}, \tag{7.5}$$

N denotes the number of samples, whereas the term u_i and v_i stand for the samples that are chosen along the camera ray and the light ray, respectively. As it is a double integral, two samples have to be chosen for the evaluation. The term $\mathrm{pdf}(u_i,v_i)$ is the probability of choosing theses two samples. Novák et al. therefore explain their sampling along a 2D-domain which is created by the two sampling distributions. The distribution of samples inside this 2D-domain is of major importance to the convergence, as graphical explanation will show in the next section and as the results in Chapter 9 will make clear. Simple approaches like uniform sampling and exponential sampling will, in most cases, not converge in a reasonable amount of time. Therefore, more complex methods for importance sampling have to be chosen.

In the volume-to-surface case, the behaviour can be simplified. As there is only one integral to evaluate, only a sample along the light ray has to be chosen. The PDF is therefore just the probability of choosing that sample and not a joint probability. As in the volume-to-volume case, the Monte Carlo estimator can be constructed with the integrand of the corresponding formula (Equation 7.2). The importance sampling method that are described in the following chapter can be used for the volume-to-volume as well as the volume-to-surface contribution by just adjusting the input to the sampling functions.

7.3 Importance Sampling Techniques

The techniques that are introduced for importance sampling are separated into a simple technique that works well in isotropic media and an advanced technique that works well in anisotropic media. Therefore, first the isotropic case will be discussed, to be able to expand this method for the anisotropic case.

7.3.1 Isotropic Sampling

In the isotropic case, the most variation is caused by the inverse squared distance term. Therefore the PDF

$$\mathrm{pdf}(u_i, v_i) \propto \frac{1}{w(u, v)} \tag{7.6}$$

has to be proportional to this term.

A sampling method that solved this problem effectively was already introduced by Kulla et al. [KF11]. In their method, the position on a ray is sample according to the position of a point light. The method can be seen as taking steps along the angle, therefore, this method is often called equi-angular sampling. A visualization of this method is shown in Figure 7.3.

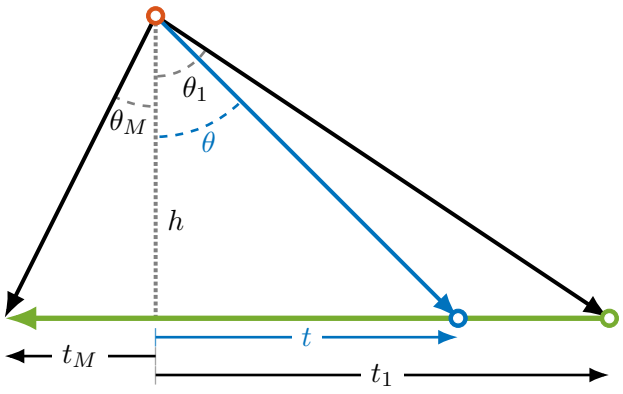


Figure 7.3: A visualization of the importance sampling by Kulla et al. [KF11]. The PDF of this technique is proportional to the inverse squared distance.

For this method, the closest point to the point light along the ray has to be found. With this point, the distances along the ray must be changed, as this point serves now as the new origin. Note that the distance t_1 and t_M can now be negative.

The PDF for a distance t can be calculated with

$$\mathrm{pdf}(t) = \frac{h}{(\theta_M - \theta_1)(h^2 + t^2)}, \tag{7.7}$$

where h is the distance of the closest point on the ray to the point light and θ_1 and θ_M are the angles to the start and end of the ray, respectively. The naming of these variables was changed in this evaluation compared to the original implementation to be consistent within this chapter and the next approach. The naming results from taking M samples along the angle which are distributed between the start θ_1 and the end θ_M in the advanced approach of the following section.

To get a sampled distance along this ray, the inverse transform sampling method

$$\mathrm{cdf}^{-1}(\xi) = h \tan\left((1 - \xi)\theta_1 + \xi\theta_M\right) \tag{7.8}$$

is used. The term ξ has to be a random number in the interval $[0, 1)$ which will be remapped to the desired distribution $\mathrm{pdf}(t)$ with this function.

The conversion from a distance along a ray to an angle is

$$\theta_x = \tan^{-1}\left(\frac{x}{h}\right), \tag{7.9}$$

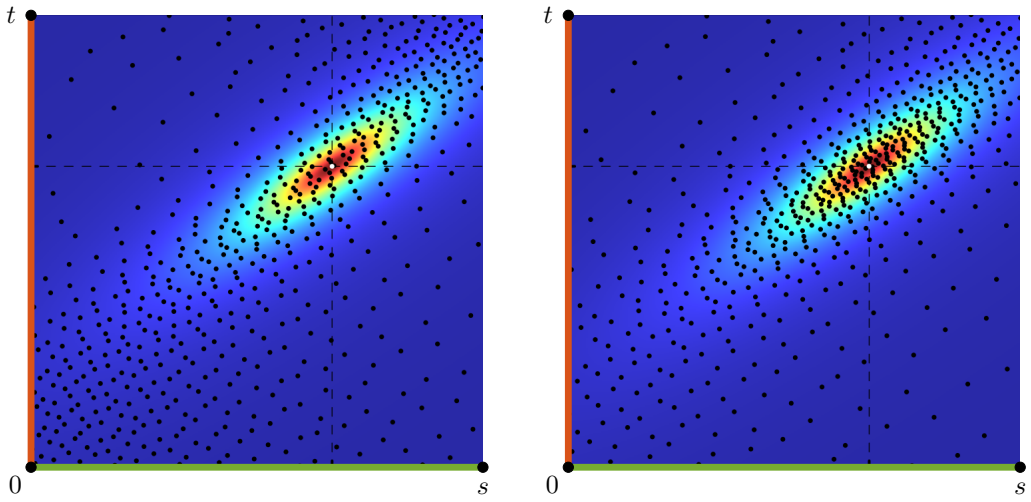
whereas the inverse function is

$$x = h \tan(\theta_x). \tag{7.10}$$

As the equi-angular sampling is not suited for lights other than point lights, the position on larger lights, like a light ray, has to be sampled to determine the whole influence of the light on the ray. By distributing these samples uniformly along the light ray, the samples are not distributed according to the distance to the camera ray and are therefore not ideally matched to the target distribution. The visualization from Novák et al. where the samples are distributed in the 2D domain this way can be seen in Figure 7.4 (left).

To improve the sampling of the position on the light ray, Novák et al. construct a joint distribution for both the light ray and the camera ray. In this joint distribution, the light ray is first sampled using a marginal PDF. This sample is then used for the conditional PDF along the camera ray. With this technique, the variation along the whole 2D domain can be accounted for.

For the marginal PDF along the light ray, the points with the closest distance to the other rays, u_h and v_h, as well as the distance between those points, h, has to be determined. With these values, a similar change in parameter, as with the equi-angular sampling can be applied. The distances to the samples u and v are now denoted as $\hat{u}$ and $\hat{v}$ where $\hat{u} = u - u_h$ and $\hat{v} = v - v_h$.



Uniform & Equi-Angular Sampling | Joint Distribution Sampling

Figure 7.4: A comparison between choosing a uniformly distributed sample on the light ray and an equi-angular sample on the camera ray (left), to choosing these samples with a joint distribution (right). Source: [NNDJ12b]

The same change along the light ray has to be applied to v_0 and v_1, the start and end points of the light ray. The marginal PDF for the light ray can then be calculated with

$$\mathrm{pdf}(\hat{v}, \hat{v}_0, \hat{v}_1) = \frac{\int_{\hat{u}_1}^{\hat{u}_0} w(\hat{u}, \hat{v}, h, \theta)^{-2} d\hat{u}}{\int_{\hat{v}_1}^{\hat{v}_0} \int_{\hat{u}_1}^{\hat{u}_0} w(\hat{u}, \hat{v}, h, \theta)^{-2}\, d\hat{u}\, d\hat{v}}. \tag{7.11}$$

The denominator of this equation serves just for the normalization of the PDF. With the help of the law of cosines, the term $w(\hat{u}, \hat{v}, h, \theta)^{-2}$ can be written as $h^2 + \hat{u}^2 + \hat{v}^2 - 2\hat{u}\hat{v}\cos\theta$ where $\cos\theta$ is created by the angle of the camera ray and the light ray. As there is currently no analytic solution for this equation, simplifications must be made. In this simplification, the camera ray is assumed to be infinite. With this the PDF can be solved as

$$\mathrm{pdf}(\hat{v}, \hat{v}_0, \hat{v}_1) = \frac{\sin(\theta)}{\sqrt{h^2 + \hat{v}^2 \sin^2(\theta)}(A(\hat{v}_1) - A(\hat{v}_0))}. \tag{7.12}$$

In this equation, the term A is used as $A(x) = \sinh^{-1}\left(\frac{x}{h}\sin(\theta)\right)$. The CDF of Equation 7.12 is obtained via integration

$$\mathrm{cdf}(\hat{v}, \hat{v}_0, \hat{v}_1) = \frac{A(\hat{v}_0) - A(\hat{v})}{A(\hat{v}_0) - A(\hat{v}_1)}. \tag{7.13}$$

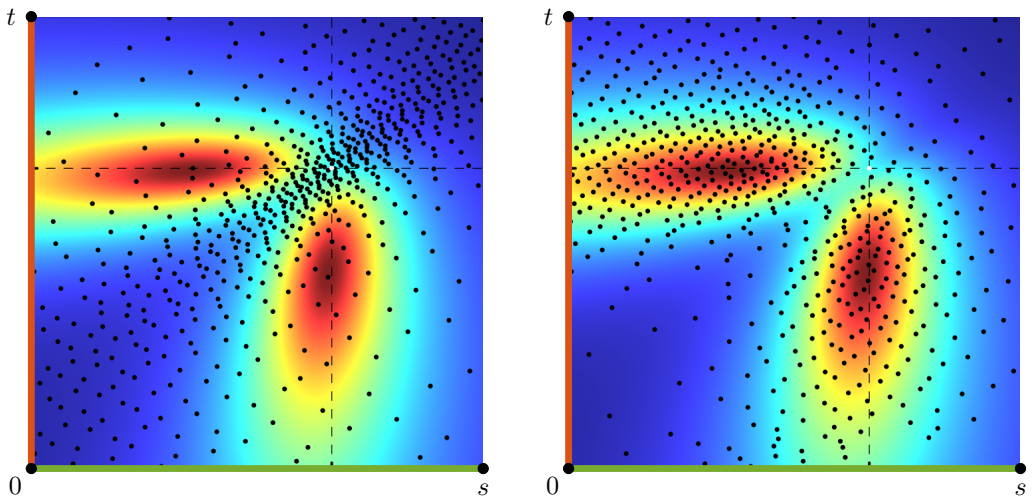
Finally, for the CDF^{-1}, the CDF has to be inverted, which results in

$$\mathrm{cdf}^{-1}(\xi, \hat{v}_0, \hat{v}_1) = \frac{h \sinh((1-\xi)A(\hat{v}_0) + \xi A(\hat{v}_1))}{\sin(\theta)}. \tag{7.14}$$

With Equation 7.14, which needs a random number $\xi \in [0, 1)$ as input, a point on the light ray can be sampled with this inverse transform sampling. This point can be seen as a point light on the light ray.

In the second step of this sampling method, a point on the camera ray has to be sampled according to the sampled position on the light ray with a conditional PDF. In the simple joint distribution, the conditional part is identical to the equi-angular sampling method. The results is shown in Figure 7.4 (right). Figure 7.4 therefore shows the difference between sampling the light ray uniformly (left) and sampling it with a specialized method that takes the camera ray into account (right). As is shown in this visualization, the samples match the target distribution better with the joint distribution approach. Chapter 9 provides a comparison of these two approaches with rendered results to show the difference in convergence. This approach for a joint distribution sampling is from now on mentioned as the simple joint distribution approach, as the next section advances this approach.

7.3.2 Anisotropic Sampling



Equi-Angular Sampling of Camera Ray Advanced Angular Sampling of Camera Ray

Figure 7.5: A comparison between the simple joint distribution (left), to the advanced approach (right). Source: [NNDJ12b]

For anisotropic media, the inverse squared distance is not the only term to consider for the PDF: The phase functions along the light ray and camera ray have a large influence on the variance. In Figure 7.5, the approach for a simple joint distribution can be seen to the left. This approach does not consider the phase functions and can therefore not account for the variance. The PDF that is now needed has to be proportional to the inverse squared distance, as in the isotropic case, but also to the product of both phase functions

$$\mathrm{pdf}(u_i, v_i) \propto \frac{f_s(u_i) f_s(v_i)}{w(u, v)}. \tag{7.15}$$

For the construction of the conditional PDF in the advanced case, the equi-angular sampling is taken as a base for the following calculations. As the equi-angular sampling can be seen as taking uniformly distributed steps along the angle, this advanced approach can be seen as taking the steps with a distribution proportional to the product of the phase functions. As this sampling approach works in the angular domain, the distance does not need to be considered while evaluating the phase functions. As is shown in Figure 7.6 (bottom), the product of the phase functions has to be calculated along the arc that is spanned by the projection of the camera ray onto the point that was sampled with the marginal PDF. Figure 7.6 (top) shows the unit sphere around the sampled point, as well as the projection of the camera ray onto the arc.

In Figure 7.6 (bottom left), the isotropic case is shown. Here, the result of the phase function at all points is constant. Compared to this, the anisotropic case, which can be seen in Figure 7.6 (bottom right), is a lot more complex, as the result of both phase functions, the one along the light ray and the one along the camera ray, varies for different positions. This variation is the main reason, why the sampling of the simple joint distribution does not lead to a sufficient result in the anisotropic case.

An illustration of the product of the phase functions along the arc in two different cases is shown in Figure 7.7. As can be seen in these examples, the green line, which is the result of the phase function along the camera ray, is always monotonic. Therefore, the main variation comes from the result of the phase function along the light ray. Figure 7.7 shows in both cases the peak θ_{peak} at which the result of the phase function along the light ray (orange line) reaches its highest point. The peak does not have to be in this interval, but for illustration purposes this is the case in both of these examples. As the phase functions cannot be calculated along the whole arc, samples have to be distributed to create a piecewise function which approximates the product of both phase functions. The samples taken can be seen as the blue points and the reconstructed function as the dotted black line connecting them.

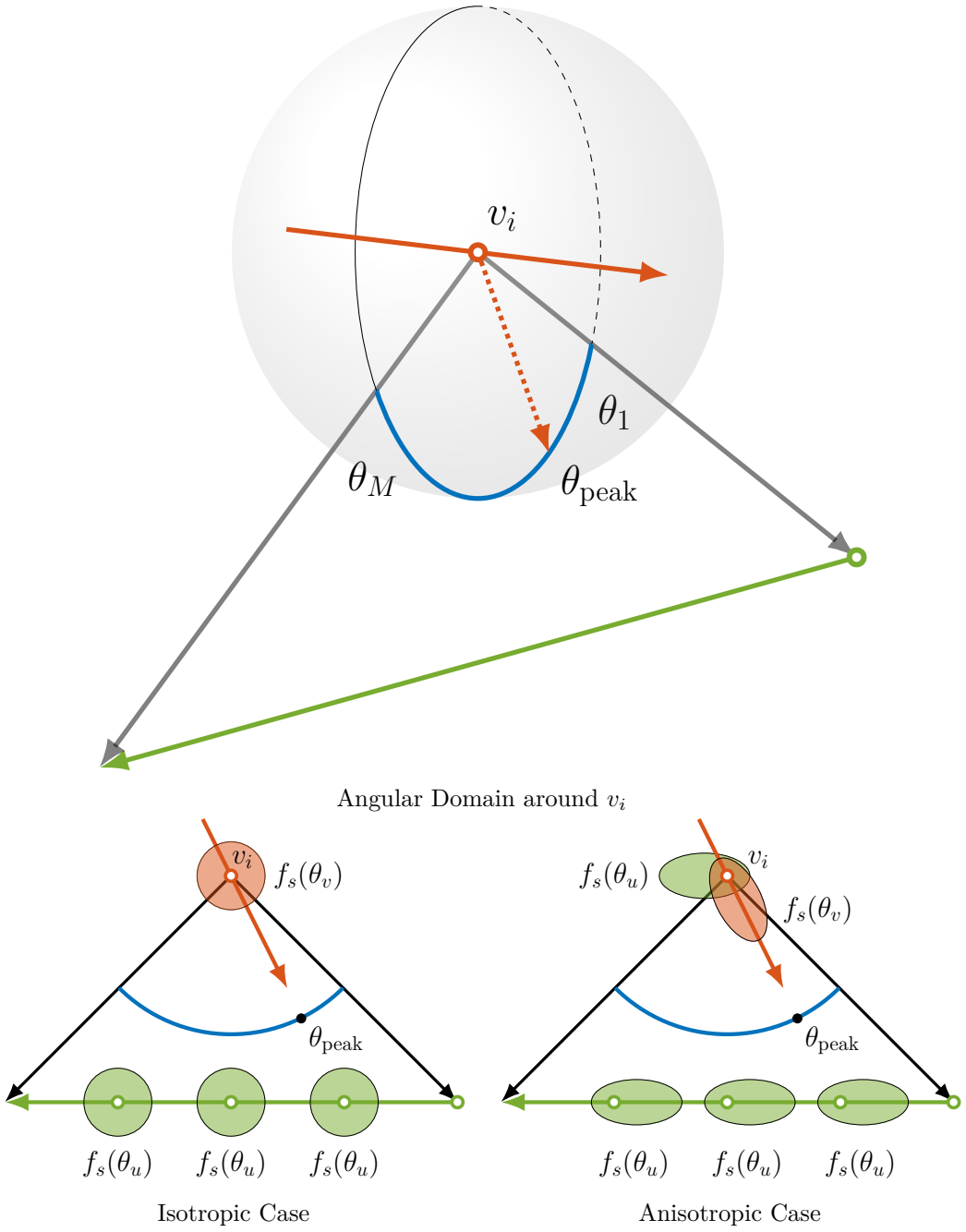


Figure 7.6: An illustration of the spherical domain around v_i as well as the arc that is generated by the projection of the camera ray (green) onto the sphere (top). Furthermore, the projection of the light ray into the angular domain can be seen in the same figure. The figures on the bottom show the isotropic (left) and anisotropic (right) cases. In the isotropic case, the product of the phase functions at θ_{peak} is equal to result at all other angles.

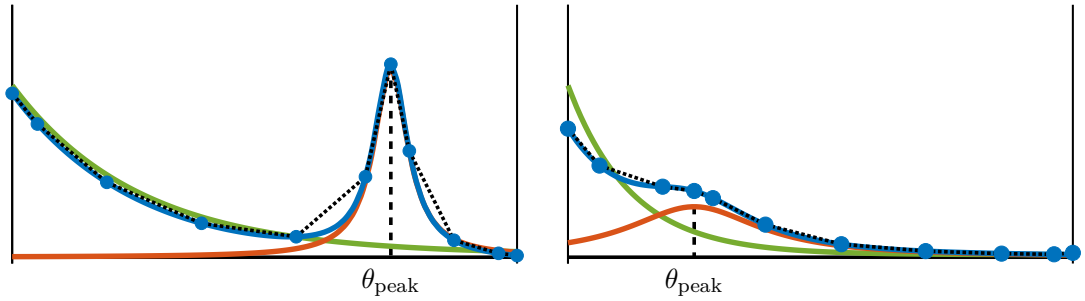


Figure 7.7: Two examples of the product of the phase functions (blue). The green line is the phase function along the camera ray and the orange line the phase function along the light ray.

7.3.2.1 Distribution of Samples Along the Arc

As is shown in Figure 7.7, especially in the example at the top, sampling the position of the peak is of major importance to faithfully reconstruct the function. The peak is mainly determined by the direction of the light ray. Therefore, the angle along the spherical arc can be calculated by the projection of the light ray onto the arc. A visual example of this is shown in Figure 7.6 (top) where the light ray (orange) is projected (dashed orange) onto the arc (blue). For this calculation, the directions $\vec{a}$ and $\vec{b}$, are the directions to the start and the end of the camera ray. The normal of the plane that can be constructed from $\vec{a}$ and $\vec{b}$ is noted as $\vec{c} = \text{norm}(\vec{a} \times \vec{b})$.

The projection of the light direction $\vec{d}$ onto the plane that is formed by $\vec{a}$ and $\vec{b}$ can be calculated with

$$\vec{e} = \text{norm}((\vec{c} \times \vec{d}) \times \vec{c}). \tag{7.16}$$

If the peak lies on the arc, $\vec{e}$ must lie between $\vec{a}$ and $\vec{b}$. If this is not the case, the negative peak might lie between these directions. To get the direction to the negative peak, $\vec{e}$ has to be inverted.

Before distributing the samples along the ray, the directions are change into angles. Uniformly to the equi-angular sampling, the point with the closest distance is used as the new center. The angles are measured from the direction of the sample along the light ray to the closest point on the camera ray. Figure 7.6 (top) shows that the angle to the origin of the camera ray is called θ_1, whereas the angle to the end of the ray is called θ_M. The M stands for the number of samples that are about to be distributed along the arc. The angle θ_{peak} is given if the peak or the negative peak lies within the arc.

For the distribution of samples along the arc, there are now two cases to consider, which both can be handled similarly. In the first case, neither the peak, nor the negative peak, lie on the arc. In this case the samples are distributed with a cosine-warped uniform spacing. The angle for the sample with the index $j \in [1, M]$ can be determined with

$$\theta_j = \frac{\theta_M - \theta_1}{2} \left(1 - \cos\left(\frac{\pi(j-1)}{M}\right)\right). \tag{7.17}$$

This equation distributes the samples closer to the boundary of the interval, as more variance is expected there.

For the second case, where the peak lies on the arc, first, the index of the sample with the peak has to be determined. The index

$$j_{\text{peak}} = \left\lfloor \frac{\theta_{\text{peak}} - \theta_1}{\theta_M - \theta_1} M - 0.5 \right\rceil \tag{7.18}$$

determines the index of the peak. The samples are then distributed as in the previous case, but within the two intervals $[\theta_1, \theta_{\text{peak}}]$ and $[\theta_{\text{peak}}, \theta_M]$. After this procedure, there are M θ distributed along the arc, where one θ lies on the peak, if the peak is on the arc. With the results of the phase functions at these positions, the phase functions along the arc can be reconstructed with a piecewise linear function.

7.3.2.2 Construction of the PDF for the Product of the Phase Functions

To construct the PDF of the product of the phase functions along the arc, the results of the phase functions at the sampled positions have to be calculated. In this case, θ_u is the angle between the camera ray and the direction from the sampled position on the light ray to the sampled position along the arc. With θ_u the phase function along the camera ray $f_s(\theta_u)$ can be calculated. Uniformly, θ_v is the angle between the direction of the light ray and the direction from the sampled position on the light ray to the sampled position along the arc, which can be used to calculate the phase function $f_s(\theta_v)$. The product of both phase functions at a sampled angle θ_j on the arc is from now on called $f_{\text{uv}}(\theta_j)$. At this point, all variables necessary for the construction of the piecewise PDF of the product of the phase functions are available.

It should be noted that the mathematical derivation of the PDF, the CDF and its inverse are contributions of this thesis that were derived from descriptions of Novák et al. from the original paper.

As this PDF is a piecewise linear function, the well-known slope-intercept form is used for each segment. The slope-intercept form is denoted as $y = kx + d$ where k is the slope of the line and d is the coordinate where the line crosses the y-axis. In our case, x is the θ and y is the product of the phase functions with this θ, $f_{\text{uv}}(\theta)$. The value for k_j can be calculated with

$$k_j = \frac{f_{\text{uv}}(\theta_{j+1}) - f_{\text{uv}}(\theta_j)}{\theta_{j+1} - \theta_j}, \tag{7.19}$$

whereas d_j can be calculated with

$$d_j = f_{\text{uv}}(\theta_{j+1}) - k_j\theta_{j+1} = f_{\text{uv}}(\theta_j) - k_j\theta_j, \tag{7.20}$$

for a segment with the θ_j at the start and θ_{j+1} at the end of the segment. Naturally, there is one segment less, than there are θ distributed along the arc, as one segment is always enclosed by two θ.

Repeating this procedure for every segment yields the piecewise linear PDF $\widetilde{\text{pdf}}_{f_{\text{uv}}}(\theta)$ for the product of the phase functions

$$\widetilde{\text{pdf}}_{f_{uv}}(\theta) = \begin{cases} k_1\theta + d_1, & \theta \in [\theta_1, \theta_2] \\ k_2\theta + d_2, & \theta \in (\theta_2, \theta_3] \\ \quad\vdots & \\ k_N\theta + d_N, & \theta \in (\theta_N, \theta_M] \end{cases}. \tag{7.21}$$

It is important to note, that $\widetilde{\text{pdf}}_{f_{\text{uv}}}(\theta)$ is not normalized.

For the normalization of $\widetilde{\text{pdf}}_{f_{\text{uv}}}(\theta)$ the CDF of it, $\widetilde{\text{cdf}}_{f_{\text{uv}}}(\theta)$, is needed. The calculation of this piecewise quadratic function can be done with

$$\widetilde{\text{cdf}}_{f_{\text{uv}}}(\theta) = \begin{cases} \displaystyle\int_{\theta_1}^{\theta_2} k_1\theta + d_1 \, d\theta, & \theta \in [\theta_1, \theta_2] \\ \widetilde{\text{cdf}}_{f_{\text{uv}}}(\theta_2) + \displaystyle\int_{\theta_2}^{\theta_3} k_2\theta + d_2 \, d\theta, & \theta \in (\theta_2, \theta_3] \\ \quad\vdots & \quad\vdots \\ \widetilde{\text{cdf}}_{f_{\text{uv}}}(\theta_N) + \displaystyle\int_{\theta_N}^{\theta_M} k_N\theta + d_N \, d\theta, & \theta \in (\theta_N, \theta_M] \end{cases}. \tag{7.22}$$

The CDF is just the integration over the PDF. One of the integrals at an index j is

$$\int_{\theta_j}^{\theta_{j+1}} k_j\theta + d_j \, d\theta = \left(\frac{k_j}{2}\theta_{j+1}^2 + d_j\theta_{j+1}\right) - \left(\frac{k_j}{2}\theta_j^2 + d_j\theta_j\right). \tag{7.23}$$

The normalization of the $\widetilde{\mathrm{pdf}}_{f_{\mathrm{uv}}}(\theta)$ can be achieved by dividing it by the $\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta)$ over the interval $[\theta_1, \theta_M]$. The $\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta)$ over this interval is from now on called $\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}$. The normalized $\widetilde{\mathrm{pdf}}_{f_{\mathrm{uv}}}(\theta)$, $\mathrm{pdf}_{f_{\mathrm{uv}}}(\theta)$, can therefore be created with

$$\mathrm{pdf}_{f_{\mathrm{uv}}}(\theta) = \begin{cases} \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}^{-1}(k_1\theta + d_1), & \theta \in [\theta_1, \theta_2] \\ \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}^{-1}(k_2\theta + d_2), & \theta \in (\theta_2, \theta_3] \\ \vdots & \vdots \\ \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}^{-1}(k_N\theta + d_N), & \theta \in (\theta_N, \theta_M] \end{cases}. \tag{7.24}$$

The corresponding CDF is

$$\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta) = \begin{cases} \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}^{-1} \int_{\theta_1}^{\theta_2} k_1\theta + d_1 \, d\theta, & \theta \in [\theta_1, \theta_2] \\ \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_2) + \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}^{-1} \int_{\theta_2}^{\theta_3} k_2\theta + d_2 \, d\theta, & \theta \in (\theta_2, \theta_3] \\ \vdots & \vdots \\ \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_N) + \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}^{-1} \int_{\theta_N}^{\theta_M} k_N\theta + d_N \, d\theta, & \theta \in (\theta_N, \theta_M] \end{cases}. \tag{7.25}$$

The last thing to do, is to inverse this CDF, to get the CDF^{-1} for the inverse transform sampling. For the piecewise CDF^{-1} of $\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta)$, $\mathrm{cdf}^{-1}_{f_{\mathrm{uv}}}(\xi)$, there are two cases to consider per segment. In the first case, the k_j with the current index j is 0, whereas in the second case, k_j is not equal to 0. For this procedure, the piecewise function

$$
\mathrm{cdf}^{-1}_{f_{\mathrm{uv}}}(\xi) =
$$

$$
\begin{cases}
\left(\theta_1 d_1 + \xi\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}} - \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_1)\right) d_1^{-1}, \\
\qquad\qquad k_1 = 0,\ \xi \in [\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_1), \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_2)] \\
\left(\sqrt{k_1(\theta_1^2 k_1 + 2\xi\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}} - 2\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_1)) + 2\theta_1 d_1 k_1 + d_1^2} - d_1\right) k_1^{-1}, \\
\qquad\qquad k_1 \neq 0,\ \xi \in [\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_1), \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_2)] \\
\left(\theta_2 d_2 + \xi\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}} - \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_2)\right) d_2^{-1}, \\
\qquad\qquad k_2 = 0,\ \xi \in [\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_2), \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_3)] \\
\left(\sqrt{k_2(\theta_2^2 k_2 + 2\xi\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}} - 2\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_2)) + 2\theta_2 d_2 k_2 + d_2^2} - d_2\right) k_2^{-1}, \\
\qquad\qquad k_2 \neq 0,\ \xi \in [\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_2), \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_3)] \\
\qquad\qquad\qquad \vdots \\
\left(\theta_N d_N + \xi\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}} - \widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_N)\right) d_N^{-1}, \\
\qquad\qquad k_N = 0,\ \xi \in [\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_N), \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_M)] \\
\left(\sqrt{k_N(\theta_N^2 k_N + 2\xi\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}} - 2\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_N)) + 2\theta_N d_N k_N + d_N^2} - d_N\right) k_N^{-1}, \\
\qquad\qquad k_N \neq 0,\ \xi \in [\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_N), \mathrm{cdf}_{f_{\mathrm{uv}}}(\theta_M)]
\end{cases}, \tag{7.26}
$$

takes a value ξ as input, which has to be in the interval $[0, 1)$. This function can produce an angle θ along the arc with a distribution according to $\mathrm{pdf}_{f_{\mathrm{uv}}}(\theta)$. As the sampled angle is now generated, the last step is to combine the PDF of the product of the phase functions, with the PDF of the inverse squared distance.

7.3.2.3 Creation of the final PDF

For the final PDF, the PDF for the product of the phase functions, $\mathrm{pdf}_{f_{\mathrm{uv}}}(\theta)$, as well as the PDF for the inverse squared distance is needed. As explained for the simple joint distribution, the PDF for the inverse squared distance is given by the equi-angular approach. The combination of both PDFs is

$$\widetilde{\mathrm{pdf}}(\theta) = \mathrm{pdf}_{f_{\mathrm{uv}}}(\theta)\,\mathrm{pdf}_{w^{-2}}(\theta). \tag{7.27}$$

As this $\widetilde{\mathrm{pdf}}(\theta)$ is not normalized yet, it has to be integrated to get the corresponding $\widetilde{\mathrm{cdf}}(\theta)$

$$\widetilde{\mathrm{cdf}}(\theta) = \frac{\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta)}{\widetilde{\mathrm{cdf}}_{f_{\mathrm{uv}}}(\theta_M - \theta_1)} = \frac{\mathrm{cdf}_{f_{\mathrm{uv}}}(\theta)}{\theta_M - \theta_1} \tag{7.28}$$

and then be divided by it. By dividing the $\widetilde{\mathrm{pdf}}(\theta)$ through the $\widetilde{\mathrm{cdf}}(\theta)$, a normalized PDF, $\mathrm{pdf}(\theta)$, is created

$$\mathrm{pdf}(\theta) = \frac{\widetilde{\mathrm{pdf}}_{f_{\mathrm{uv}}}(\theta)h}{h^2 + t^2}. \tag{7.29}$$

As can be seen from Equation7.29, for the $\mathrm{pdf}(\theta)$, the $\widetilde{\mathrm{pdf}}_{f_{\mathrm{uv}}}(\theta)$ can be used, as the normalization term for the product of the phase functions drops out in the normalization step of the combination of both PDFs. Similarly, the term $\theta_M - \theta_1$ in the denominator drops out for the PDF for the inverse squared distance. The results of sampling with this technique in the 2D domain is shown in Figure 7.5 (right). Compared to Figure 7.5 (left), the samples that are generated match the distribution of the underlying function marginally better. Therefore, the convergence is a lot faster, as can be seen by many examples in Chapter 9.

7.4 Discussion

Virtual Ray Lights is an unbiased method for volumetric lighting that improves previous virtual light methods light VPL in most cases. For the estimation of volume-to-volume radiance, VRL improves the convergence and the occurrence of singularities marginally, as can be seen in Figure 7.8 (right). This is the case as the media is sampled more densely with a ray, than with a single point. This leads to a clear improvement in the order and amount of singularities.

For the volume-to-surface radiance, this property of VRL is diminished, as when a light ray hits a surface, at the point of intersection, the energy of the ray is compressed into a small area. Although the order of the singularities is reduces compared to VPL, the amount of the singularities is not. This effect can be seen in Figure 7.8 (left).

In general, VRL provide great results for the volume-to-volume contribution and improve the volume-to-surface contribution. As the importance sampling is suited for isotropic and anisotropic media, this method yields rapid convergence, even with highly anisotropic media (Henyey-Greenstein $g = \pm 0.95$). While other techniques work well in isotropic media, the case of anisotropic media is often not evaluated as thoroughly as it is done for VRL.

For a visual comparison, Chapter 9 compares results with different scene settings, anisotropy and number of samples. In these results, the advantages and disadvantages of VRL are shown. Especially, the different importance sampling techniques are compared to other, simpler, importance sampling techniques. This shows the vast improvements that these techniques bring.

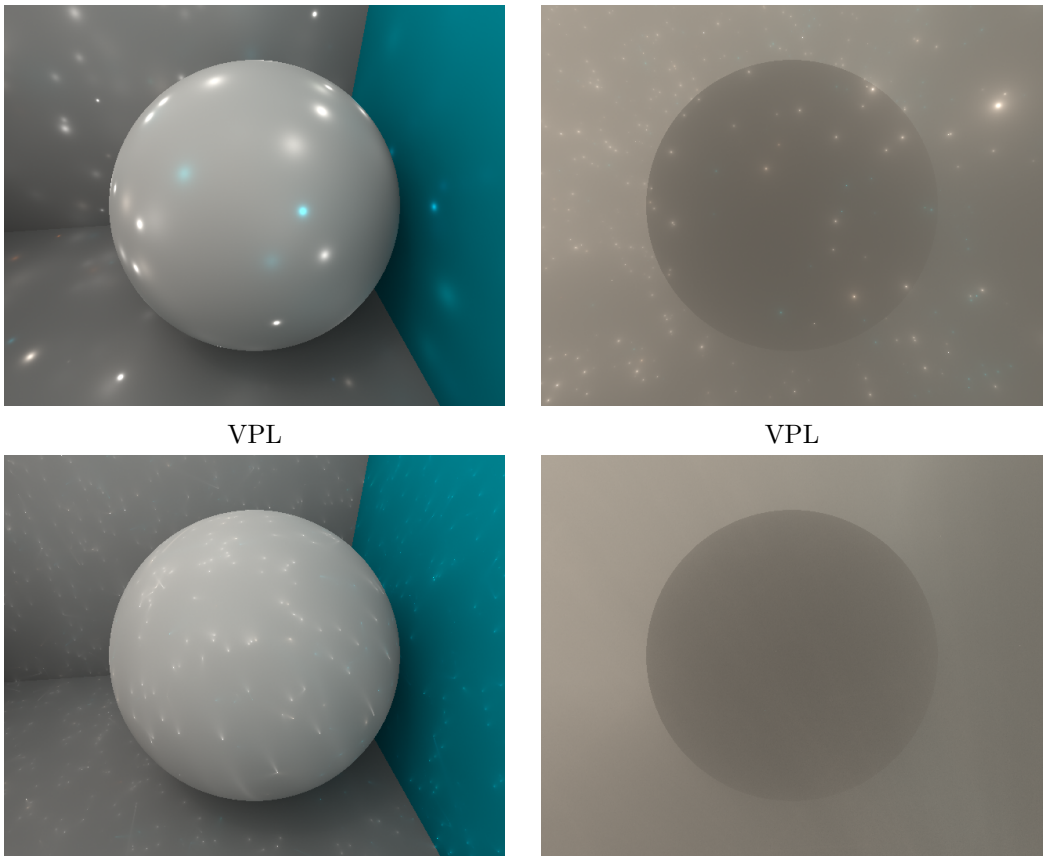


Figure 7.8: A comparison of the results from VPL and VRL. For both scenes, the same amount of virtual lights was used. The surface radiance (left) shows intensities of higher order for VPL. In the case of radiance from the medium (right), the singularities for VRL are barely visible compared VPL. Source: [NNDJ12b]

CHAPTER 8

Virtual Ray Lights - Implementation

For the implementation of this method an educational path tracer [ZF18b] was used as a basis, although a lot of changes and additions were made to the existing framework. Appendix B provides some additional information on the framework, credits to other contributions and availability.

This chapter is separated into four parts. In the first part, the general principle for the implementation is discussed. The second part describes the changes to the TRACE-function which is now used for the tracing of light and camera rays. The following parts describes the implementation of the formulae for the volume-to-volume and the volume-to-surface contribution which were discussed in the previous chapter. In the last part, the implementation of the sampling functions that were described previously are explained in detail.

The notation for the pseudocode in this chapter is shown in Table 8.1. In the interest of simplicity, the medium is assumed to be infinite and homogeneous as including these properties would decrease the readability while providing no implementation-specific information. Furthermore, there is only one light considered for the aforementioned reason.

Notation	Description
$\vec{d}$	a normalized direction, with its original length stored
$\bar{s}$	a segment
x_l	a variable associated with a light ray
x_c	a variable associated with a camera ray

Table 8.1: Notation of various variables for code examples.

8.1 General Principle

To understand the general principle, it is necessary to understand that this is a progressive method. Therefore, for the convergence, it does not matter if 100 samples per pixel are used and for each sample the contribution of one light ray is computed, or if for one sample per pixel the contribution of 100 light rays is computed.

Furthermore, the tracing of photons, which was already explained in Chapter 6, is now expanded to store entire segments. Therefore, for the following code examples, a few changes compared to the code examples from previous chapters are made. First, the new primitive Segment is introduced. This structure looks as described in Table 8.2.

Name	Description
o	the origin
e	the end
$\vec{d}$	the normalized direction
dist	the distance
power	for a light ray, the power at the origin

Table 8.2: The structure of the Segment data type.

A segment is a fixed length ray with an origin and an end point defined. The direction is stored as a unit vector. For computational advantages, the distance is computed once and stored within the structure. This structure is used for light and camera rays. For light rays, the power at the origin of the ray can be stored, for a camera ray, this property is disregarded. As is shown in Table 8.1, the notation for a segment in the code examples is $\bar{x}$, where x is the name of the segment. When a segment is initiated, only the origin and direction have to be provided, all other values can be empty which is denoted with the symbol $\emptyset$.

Algorithm 8.1: MainVRL()

```
1  for every sample do
2      for the number of light rays per sample do
3          s̄_l = Segment(p_l, ∅, d⃗_l, ∅, power_l)
4          TraceFromLight(s̄_l, 0)
5      end
6      for every pixel do
7          d⃗_c = ComputeCameraRayDirection(p_pixel)
8          s̄_c = Segment(p_c, ∅, d⃗_c, ∅, ∅)
9          color = TraceFromCamera(s̄_c, 0)
10     end
11 end
```

In the MainVRL-function (Algorithm 8.1) of this approach there are two passes for every sample. First, for the amount of light rays that is specified to be traced per sample, a new light ray is created and traced through the scene. As can be seen in Line 3, the position, direction and power of the light are given for the construction of a light ray. In practice, these values will change depending on the type of light. The spots for the end of the ray and the distance are left empty, as these values have to be determined while tracing the light ray through the scene. The newly created light rays are stored into a list which contains all current light rays.

The second loop, loops through all pixels to determine the color of the pixel. As in previous examples, the direction of the camera ray is created with the ComputeCameraRayDirection-function which was already explained in Chapter 3. The only thing that changes within this loop is the storing of the camera ray as a Segment. This is only done for continuity purposes.

8.2 The Tracing Functions

In the Main-function, there are two tracing functions. One for tracing the light ray (TraceFromLight) and one for tracing the camera ray (TraceFromCamera). While these tracing functions are discussed separately in this chapter, it is useful to unify these functions, as they share a lot of the same properties and decisions to be made. Especially when considering boundary and intersection decision for the medium, it is helpful to just use the same logic in one function, as opposed to two.

8.2.1 The TraceFromLight-Function

The function to trace a ray of light through the scene can be seen in Algorithm 8.2. As an input this function take a segment $\bar{s}_l$ that represents a light ray and a depth value. At the start, the current depth value is compared to the maximum number of bounces to assure an early return if that number is reached. Following this, the free-flight distance dist_{ff} is computed and the scene is intercepted. If an objected is intercepted, the hit point p_{hit} is computed. This point is stored as the endpoint of the current segment and the distance to this point is stored as the distance of the segment. As the segment is now complete, it can be stored with the other light rays with the StoreLightSegment-function. This function will not be explained further, as this is an implementation-specific function.

The next step is to evaluate the direction of reflection from the current hit point. The cases for the diffuse, specular and refractive objects are simplified compared to Chapter 3, as the computation of most of these values is equivalent. In the diffuse case, the power of the segment has to be adjusted by the color of the current object, which has to be divided by the probability of reflection. Furthermore, the transmission from the origin of the segment to the hit point has to be computed. The formula for the Transmission-function is shown in Section 5.1.3. Implementation details are omitted from this documentation due to the simplicity of the formula.

Algorithm 8.2: TraceFromLight($\bar{s}_l$, depth)

if depth $\geq$ bounces **then return**
$\text{dist}_{\text{ff}} = -\ln(\xi/\sigma_t)$
intersection = IntersectScene($\bar{s}_l$)
if intersection **then**
  $p_{\text{hit}} = \bar{s}_l.o + \bar{s}_l.\vec{d} \cdot$ intersection.dist
  $\bar{s}_l.e = p_{\text{hit}}$
  $\bar{s}_l$.dist = intersection.dist
  StoreLightSegment($\bar{s}_l$)
  obj = intersection.obj
  if obj *is diffuse* **then**
    $\vec{d}$ = DiffuseDirection($\bar{s}_l.\vec{d}$, obj.$\vec{n}$)
    $\text{prob}_{\text{refl}} = \max(\text{obj.color}.r, \text{obj.color}.g, \text{obj.color}.b)$
    power = $\bar{s}_l$.power $\cdot$(obj.color / $\text{prob}_{\text{refl}}$)
    power = power $\cdot$Transmission($\bar{s}_l$.dist)
    $\bar{s}_{l_{\text{new}}}$ = Segment($p_{\text{hit}}, \emptyset, \vec{d}, \emptyset$, power)
    TraceFromLight($\bar{s}_{l_{\text{new}}}$, depth +1)
  end
  if obj *is specular* **or** obj *is refractive* **then**
    $\vec{d}$ = Direction($\bar{s}_l.\vec{d}$, obj.$\vec{n}$)
    power = $\bar{s}_l$.power $\cdot$Transmission($\bar{s}_l$.dist)
    $\bar{s}_{l_{\text{new}}}$ = Segment($p_{\text{hit}}, \emptyset, \vec{d}, \emptyset$, power)
    TraceFromLight($\bar{s}_{l_{\text{new}}}$, depth +1)
  end
else
  $\bar{s}_l.e = \bar{s}_l.o \cdot \infty$
  $\bar{s}_l$.dist = ∞
  StoreLightSegment($\bar{s}_l$)
end
if intersection.dist > dist_{ff} **then**
  $p_{\text{scatter}} = \bar{s}_l.o + \bar{s}_l.\vec{d} \cdot \text{dist}_{\text{ff}}$
  $\bar{s}_l.e = p_{\text{scatter}}$
  $\bar{s}_l$.dist = dist_{ff}
  if $\sigma_s/\sigma_t < \xi$ **then return**
  $\vec{d}$ = SampleHenyeyGreenstein($\bar{s}_l.\vec{d}$)
  power = $\bar{s}_l$.power $\cdot$Transmission($\bar{s}_l$.dist)
  $\bar{s}_{l_{\text{new}}}$ = Segment($p_{\text{scatter}}, \emptyset, \vec{d}, \emptyset$, power)
  TraceFromLight($\bar{s}_{l_{\text{new}}}$, depth +1)
end

The final power at the current position is computed by the product of these term. With theses values, a new segment $\bar{s}_{l_{\text{new}}}$ can be created that starts at the current hit point into a direction that is chosen by the diffuse reflection and the power that was computed. This segment is then traced further through the scene. For specular and refractive objects, the computation of the values for a new segment is similar. The only difference is that the power does not have to be adjusted by the current surface color as these object are considered to be without a diffuse component. Other than that, the direction for a reflection is determined depending on the properties of the current object and the segment is further traced through the scene.

In the case that there is no intersection with any objects, the segment is continued to the end of the scene boundaries. In Algorithm 8.2 the scene boundaries are denoted as the distance ∞. Following this process, the segment is stored with the other segments.

A scattering event exists, if the free-flight distance dist_{ff} is shorter than the distance to the nearest intersection. In the case of a scattering event, the point of this event is computed and the current segment is adjusted accordingly. Although this segment is now complete, it is not stored with the other segments, as the contribution of this segment is already included with the previous computations. With the comparison of the term σ_s/σ_t to a random value $\xi \in [0, 1)$, the possibility of the scattering event is determined. If it is determined that a scattering event occurs, the process of creating a new segment and tracing this segment is similar to previous cases. The only difference is that the direction $\vec{d}$ of the new segment has to be determined by sampling the Henyey-Greenstein phase function with the current direction (SampleHenyeyGreenstein). As this procedure is well documented, it is exempt from this documentation.

Note that there is a difference in the behaviour of the scattering compared to other methods. As can be seen in Figure 7.1, compared to other methods that sample light scattering inside a medium, the light rays continue their path after a scattering event. This results in a more densely sampled media.

8.2.2 The TraceFromCamera-Function

The tracing of the camera segment can be seen in the TraceFromCamera-function in Algorithm 8.3. This function takes as input the current camera segment $\bar{s}_c$, a reference to a color value and a depth value. At the start of the function a comparison of the depth value with a threshold determines if an early exit is necessary. This is the simplest way of implementing a maximum depth. Other, more sophisticated, options are mentioned in Section 3.2.1.

The function is then separated into two parts, one part at which an intersection with an object occurs, and the other part where no intersection happen. In the case of an intersection, first the hit point is computed and the current segment is adjusted according. With this complete segment, the radiance of the medium up to the hit point can be computed with the RadianceMedium-function, which will be discussed in the next section. After the computed color is added to the current color, the case-separation between diffuse objects and specular or refractive objects is made.

Algorithm 8.3: TraceFromCamera($\bar{s}_c$, color, depth)

```
1  if depth ≥ threshold then return
2  if intersection then
3      p_hit = s̄_c.o + s̄_c.d⃗ · intersection.dist
4      s̄_c.e = p_hit
5      s̄_c.dist = intersection.dist
6      color = color + RadianceMedium(s̄_c)
7      obj = intersection.obj
8      if obj is diffuse then
9          color = color + RadianceSurface(s̄_c, obj.color, obj.n⃗)
10     end
11     if obj is specular or obj is refractive then
12         d⃗ = Direction(s̄_c.d⃗, obj.n⃗)
13         s̄_cnew = Segment(p_hit, ∅, d⃗, ∅, ∅)
14         TraceFromCamera(s̄_cnew, temp, depth + 1)
15         color = color + temp · Transmission(s̄_c.dist)
16     end
17 else
18     s̄_c.e = s̄_c.o · ∞
19     s̄_c.dist = ∞
20     color = color + RadianceMedium(s̄_c)
21 end
```

In the case of a diffuse object, the radiance at the hit point can be computed with the RadianceSurface-function, which will be discussed in the next section as well. In the case of a specular or refractive object, the direction of the reflection is computed and a new segment is used to gather the radiance along the continued path. The temporary color that is computed with this process has to adjusted by the transmission from the origin of the segment up to the current hit point.

In the simplest case, no intersection with the segment is computed. In this case, the segment is extended to the scene boundaries, which are denoted with the distance ∞ in this code example. The radiance from the medium is then computed for this segment.

8.3 Radiance Estimation

The estimation of the radiance is separated into two cases. The first case handles the contribution of light rays to a camera segment and is computed in the RadianceMedium-function. This is the volume-to-volume contribution, as it was previously called. The volume-to-surface contribution is handled in the second case. There, the contribution of light rays to a specific point on a surface is computed. This is done in the RadianceSurface-function.

8.3.1 The RadianceMedium-Function

Algorithm 8.4: RadianceMedium($\bar{s}_c$)

for *every light ray that is stored* **do**
  $\bar{s}_l$ = *the light segment at the current loop iteration*
  $[\text{cp}_c, \text{cp}_l]$ = ClosestPointsOnSegments($\bar{s}_c, \bar{s}_l$)
  $[\text{sample}_l, \text{pdf}_l]$ = GetLightRaySample($\bar{s}_c, \bar{s}_l, \text{cp}_c, \text{cp}_l$)
  cp_c = ClosestPointOnSegment($\bar{s}_c, \text{sample}_l$)
  if $g = 0$ **then**
    $[\text{sample}_c, \text{pdf}_c]$ = SimpleJointDistribution($\bar{s}_c, \text{sample}_l, \text{cp}_c$)
  else
    $[\text{sample}_c, \text{pdf}_c]$ = AdvancedJointDistribution($\bar{s}_c, \bar{s}_l, \text{sample}_l, \text{cp}_c$)
  end
  $\vec{d} = \text{sample}_l - \text{sample}_c$
  intersection = IntersectScene($\text{sample}_c, \vec{d}$)
  if intersection.dist $< \vec{d}$.length **then continue**
  f_{s_c} = PhaseFunctionHG($\bar{s}_c.\vec{d}, \vec{d}$)
  f_{s_l} = PhaseFunctionHG($\bar{s}_l.\vec{d}, \vec{d}$)
  trans_c = Transmission($\text{sample}_c - \bar{s}_c.o$)
  trans_l = Transmission($\text{sample}_l - \bar{s}_l.o$)
  trans = Transmission($\vec{d}$.length)
  $\text{integrand} = (\bar{s}_l.\text{power} \cdot \sigma_s \cdot \sigma_s \cdot f_{s_c} \cdot f_{s_l} \cdot \text{trans}_c \cdot \text{trans}_l \cdot \text{trans}) / \vec{d}.\text{length}^2$
  color = color + (integrand / ($\text{pdf}_c \cdot \text{pdf}_l$))
end
return color / *(number of light rays)*

The code example for the volume-to-volume contribution is shown in Algorithm 8.4, which describes the RadianceMedium-function. This function only needs the current camera segment $\bar{s}_c$ as an input. Other values, like all current light rays or medium properties, are assumed to be global for simplification purposes. This function loops through all light rays and computes the contribution of each light ray to the camera segment. The current light ray inside the loop body is denoted as the segment $\bar{s}_l$.

In a first step, the closest points on both segments to the other segment have to be computed. As this is mainly a geometrical task, the function ClosestPointsOnSegments disregarded from this documentation, but can be viewed in the provided source code. The points that are created from this function are cp_c, which is the closest point on the camera segment $\bar{s}_c$ to the light segment $\bar{s}_l$ and cp_l, which is the closest point on the camera segment $\bar{s}_l$ to the light segment $\bar{s}_c$.

With the points cp_c and cp_l and the segments $\bar{s}_c$ and $\bar{s}_l$ it is possible to create the sample on the light segment. This can be done by giving all these variables to the GetLightRaySample-function. This function, which will be explained in the next section, creates a sample sample_l on the light segment and the corresponding PDF pdf_l for this sample. With this sample, the closest point on the camera segment to this sample can be computed with the function ClosestPointOnSegment. The computed point is stored in the variable cp_c. Similar to the ClosestPointsOnSegments-function, the ClosestPointOnSegment function is a geometrical task, that is well documented and therefore not regarded in this documentation.

For the next part, the code example shows the ideal choice for the sampling function of the camera sample sample_c. In the isotropic case (Henyey-Greenstein, $g = 0$), the function SimpleJointDistribution is chosen, which corresponds to the equi-angular sampling method, that was explained in the previous chapter. In the anisotropic case, the AdvancedJointDistribution-function is used, which implements the sampling for the advanced joint distribution, described in the previous chapters, as well. Both sampling methods, which are explained in the following section, provide a sample sample_c along the camera segment and a PDF pdf_c corresponding to that sample. For educational and comparison purposes, in the implementation that is used to create the results of the following chapter, the sampling function can be chosen manually. The choice can be made between uniform, exponential, simple joint distribution and advanced joint distribution importance sampling. For this example code this is disregarded and only the ideal method is used.

The sampling is now complete, as the samples sample_l and sample_c are created. The following part computes the properties necessary to put everything into Equation 7.3. By intersecting the scene with the ray that is created by these two samples, the possibility that something is inbetween the samples is evaluated. If this is the case, the current light segment is not considered and the loop progresses with the next light segment. If there is nothing blocking the connection between the samples, the phase functions for the direction of the camera segment and the light segment are computed.

The formula for the PhaseFunctionHG-function is provided in Section 5.1.4. Due to the basic nature of this formula, implementation details are omitted. The transmission is computed for three parts, one for the origin of the camera segment to the sample on the camera segment, once from the origin of the light segment to the sample on the light segment and once between the samples. This computation can be done by providing the sum of these distances as an input to the transmission function, as these examples are considering only homogeneous media, but is separated for educational purposes to clearly show the different variables involved.

For the final integrand (see Equation 7.4 for an explanation of this terminology), the terms for the power of the light segment, the σ_s at the position of both samples, the results of the phase functions and transmission are multiplied and divided by the squared distance. Due to this being an example for homogeneous media only, the σ_s at the positions of the samples is the same. For the final contribution of this light segment, the integrand has to be divided by the product of the PDF pdf_l of the light sample sample_l and the PDF pdf_c of the camera sample sample_c.

After the loop is finished and all contributions of the light segments are summed up, the final result is divided by the number of segments that were evaluated in the loop to compute the final color that is returned.

8.3.2 The RadianceSurface-Function

The computation of the volume-to-surface contribution is similar to the volume-to-volume contribution and can be seen in Algorithm 8.5. This function needs the camera segment $\bar{s}_c$ as well as the color and normal of the object at the end of the camera segment as an input. In contrast to the RadianceMedium-function, the camera segment is not sampled, as the contribution of all light segments to the end position of the camera segment is computed. Therefore the functions SimpleJointDistribution and AdvancedJointDistribution are now used on the light segment to get a sample sample_l. The process is exactly the same, only the camera segment and the light segment, as well as all corresponding variables, are switched for the input into these functions. Instead of providing a sampled position, as in the volume-to-volume case, the end of the camera segment $\bar{s}_c.e$ is provided.

The sample sample_l and the PDF pdf_l that are generated can now be used to generated to necessary values of Equation 7.2. For these computations the end of the camera segment $\bar{s}_c.e$ serves as the second point of evaluation. As in the previous case, the connection between these points is evaluated for intersection, to possibly discard this light segment. The terms for the formula are mostly the same, only instead of the second phase function, the cosine-weigthed BRDF f_{r_c} is computed at the end of the camera segment. Here, the color and the normal of the object at the end of the camera segment are needed for this computation. The transmission is evaluated for the distance between the origin of the light segment and the sample on the light segment, as well as the distance between the light sample and the end of the camera segment.

Algorithm 8.5: RadianceSurface($\bar{s}_c$, obj.color, obj.$\vec{n}$)

for *every light ray that is stored* **do**
  $\bar{s}_l$ = *the light ray at the current loop iteration*
  cp_l = ClosestPointOnSegment($\bar{s}_l, \bar{s}_c.e$)
  if $g = 0$ **then**
    $[\text{sample}_l, \text{pdf}_l]$ = SimpleJointDistribution($\bar{s}_l, \bar{s}_c.e, \text{cp}_l$)
  else
    $[\text{sample}_l, \text{pdf}_l]$ = AdvancedJointDistribution($\bar{s}_l, \bar{s}_c, \bar{s}_c.e, \text{cp}_l$)
  end
  $\vec{d} = \text{sample}_l - \bar{s}_c.e$
  intersection = IntersectScene($\bar{s}_c.e, \vec{d}$)
  if intersection.dist $< \vec{d}$.length **then continue**
  $f_{r_c} = \text{obj.color} \cdot (\text{obj}.\vec{n} \cdot \vec{d})$
  $f_{s_l} = \text{PhaseFunctionHG}(\bar{s}_l.\vec{d}, \vec{d})$
  $\text{trans}_l = \text{Transmission}(\text{sample}_l - \bar{s}_l.o)$
  trans = Transmission($\vec{d}$.length)
  integrand = $(\bar{s}_l.\text{power} \cdot \sigma_s \cdot f_{r_c} \cdot f_{s_l} \cdot \text{trans}_l \cdot \text{trans}) / \vec{d}.\text{length}^2$
  color = color + (integrand / pdf_l)
end
return color / *(number of light rays)*

The integrand is then constructed by the product of the power of the light segment, the σ_s at the position of the light sample, the cosine-weighted BRDF, the result of the phase function and the transmission which are divided by the squared distance. This integrand is then divided by the PDF pdf_l of the sample sample_l on the light segment. As in the previous function, the final sum of the contribution of all light segments is divided by the number of all light segments that are evaluated. This final value is then returned as the result of this function.

8.4 The Sampling Functions

The three sampling functions that are used in the algorithms above are already part of the discussion of the previous chapter. The formulae that are explained in detail there are provided implementation examples for in this section. First, the function to create a sample on the light ray is explained, afterwards the functions for creating samples for the simple and advanced joint distribution techniques are evaluated.

Algorithm 8.6: GetLightRaySample($\bar{s}_c, \bar{s}_l, \text{cp}_c, \text{cp}_l$)

1 $\text{dist}_l = (\bar{s}_l.o - \text{cp}_l).\text{length}$
2 $\text{dist}_{\text{cp}} = (\text{cp}_c - \text{cp}_l).\text{length}$
3 $\theta = \cos^{-1}(\bar{s}_c.\vec{d} \cdot \bar{s}_l.\vec{d})$
4 $\text{start} = \sinh^{-1}(-\text{dist}_l \,/\, \text{dist}_{\text{cp}} \cdot \sin(\theta))$
5 $\text{end} = \sinh^{-1}((\bar{s}_l.\text{dist} - \text{dist}_l)/\, \text{dist}_{\text{cp}} \cdot \sin(\theta))$
6 $\text{dist} = \text{dist}_{cp} \cdot \sinh((1-\xi) \cdot \text{start} + \xi \cdot \text{end})/\, \sin(\theta)$
7 $\text{sample} = \text{cp}_l + \bar{s}_l.\vec{d} \cdot \text{dist}$
8 $\text{pdf} = \sin(\theta)/((\text{end} - \text{start}) \cdot \text{sqrt}(\text{dist}_{cp}^2 + \text{dist}^2 \cdot \sin^2(\theta)))$
9 **return** [sample, pdf]

8.4.1 The GetLightRaySample-Function

The functions that creates a sample on the light segment by a distribution corresponding to a marginal PDF in the joint distribution that was described in Section 7.3, is implemented into the function GetLightRaySample which is shown in Algorithm 8.6. This functions takes two segments, a camera segment $\bar{s}_c$ and a light segment $\bar{s}_l$, as well as two points, the closest point on the camera segment cp_c and the closest point on the light segment cp_l as input.

The first thing to do is to create the distance dist_l from the origin of the light segment to the closest point on it and the distance dist_{cp} between the closest points which is the shortest distance between the segments. In the original formula dist_{cp} is denoted as h. The angle θ is given by the angle of both segment directions.

For the next variable a slight change of naming was applied, compared to the Equation 7.12. The variable start is the term $A(\hat{v}_0)$ and the variable end is equivalently the term $A(\hat{v}_1)$ of this equation. The CDF^{-1} (Equation 7.14) is computed in the next line, where the distance dist is the distance from the closest point cp_l on the light segment to the sampled position. The sampled position sample can therefore be computed by a simple vector operation. The corresponding PDF can be computed by inserting all terms into the Equation 7.12. The created sample and the corresponding PDF are returned as the result of this function.

8.4.2 The SimpleJointDistribution-Function

Algorithm 8.7: SimpleJointDistribution($\bar{s}, p, \text{cp}$)

1 $\text{dist}_s = (\bar{s}.o - \text{cp}).\text{length}$
2 $\text{dist}_{\text{cp}} = (\text{cp} - p).\text{length}$
3 $\text{start} = \tan^{-1}(-\text{dist}_s \,/\, \text{dist}_{\text{cp}})$
4 $\text{end} = \tan^{-1}((\bar{s}.\text{dist} - \text{dist}_s)/\, \text{dist}_{\text{cp}})$
5 $\text{dist} = \text{dist}_{\text{cp}} \cdot \tan((1 - \xi) \cdot \text{start} + \xi \cdot \text{end})$
6 $\text{sample} = \text{cp} + \bar{s}.\vec{d} \cdot \text{dist}$
7 $\text{pdf} = \text{dist}_{\text{cp}} \,/((\text{end} - \text{start}) \cdot (\text{dist}_{\text{cp}}^2 + \text{dist}^2))$
8 **return** [sample, pdf]

The function for creating a sample along a segment in the simple joint distribution case is called SimpleJointDistribution and is shown in Algorithm 8.7. This is the equi-angular sampling that was discussed in Section 7.3.1 and provides the conditional PDF in the joint distribution. In the volume-to-volume contribution case, it is used to create a sample on the camera segment according to the sampled position on the light segment. In the volume-to-surface contribution case, it is used to create a sample on the light segment according to the end position of the camera segment and is therefore not part of a joint distribution, as there is only one distribution in the volume-to-surface case. This function takes a segment $\bar{s}$ on which the sample should be generated as well as the point p, which is the point of interest for the creation of the PDF which is proportional to the inverse-squared distance. Furthermore, the closest point cp on the segment to the given point p is given.

In the first part, similar to the GetLightRaySample-function, the distance dist_s between the origin of the segment and the closest point and the distance dist_{cp} between the closest point and the point of interest are computed. The start and end variables are corresponding to the terms θ_1 and θ_M of Equation 7.7. In the next line, the CDF^{-1} (Equation 7.8) can already be computed with the generated values to compute the distance dist. As in the previous case, this distance is the distance from the closest point on the segment to the sampled position. The PDF can be computed by Equation 7.8 and this can be implemented without any changes. This function returns the sampled position sample and the corresponding PDF pdf.

8.4.3 The AdvancedJointDistribution-Function

Algorithm 8.8: AdvancedJointDistribution($\bar{s}, p, \vec{d}$, cp)

```
1  dist_s = (s̄_c.o − cp).length
2  dist_cp = (cp − p).length
3  ā = s̄.o − p
4  b̄ = s̄.e − p
5  c̄ = ā × b̄
6  θ_1 = tan^{-1}(− dist_s / dist_cp)
7  θ_M = tan^{-1}((s̄.dist − dist_s)/ dist_cp)
8  CreateSamplesOnArc(θ_1, θ_M, ā, b̄, c̄, d̄, s̄.d̄)
9  cdf = ConstructPdfCdf()
10 ξ̂ = ξ · cdf
11 for all piecewise segments do
12 |  linear = the current piecewise linear function
13 |  if cdf_comb < ξ̂ and ξ̂ < cdf_comb + linear.cdf then
14 |  |  if linear.k = 0 then
15 |  |  |  θ_s = (linear.θ_o · linear.d + ξ̂ − cdf_comb)/ linear.d
16 |  |  |  pdf_{f_uv} = linear.d/ cdf
17 |  |  else
18 |  |  |  temp = linear.k · (linear.θ_o^2 · linear.k + 2 · ξ̂ − 2 · cdf_comb)+
19 |  |  |        2 · linear.θ_o · linear.d · linear.k + linear.d^2
20 |  |  |  θ_s = (sqrt(temp) − linear.d)/ linear.k
21 |  |  |  pdf_{f_uv} = (linear.k · θ_s + linear.d)/ cdf
22 |  |  end
23 |  |  break
24 |  end
25 |  cdf_comb = cdf_comb + linear.cdf
26 end
27 dist = dist_cp · tan(θ_s)
28 sample = cp + s̄.d̄ · dist
29 pdf = pdf_{f_uv} · (dist_cp /(dist_cp^2 + dist^2))
30 return [sample, pdf]
```

As can already be assumed from the corresponding mathematical analysis of Section 7.3.2, the sampling function that is important for the advanced joint distribution increases the complexity of implementation. It is therefore important to know that Chapter 7, especially Section 7.3.2, is crucial for the understanding of the following implementation details.

The implementation of this AdvancedJointDistribution-function is provided in Algorithm 8.8. The function is the conditional part of the joint distribution, but can, equivalently to the simple version, also be used as a standalone sampling function in the volume-to-surface contribution case. The function needs a segment $\bar{s}$ as input as well as an interest point p and the closest point cp to this interest point on the segment, all equal to the simple case. Additionally, for the evaluation of the phase function at the interest point, an additional direction $\vec{d}$ is needed. This direction is according to direction of the segment that the interest point is taken from.

At the beginning, as in the previous cases, the distance dist_s from the origin of the segment to the closest point, as well as the distance dist_{cp} between the closest point and the interest point is computed. Afterwards, a few directions for the computation of the peak along the arc are created. The direction $\vec{a}$ from the interest point to the start of the segment, the direction $\vec{b}$ from the interest point to the end of the segment as well as the cross product $\vec{c}$ between these directions is created. Additionally, the angle from the closest point to the origin of the segment θ_1 and the angle from the closest point to the end of the segment θ_M are created. At this point, all necessary values are available to distribute the samples on the spherical arc with the CreateSamplesOnArc-function which is explained later on. The sampled θ are stored together with the product of the phase functions for the direction that can be derived from this angle.

The next step creates the piecewise linear non-normalized PDF and the corresponding CDF. The implementation of the ConstructPdfCdf-function can again be seen in Section 8.4.5. The linear piecewise function is stored as multiple instances of the data structure LinearFunction, where one of them describes one interval of the piecewise linear function. The notation for this type of structure can be seen in Table 8.3. This type has the boundaries θ_o and θ_e saved, as well as the values at these boundaries f_{s_o} and f_{s_e}. Furthermore, the slope-intercept form of this functions is computed with the slope k and the y-intercept d. Finally, the computed definite integral of this function is stored in the variable cdf. Before evaluating the CDF^{-1}, the random variable $\xi \in [0, 1)$ is multiplied

Name	Description
θ_o, θ_e	the interval boundaries
f_{s_o}, f_{s_e}	the product of the phase functions at the interval boundaries
k	the slope of the function
d	the y-intercept of the function
cdf	the computed definite integral of this function

Table 8.3: The structure of the LinearFunction data type.

with the CDF of the non-normalized PDF. The newly created $\hat{\xi}$ can then be used to find the corresponding interval of the piecewise linear non-normalized PDF. If the correct interval was reached, the linear function in that interval is used for the computation of the sampled angle θ_s and the corresponding PDF $\text{pdf}_{f_{\text{uv}}}$, which is proportional to the product of the phase functions.

There are two cases when it comes to evaluating these variables. In the first case, the slope linear.k is zero, which make the linear function in this interval constant. The angle θ_s and the PDF $\text{pdf}_{f_{\text{uv}}}$ can be computed the following way: If the phase function is isotropic, the whole piecewise linear PDF is constant. In the second case, the computation of the angle θ_s and the PDF $\text{pdf}_{f_{\text{uv}}}$ is increases in complexity, but Equation 7.26 can be followed for the implementation.

After creating a sampled angle θ_s, the distance dist from the closest point is computed. With this step, the actual sample can be created in the next step. The sampled PDF $\text{pdf}_{f_{\text{uv}}}$ is then multiplied by an adapted PDF for the inverse-squared distance to get a normalized PDF pdf. This functions then returns the sample and the PDF as a result of the computation.

8.4.4 The CreateSamplesOnArc-Function

Algorithm 8.9: CreateSamplesOnArc$(\theta_1, \theta_M, \vec{a}, \vec{b}, \vec{c}, \vec{d_1}, \vec{d_2})$

$\vec{e} = (\vec{c} \times \vec{d_2}) \times \vec{c}$
$\theta_{\text{peak}} = \cos^{-1}(\vec{a} \cdot \vec{e})$
if $\theta_{\text{peak}} \notin [\theta_1, \theta_M]$ **then**
  $\vec{e} = -\vec{e}$
  $\theta_{\text{peak}} = \cos^{-1}(\vec{a} \cdot \vec{e})$
end
if $\theta_{\text{peak}} \in [\theta_1, \theta_M]$ **then**
  $j_{\text{peak}} = \text{floor}((\theta_{\text{peak}} - \theta_1)/(\theta_M - \theta_1) \cdot M - 0.5)$
  for $j < j_{\text{peak}}$ **do**
    $\theta_j = (\theta_{\text{peak}} - \theta_1)/2 \cdot (1 - \cos((\pi \cdot (j-1))/(j_{\text{peak}} - 1)))$
    $f_{\text{uv}_j} = \text{PhaseFunctionHG}(\vec{d_1}, \theta_j) \cdot \text{PhaseFunctionHG}(\vec{d_2}, \theta_j)$
    SaveArcSample$(\theta_j, f_{\text{uv}_j})$
  end
  for $j > j_{\text{peak}}$ **and** $j < M$ **do**
    $\theta_j = (\theta_M - \theta_{\text{peak}})/2 \cdot (1 - \cos((\pi \cdot (j - j_{\text{peak}}))/(M - j_{\text{peak}})))$
    $f_{\text{uv}_j} = \text{PhaseFunctionHG}(\vec{d_1}, \theta_j) \cdot \text{PhaseFunctionHG}(\vec{d_2}, \theta_j)$
    SaveArcSample$(\theta_j, f_{\text{uv}_j})$
  end
else
  for $j < M$ **do**
    $\theta_j = (\theta_M - \theta_1)/2 \cdot (1 - \cos((\pi \cdot (j-1))/M))$
    $f_{\text{uv}_j} = \text{PhaseFunctionHG}(\vec{d_1}, \theta_j) \cdot \text{PhaseFunctionHG}(\vec{d_2}, \theta_j)$
    SaveArcSample$(\theta_j, f_{\text{uv}_j})$
  end
end

The CreateSamplesOnArc-function is necessary to distribute the samples along the spherical arc, as was described in Section 7.3.2.1. An example code of this function can be seen in Algorithm 8.9. The main principle for this function is to first find a possible peak location. If no peak location is found to be inside the interval $[\theta_1, \theta_M]$, the samples are distributed with a cosine-warped uniform spacing along the whole intervals $[\theta_1, \theta_M]$. If the peak falls into the interval $[\theta_1, \theta_M]$, the samples are distributed into the two subintervals $[\theta_1, \theta_{\text{peak}}]$ and $[\theta_{\text{peak}}, \theta_M]$ with an adapted cosine-warped uniform spacing. In addition, this function also calculates the product of the phase functions of the directions at these sampled positions with the two directions given. This function needs as input the angle from the closest point to the origin of the segment θ_1 and the angle from the closest point to the end of the segment θ_M. Additionally the direction $\vec{a}$ to the origin of the segment, the direction $\vec{b}$ to the end of the segment, their cross product $\vec{c}$ of these directions, the direction $\vec{d_1}$ of the segment as well as the direction $\vec{d_2}$ of the other segment, which the interest point belongs to, are needed.

In the first step the angle θ_{peak} for the possible peak is computed. If this angle is not in the interval $[\theta_1, \theta_M]$, the direction is reversed. For the computation of the previous mentioned cases, the first one, where θ_{peak} is in the interval $[\theta_1, \theta_M]$, is computed first. At the start, the index of the peak is determined, so the number of samples can be split up. The first loop distributes the samples that have a smaller index than j_{peak} inside the interval $[\theta_1, \theta_{\text{peak}}]$. For each of those samples the phase functions are evaluated with the PhaseFunctionHG-function and the product is stored. For simplicity, only the angle itself is given as an argument, as the direction can be constructed by rotating the direction towards the closest point with the angle θ_j with the direction $\vec{c}$ as the normal of the plane on which the rotation happens.

The pair of the angle θ and the product of the phase functions f_{uv_j} is stored with the function SaveArcSample. As this function is depending on implementation details, it is omitted from this documentation. The only thing that is done in this function is storing the pair into a data structure.

8.4.5 The ConstructPdfCdf-Function

The function ConstructPdfCdf of Algorithm 8.10 construct the piecewise linear function intervals and computes the definite integral of every interval. This function loops through the intervals and constructs the slope-intercept form, as well as computing the definite integral and storing all values in the data structure LinearFunction. The combined definite integral of all intervals is returned as the result.

Algorithm 8.10: ConstructPdfCdf()

for $j < M$ **do**
  $k = (f_{\mathrm{uv}_{j+1}} - f_{\mathrm{uv}_j})/(\theta_{j+1} - \theta_j)$
  $d = f_{\mathrm{uv}_{j+1}} - k \cdot \theta_{j+1}$
  $\mathrm{end} = (k \cdot \theta_{j+1}^2)/2 + d \cdot \theta_{j+1}$
  $\mathrm{start} = (k \cdot \theta_j^2)/2 + d \cdot \theta_j$
  $\mathrm{cdf} = \mathrm{end} - \mathrm{start}$
  SaveLinearFunction$(\theta_j, \theta_{j+1}, f_{\mathrm{uv}_j}, f_{\mathrm{uv}_{j+1}}, k, d, \mathrm{cdf})$
  $\mathrm{cdf}_{\mathrm{comb}} = \mathrm{cdf}_{\mathrm{comb}} + \mathrm{cdf}$
end
return $\mathrm{cdf}_{\mathrm{comb}}$

CHAPTER 9

Virtual Ray Lights - Results & Comparisons

This chapter presents the results of the implementation that was described in the previous chapter. Comparisons are made between the simple and the advanced joint distribution. Furthermore, comparison of reconstructions of figures to the original figures from publications are made. Although our reconstructions are only of approximate nature, the general behaviour and convergence of this implementation can be illustrated with it. The test and performance comparisons were made on an Intel Core i7-4770 CPU @ 3.4GHz with 16GB RAM running parallelized over all cores, with all results having a resolution of 512x512. A performance comparison to the original method is not possible as this method was implemented on a CPU-GPU framework where the sampling is done on the GPU, whereas our implementation was implemented in a CPU-only framework.

The first test case is a Cornell box with two cubes inside of it. In the second case, the light is only cast into one direction in an infinite medium. Similarly, in the following case, the light is cast into 10 specific directions. Another test case distributes the light uniformly in all directions. Finally, some results are compared to figures from other scientific papers to show similarities. For the different tests, the settings are always given in a table at the start of the section. In this table, the participating media is categorized as a tuple of the absorption coefficient σ_a and the scattering coefficient σ_s. In the general case, no additional bounces of the light are allowed. For some specific results, render times are provided in the format hh:mm:ss.

In most tests, there are four settings for importance sampling, uniform, exponential, joint-simple and joint-advanced. For the uniform tests the samples on the light ray were chosen uniformly, as well as the samples on the camera ray. The same principle was applied for the exponential sampling. For the joint-simple tests the samples are chosen with the simple joint distribution described in Chapter 7. The advanced technique that works well specifically for anisotropic media that was described in that chapter is denoted as joint-advanced. The difference in computing times for isotropic tests compared to anisotropic test while providing the same settings on everything else stems from the implementation of the Henyey-Greenstein phase function, as in the isotropic case, no computation of the function is needed.

The creation of converged references with a simpler method, i.e., VPT, was not possible due to the maintaining singularities along light rays even after over 24 hours of rendering. Therefore, the convergence of this implementation was assured by making a visual comparison to ensure that the methods converge to the the same results. For a further conviction of the convergence, parts of the results without visible singularities were compared statistically and reassured the correct convergence of this implementation. The references that can be seen in the following examples are therefore the results of the advanced sampling technique.

9.1 Cornell Box

Scene, Light & Medium Settings	
Light Position	(0, 1.9, -3)
Light Intensity	50 000
Medium (σ_a, σ_s)	0.1, 0.01

Converged Results

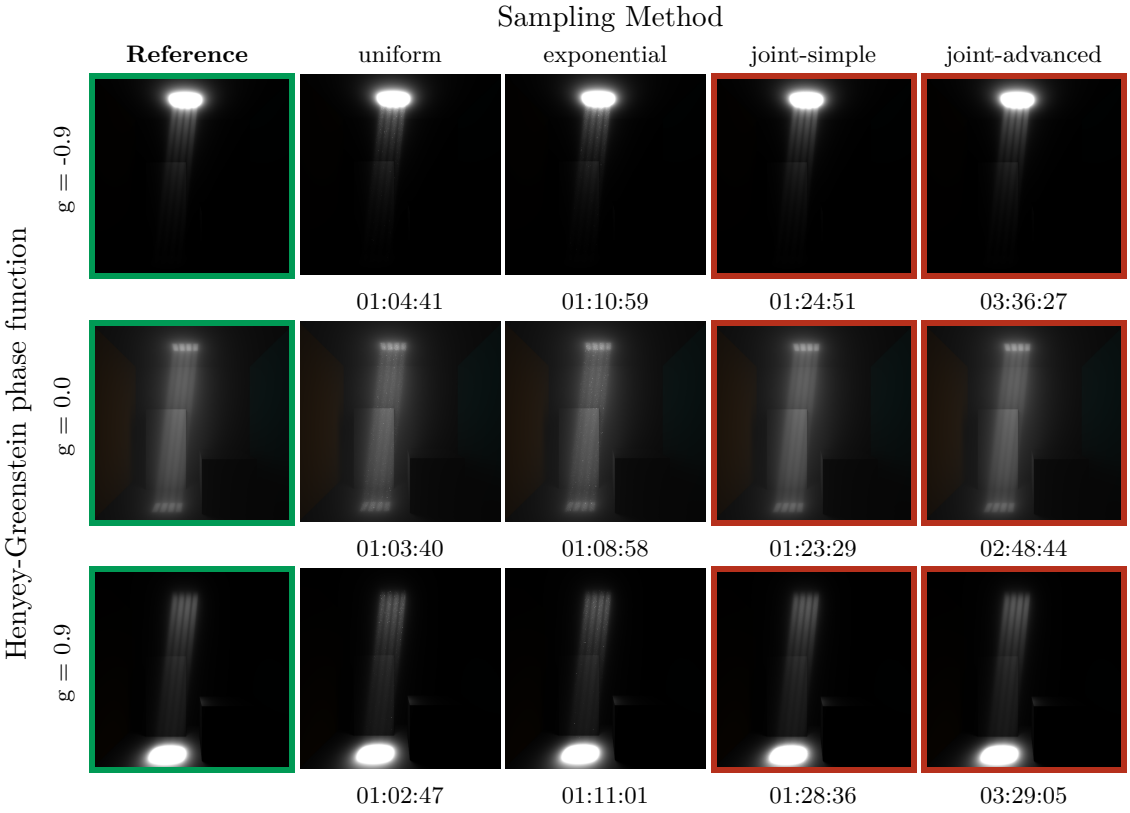


Table 9.1: Comparison of the Cornell box scene with 100 samples per pixel and 1 000 light rays per sample.

Table 9.1 shows the results of a Cornell box scene that was inspired by the scene that can be seen in the original paper with 100 samples per pixel and 1 000 light rays per sample. The starting position of a light ray is chosen randomly in the area of the 4 rectangles on the ceiling. The direction for all light rays is the same. The illumination of the surface is solely due to the contribution of the radiance from the media to the surface. In general, an approach like PPM would be suitable for this task as it would fit into the framework well. As can be seen in this comparison, for both, the uniform, as well as the exponential sampling, singularities are clearly visible along the light rays. The simple, as well as the advanced joint importance sampling provide results without any noticeable artefacts and seem to be fully converged. Values for the RMSE and PSNR are shown in Table 9.2.

	simple/advanced	
HG	RMSE	PSNR
-0.9	0.3124	58.2380
0.0	0.2369	56.6988
0.9	0.4207	55.6516

Table 9.2: Comparison of the RMSE and PSNR of Table 9.1.

Convergence Comparison

The convergence of the results with 1 sample, 10 samples and 100 samples is shown in Table 9.3, Table 9.4 and Table 9.5, respectively. There are some interesting observations that can be made from these results. First of all, the clear advantage in convergence of the advanced joint distribution technique can be seen. The results for both volume-to-volume contribution as well as volume-to-surface contribution appear more converged than the corresponding results with other techniques. Interestingly, while the simple joint distribution method shows some improvement in the volume-to-volume contribution, it is clearly the worst in the volume-to-surface contribution when anisotropic media is involved.

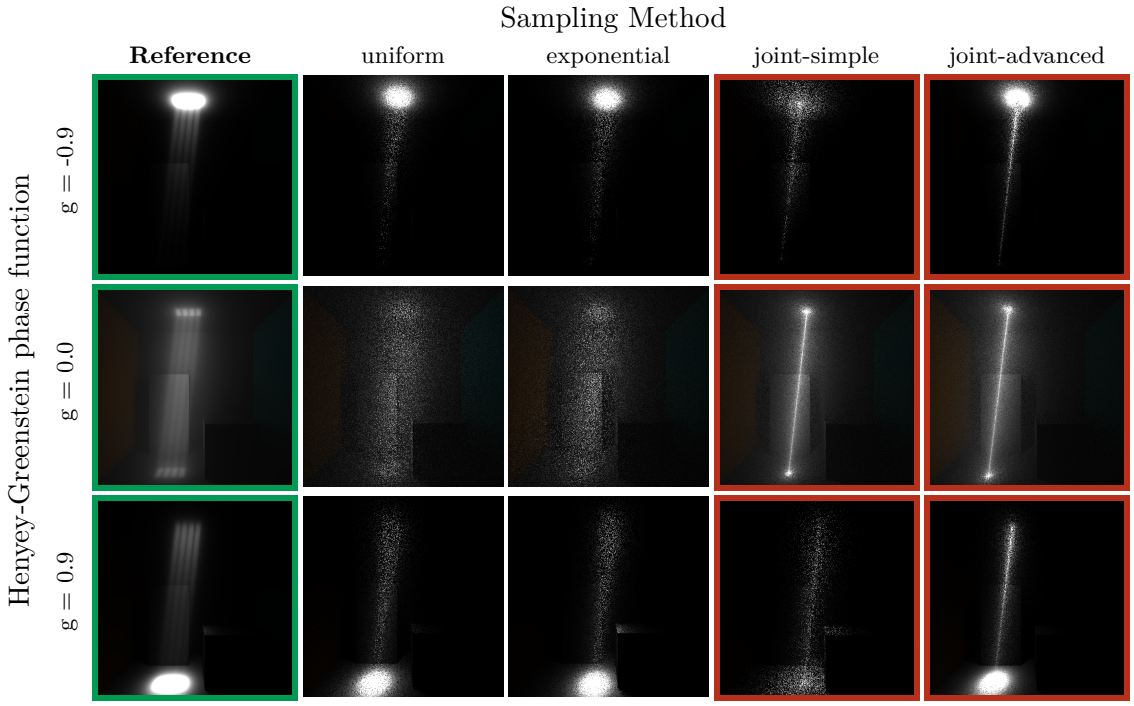


Table 9.3: Comparison of the Cornell box scene with 1 sample per pixel and 1 light ray per sample.

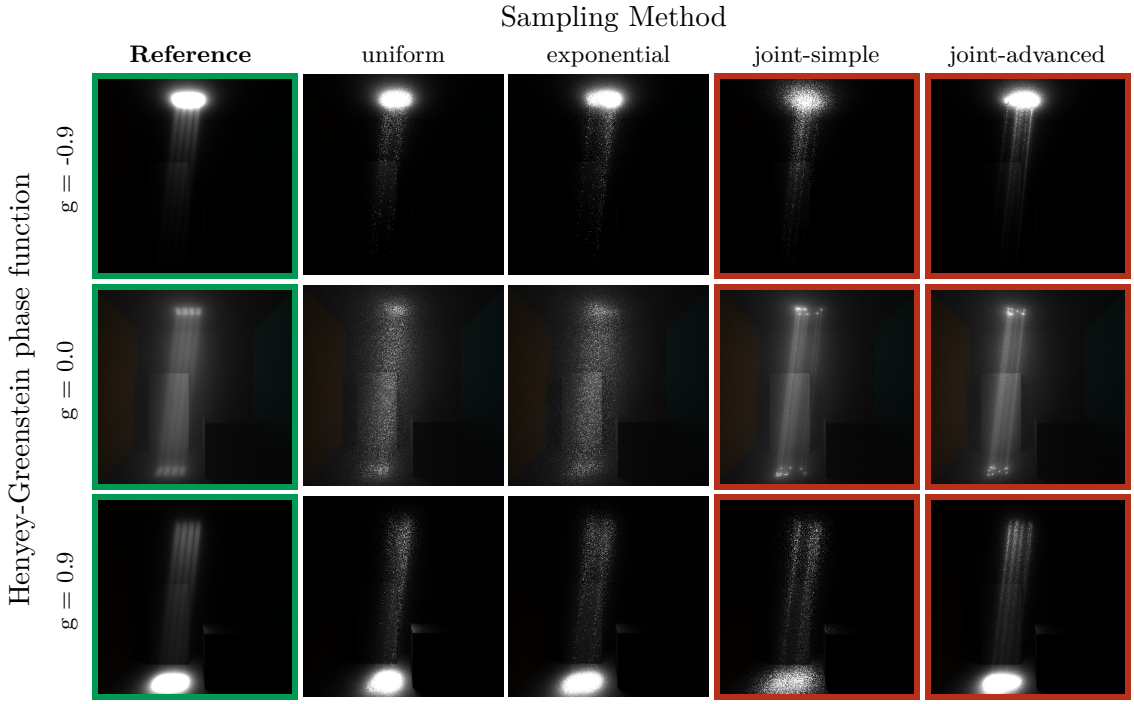


Table 9.4: Comparison of the Cornell box scene with 10 samples per pixel and 1 light ray per sample.

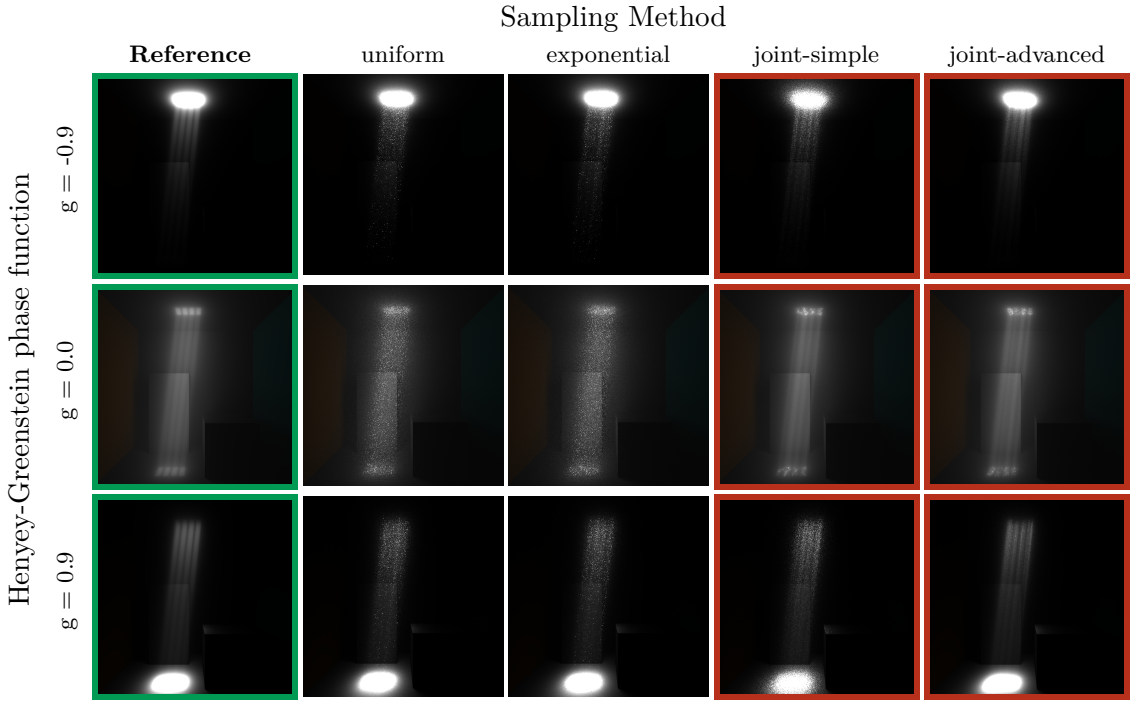


Table 9.5: Comparison of the Cornell box scene with 100 samples per pixel and 1 light ray per sample.

Equal Time Comparison

The comparison for the results after 1 minute can be seen in Table 9.6. As is shown in these results, the noise in the advance joint distribution case is almost gone. For the simple joint distribution, the volume-to-surface contribution in anisotropic media is clearly an issue, although in the volume-to-volume case, there are clear benefits over the uniform and exponential methods. For the uniform and exponential sampling, singularities are clearly visible along the light rays.

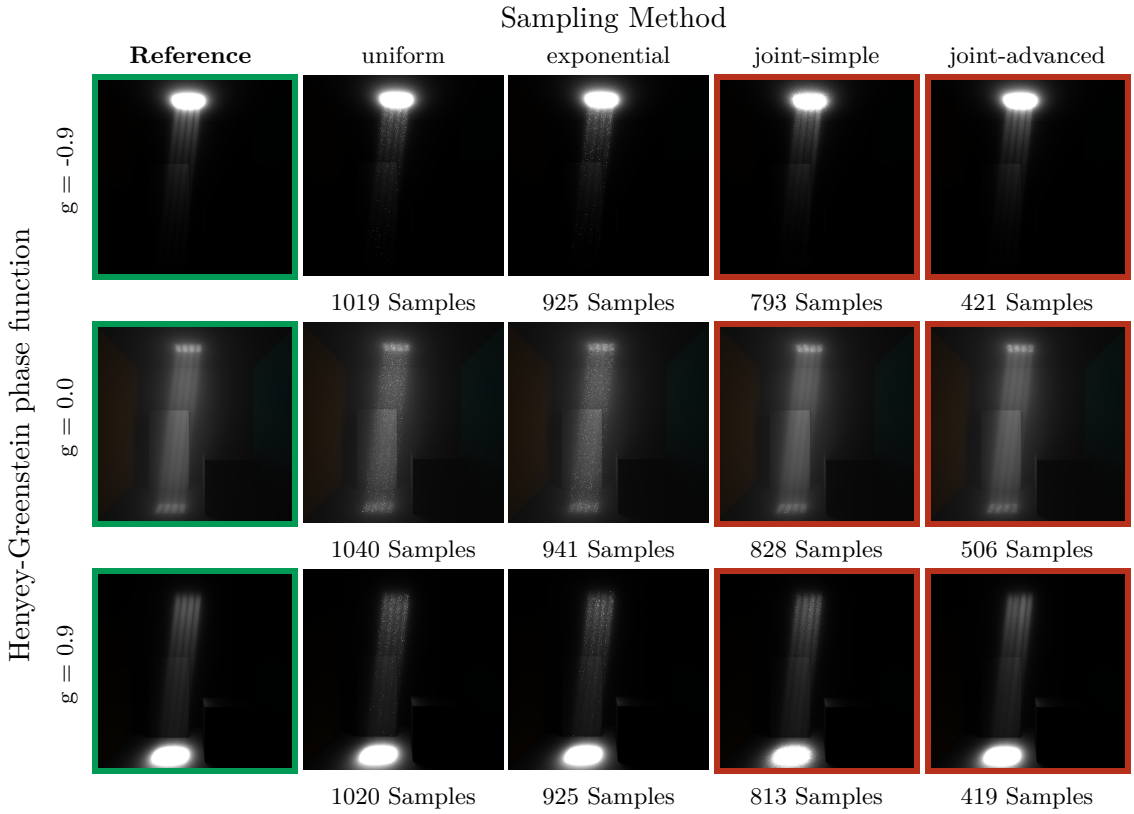


Table 9.6: Comparison of the Cornell box scene which was rendered for 60 seconds.

A Word on Progressiveness

As this is a progressive method, it is possible to adjust the number of samples and light ray, and as long as the product of both is the same, the result will be as well. As can be seen in the leftmost column, there is a difference in anti-aliasing, as the anti-aliasing is implemented as a beneficial side effect into the rendering process. It is therefore advisable to still use at least enough samples to achieve the aliasing. While the performance increases with fewer samples per pixel and more light ray per sample, as there are fewer intersections to test, the memory footprint increases as well as more light rays have to be stored at once.

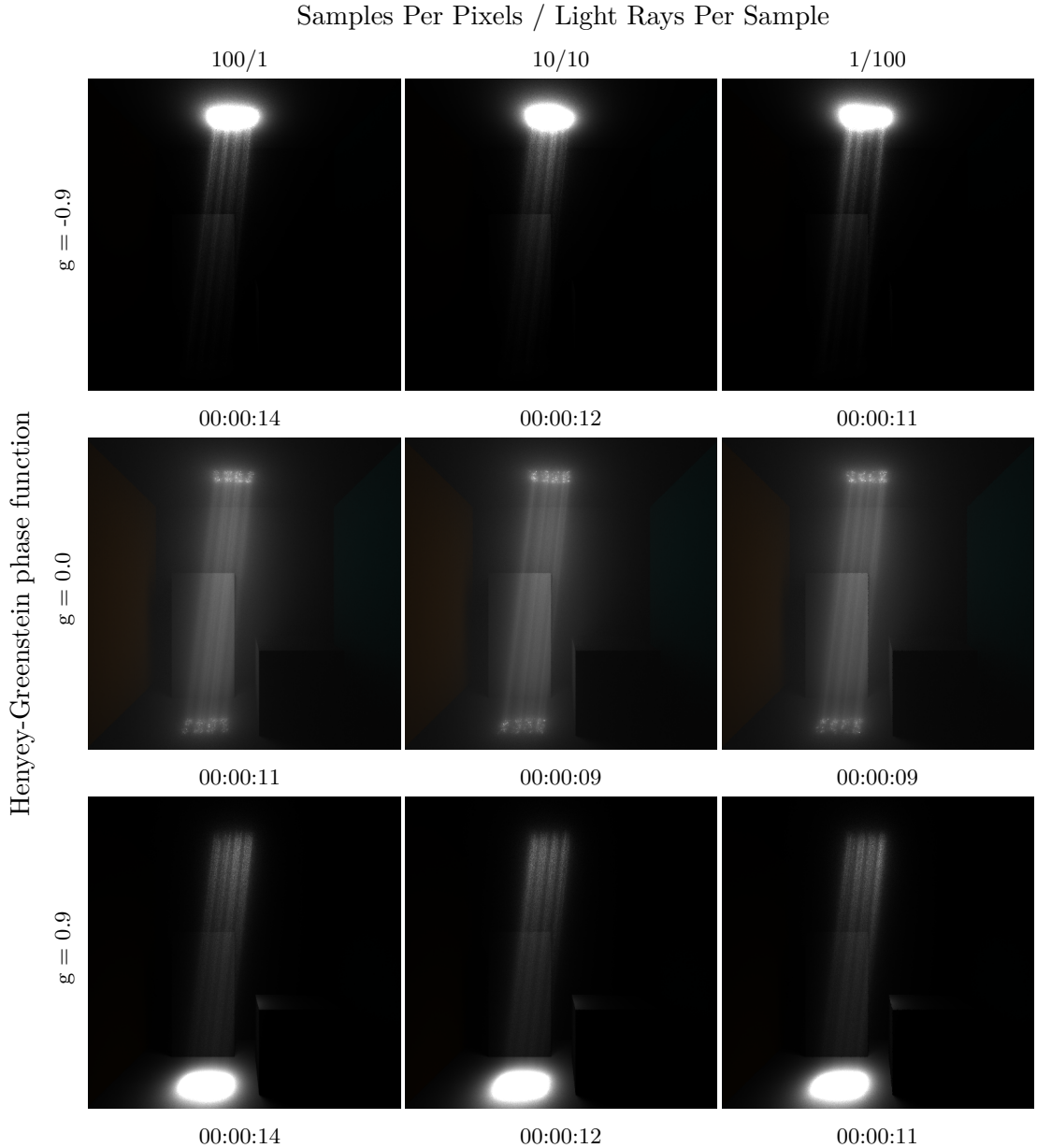


Table 9.7: Comparison of the Cornell box scene. The number of samples and light rays vary, but their product always stays 100. This equivalence of the product of light rays and samples has been addressed in Section 8.1.

Bounce Progression

For the progression of light rays through the scene Table 9.8 shows how the results look along the way. On the top, the number of light rays can be seen, whereas on the bottom, the rendering time is displayed. It should be noted that the rendering timings are not to be compared to the other results as there is a computational overhead, especially at the start, when exporting theses images.

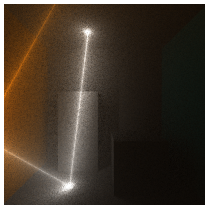
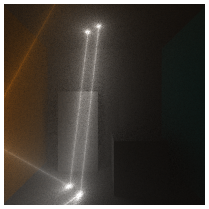
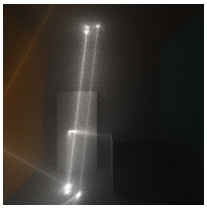
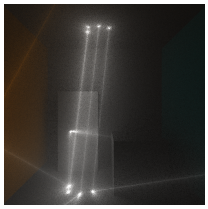
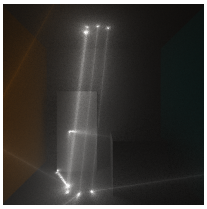
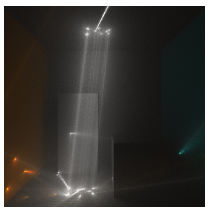
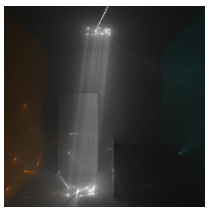
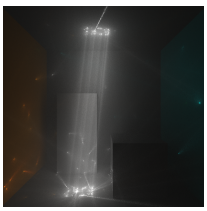
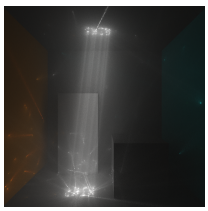
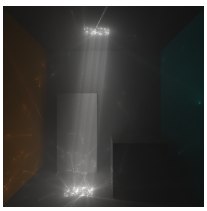
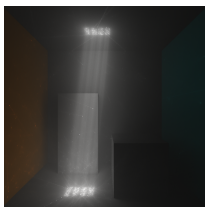
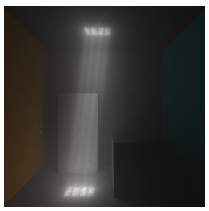
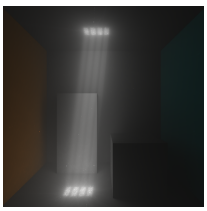
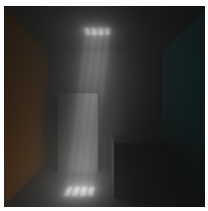
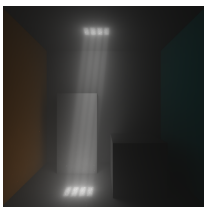
1 light ray	2 light rays	3 light rays	4 light rays	5 light rays
				
00:00:00	00:00:00	00:00:00	00:00:01	00:00:01
10 light rays	25 light rays	50 light rays	75 light rays	100 light rays
				
00:00:03	00:00:08	00:00:16	00:00:24	00:00:33
500 light rays	1000 light rays	3000 light rays	5000 light rays	10000 light rays
				
00:01:48	00:03:22	00:09:53	00:16:09	00:31:47

Table 9.8: Comparison of the Cornell box scene. This figure shows the progression of the bounces with an increasing number of light rays.

Bounces

The Cornell box with a different number of bounces for the light rays can be seen in Table 9.9, Table 9.10 and Table 9.11. All results are rendered with 100 samples per pixel and 10 light rays per sample. Table 9.9 shows the full result, whereas Table 9.10 shows just the volume-to-surface contribution and Table 9.11 shows just the volume-to-volume contribution. All images are rendered with the advanced joint distribution technique. As is shown here, significant artifacts can occur with multiple bounces in anisotropic media, as the data of the light rays is not as closely bundled anymore. This is also true for the volume-to-surface contribution, even though less visible, in the isotropic case, as the hit points of the light rays are clearly visible on the surfaces.

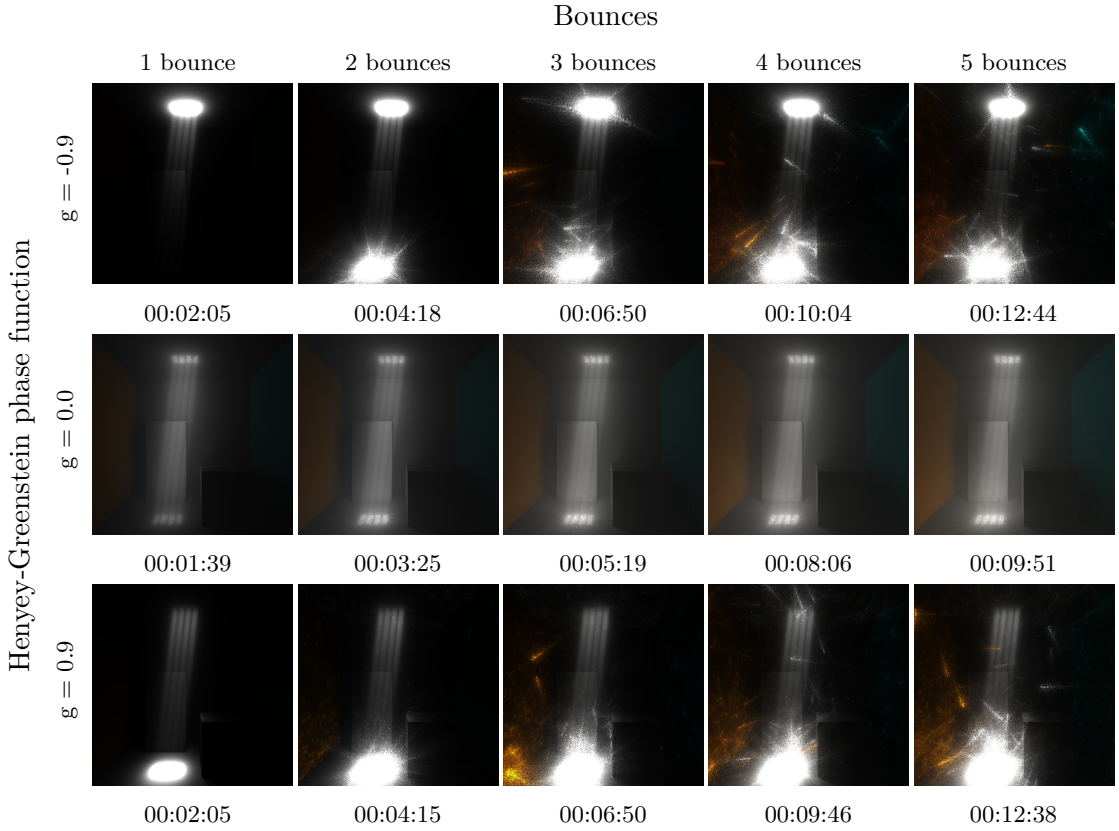


Table 9.9: Comparison of the Cornell box scene. This figure shows the progression of the renderings with a different number of bounces.

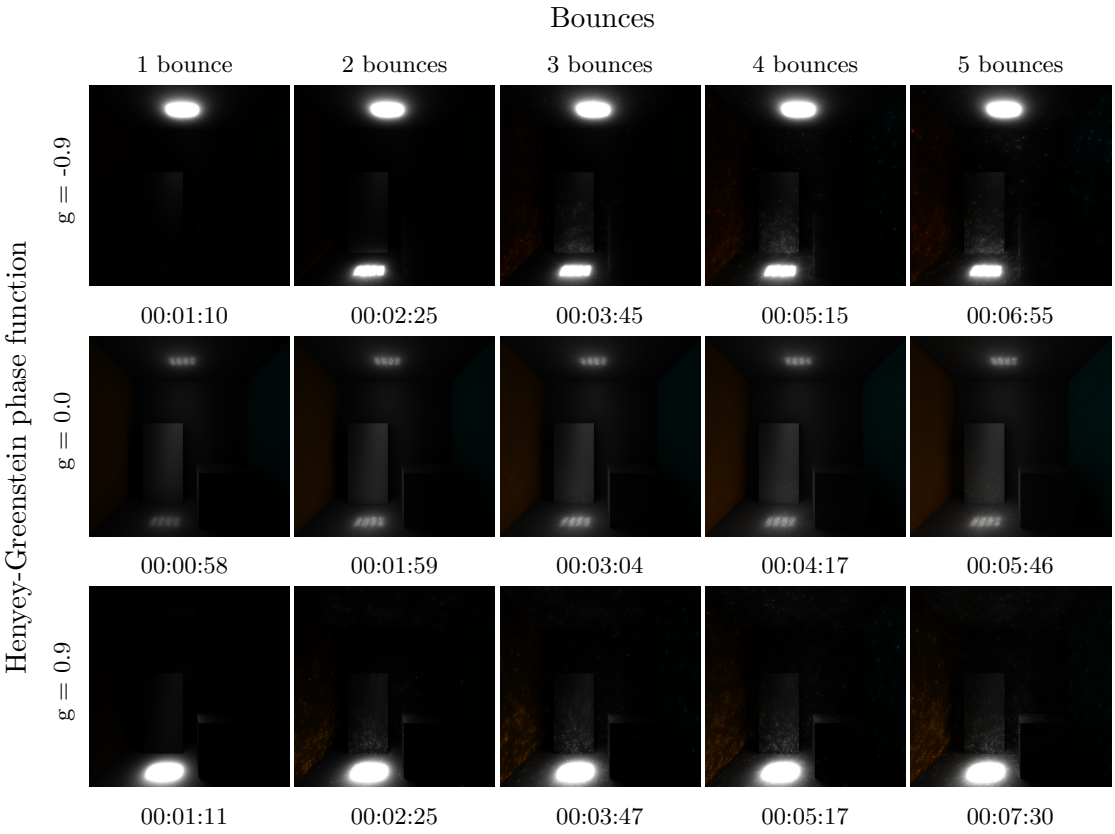


Table 9.10: Comparison of the Cornell box scene. This figure shows the progression of the renderings with a different number of bounces with volume-to-volume radiance only.

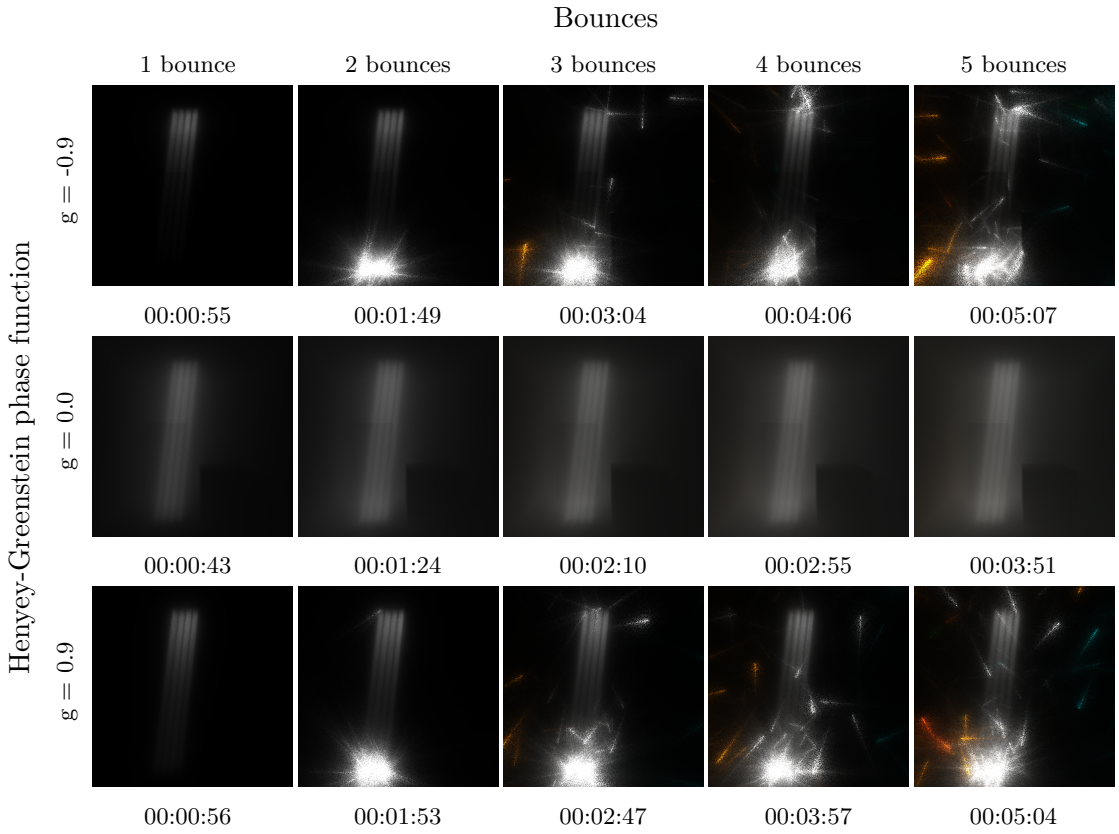


Table 9.11: Comparison of the Cornell box scene. This figure shows the progression of the renderings with a different number of bounces with volume-to-surface radiance only.

9.2 Point Light (bidirectional, 1 light ray)

Scene, Light & Medium Settings	
Light Position	(0, 0, -2)
Light Intensity	250
Medium (σ_a, σ_s)	0.1, 0.25

Converged Results

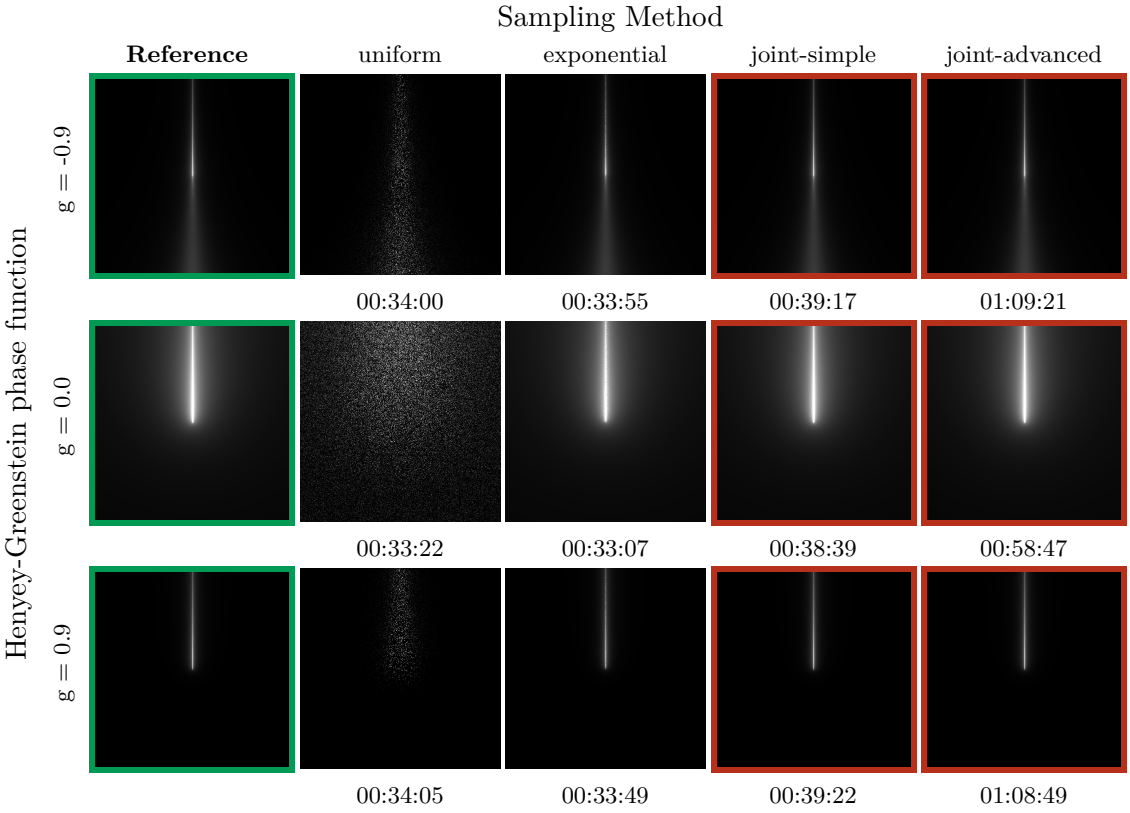


Table 9.12: Comparison of a scene with a point light (bidirectional, 1 light ray) with 50 000 sample per pixel and 1 light rays per sample.

The tests in Table 9.12 show a point light which casts one light ray into a specific direction. This is where this implementation can show its full potential. After 50 000 samples, the uniform case is not even close to a convergence and the exponential case still has significant noise in it. Both joint distributions converge to the reference solution within 50 000 samples. Table 9.13 shows the RMSE and PSNR which are similar to the results of the Cornell box tests.

	simple/advanced	
HG	RMSE	PSNR
-0.9	0.4075	55.9273
0.0	0.5491	53.3375
0.9	0.2589	59.8680

Table 9.13: Comparison of the RMSE and PSNR of Table 9.12.

Convergence Comparison

In Table 9.14, Table 9.15 and Table 9.16 the scene with 1 light ray is rendered with 1 sample, 10 samples and 100 samples per pixel. These results show where the algorithm really shines: The convergence with the joint distribution sampling is reached rapidly, and the improvement of the advanced joint distribution sampling can be seen clearly. Even after just 100 samples, a lot of noise has already been reduced and the final result can already be estimated.

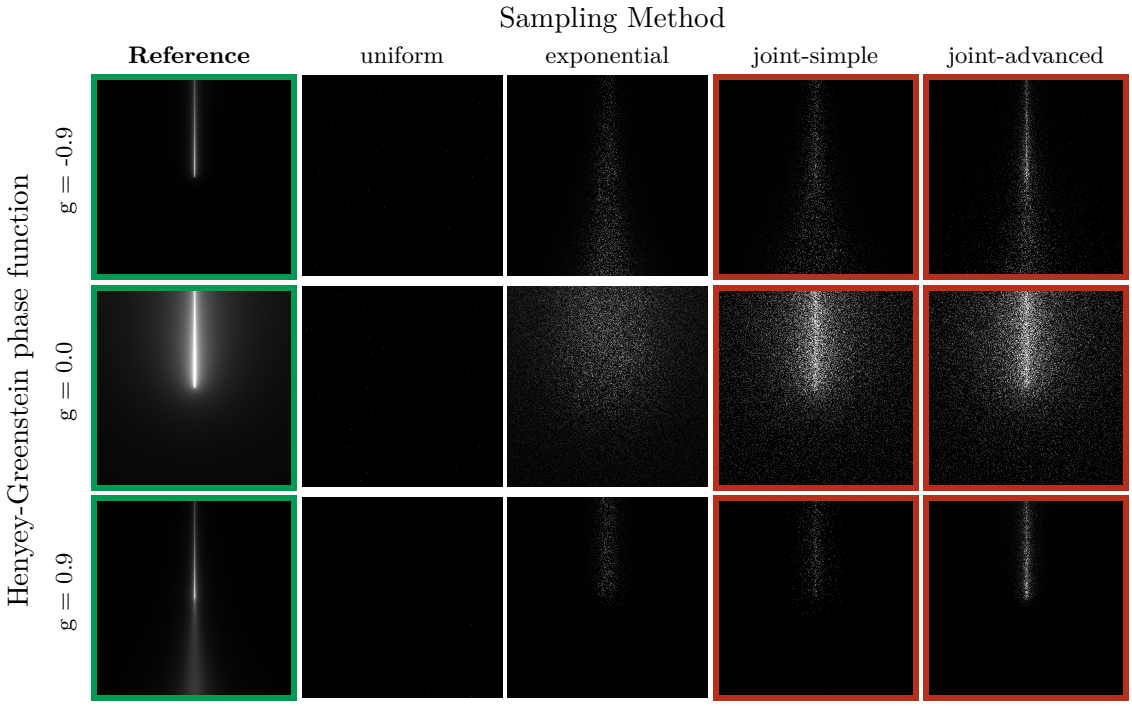


Table 9.14: Comparison of a scene with a point light (bidirectional, 1 light ray) with 1 sample per pixel and 1 light rays per sample.

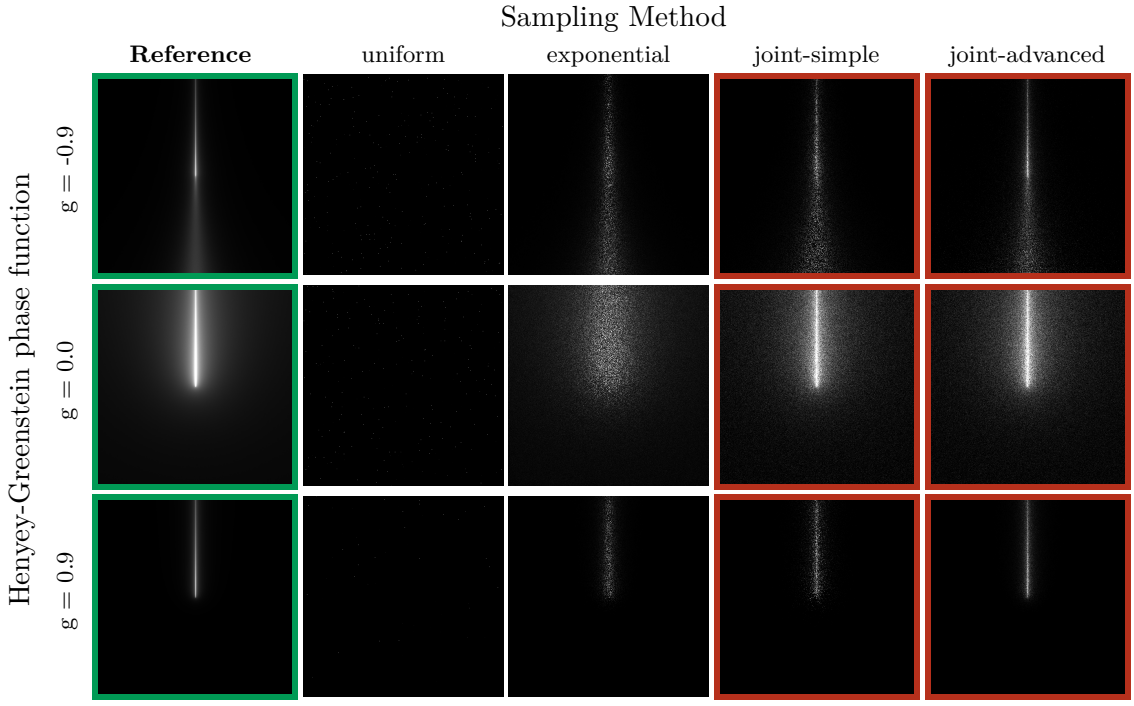


Table 9.15: Comparison of a scene with a point light (bidirectional, 1 light ray) with 10 sample per pixel and 1 light rays per sample.

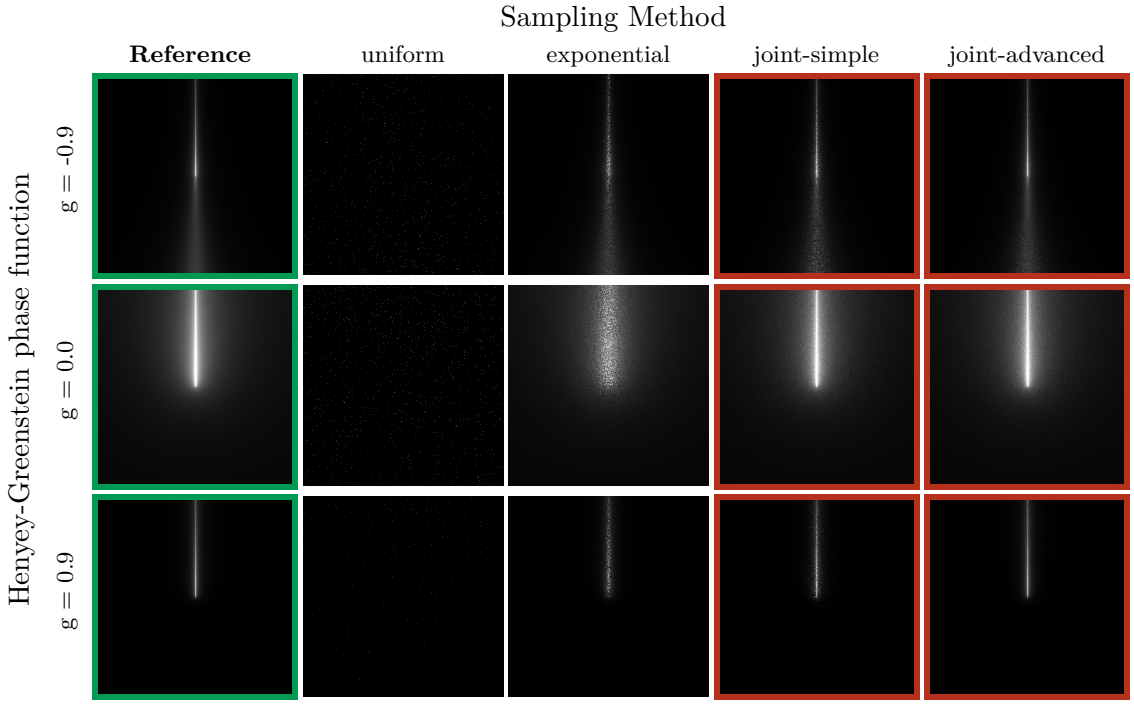


Table 9.16: Comparison of a scene with a point light (bidirectional, 1 light ray) with 100 sample per pixel and 1 light rays per sample.

Equal Time Comparison

For a comparison after 1 minute of rendering, Table 9.17 shows the results for all test cases. As is shown in these images, the joint distribution outperforms the uniform and exponential sampling by far. The advanced joint distribution has significantly less noise compared to the simple version of it in the case of anisotropic media. As the original method was implemented on a CPU-GPU setup, and these results are only from a CPU version, the difference can be expected to be even larger by performing the sampling on the GPU.

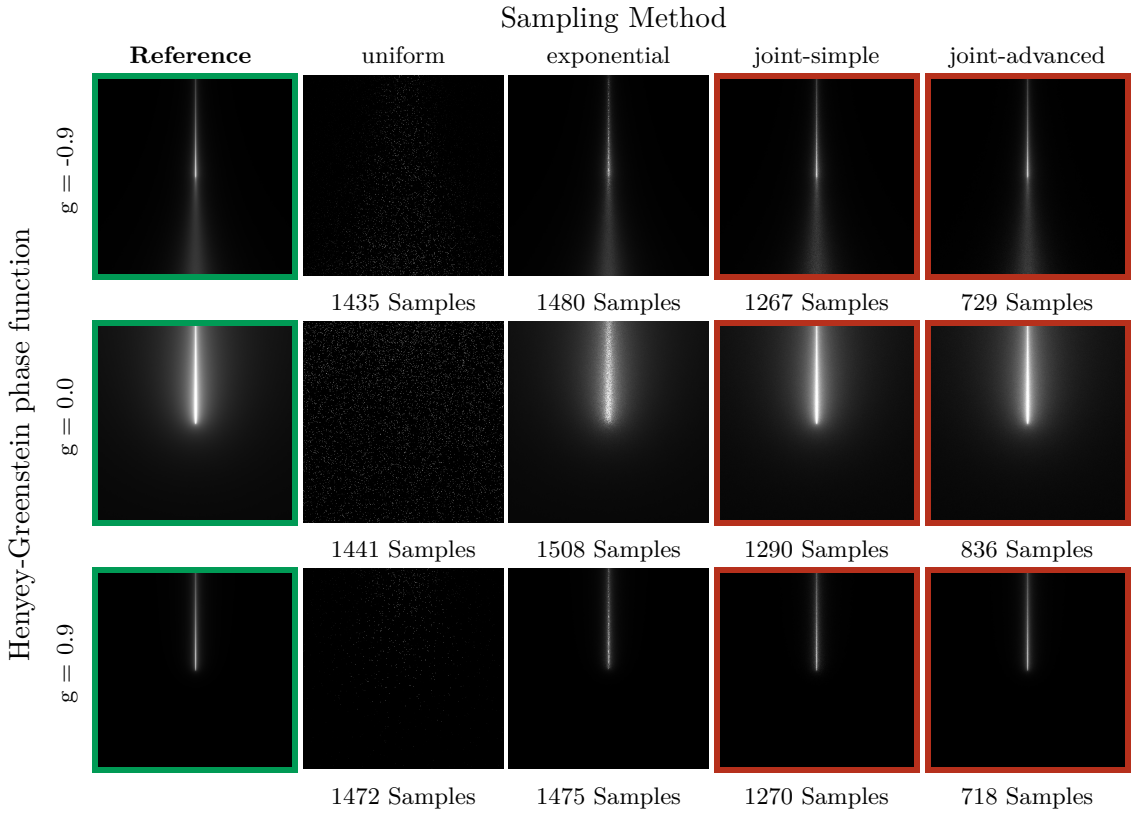


Table 9.17: Comparison of a scene with a point light (bidirectional, 1 light ray) which was rendered for 60 seconds.

9.3 Point Light (bidirectional, 10 light rays)

Scene, Light & Medium Settings	
Light Position	(0, 0, -2)
Light Intensity	250
Medium (σ_a, σ_s)	0.1, 0.25

Converged Results

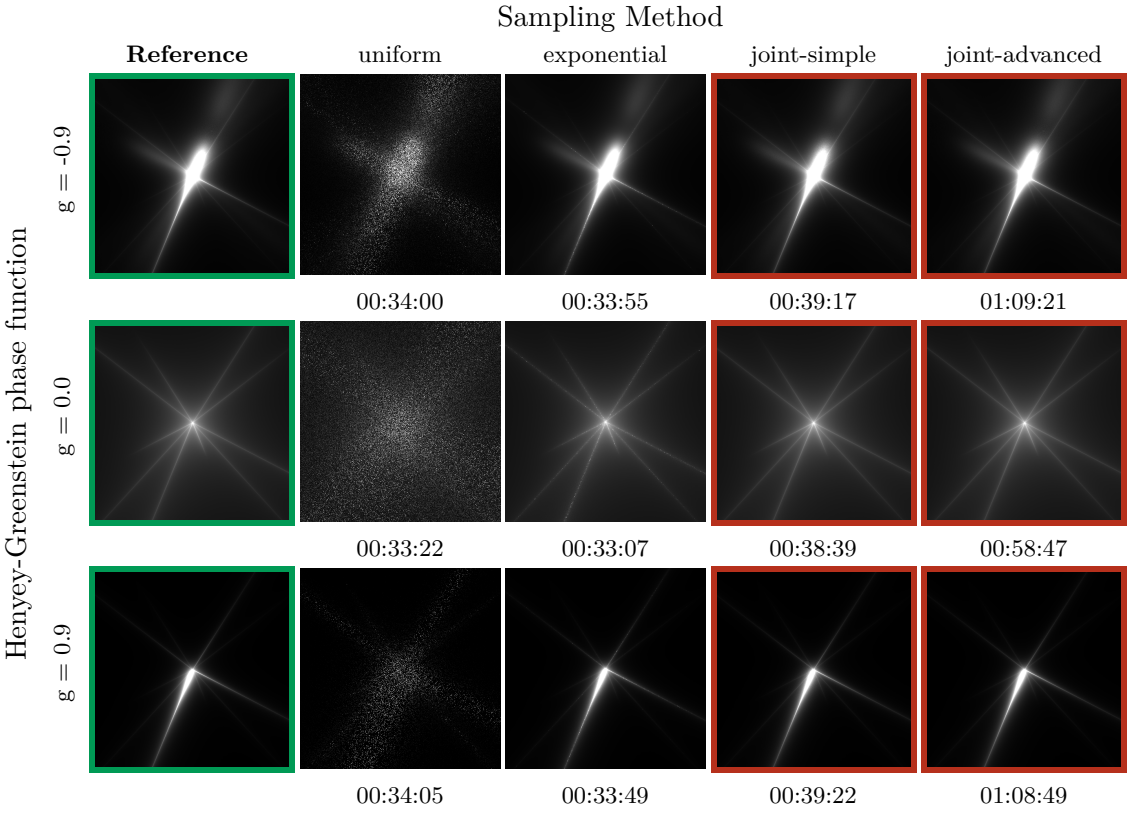


Table 9.18: Comparison of a scene with a point light (bidirectional, 10 light rays) with 50 000 sample per pixel and 10 light rays per sample.

Table 9.18 shows the results for the scene with a bidirectional point light with 10 light rays. The results allow similar conclusions as in the case with 1 light ray. The joint distribution allows for convergence after 50 000 samples. The uniform sampling does not seem to converge within a reasonable amount of time and the exponential sampling still has some disturbing singularities along the rays. The comparison of the RMSE and PSNR for these tests can be seen in Table 9.19.

	simple/advanced	
HG	RMSE	PSNR
-0.9	0.9854	48.2581
0.0	0.3547	57.1330
0.9	0.5309	53.6301

Table 9.19: Comparison of the RMSE and PSNR of Table 9.18.

Convergence Comparison

In Table 9.20, Table 9.21 and Table 9.22 the scene with 10 light rays is rendered with 1 sample, 10 samples and 100 samples per pixel. Again, the convergence of the joint distribution is clearly in favor compared to the other distributions. In the case of 100 samples, some noise may appear for back-scattering (i.e., g<0) for the advanced joint distribution. Although a bit disturbing, the overall results are clearly better than in the simple joint distribution.

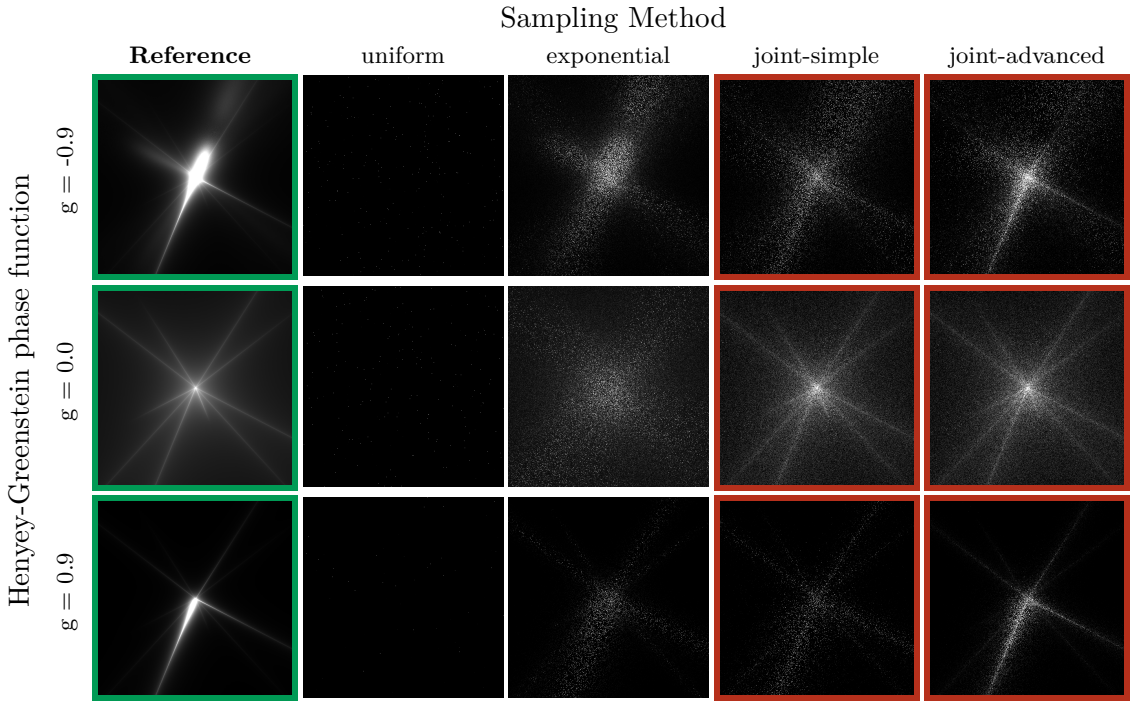


Table 9.20: Comparison of a scene with a point light (bidirectional, 10 light rays) with 1 sample per pixel and 10 light rays per sample.

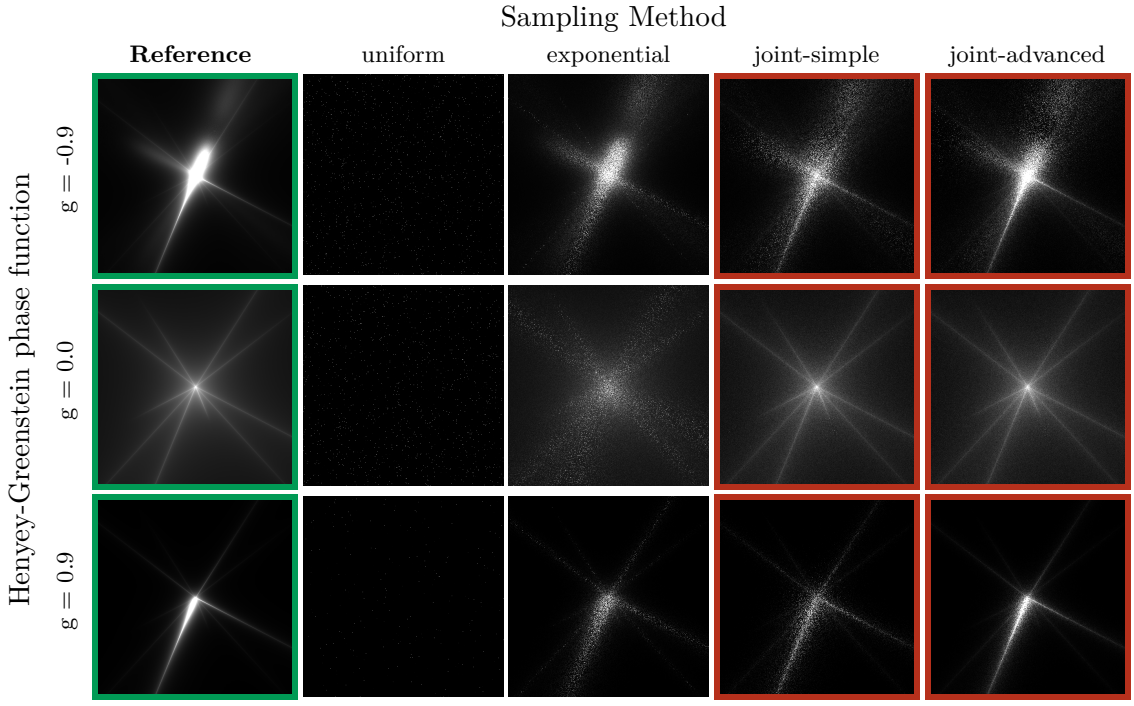


Table 9.21: Comparison of a scene with a point light (bidirectional, 10 light rays) with 10 sample per pixel and 10 light rays per sample.

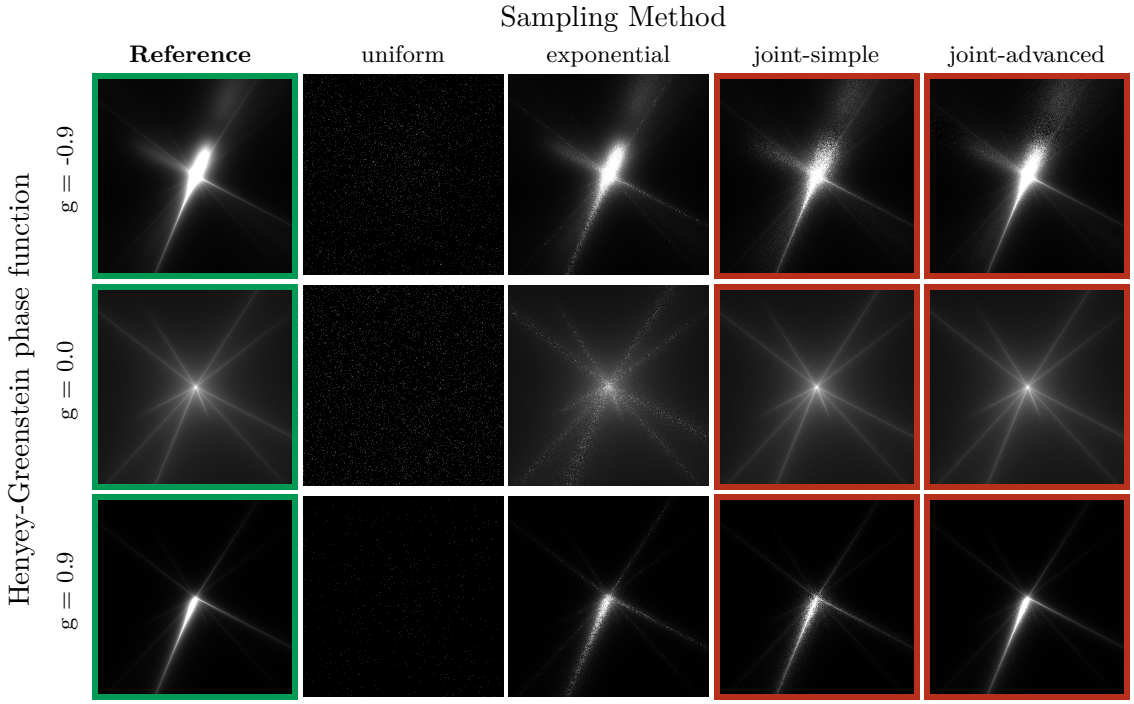


Table 9.22: Comparison of a scene with a point light (bidirectional, 10 light rays) with 100 sample per pixel and 10 light rays per sample.

Equal Time Comparison

The comparison after 1 minute of render time is shown in Table 9.23. Again, the results of the advanced joint distribution show the best results. As was previously described, the difference may get even larger with an implementation of this method on the GPU.

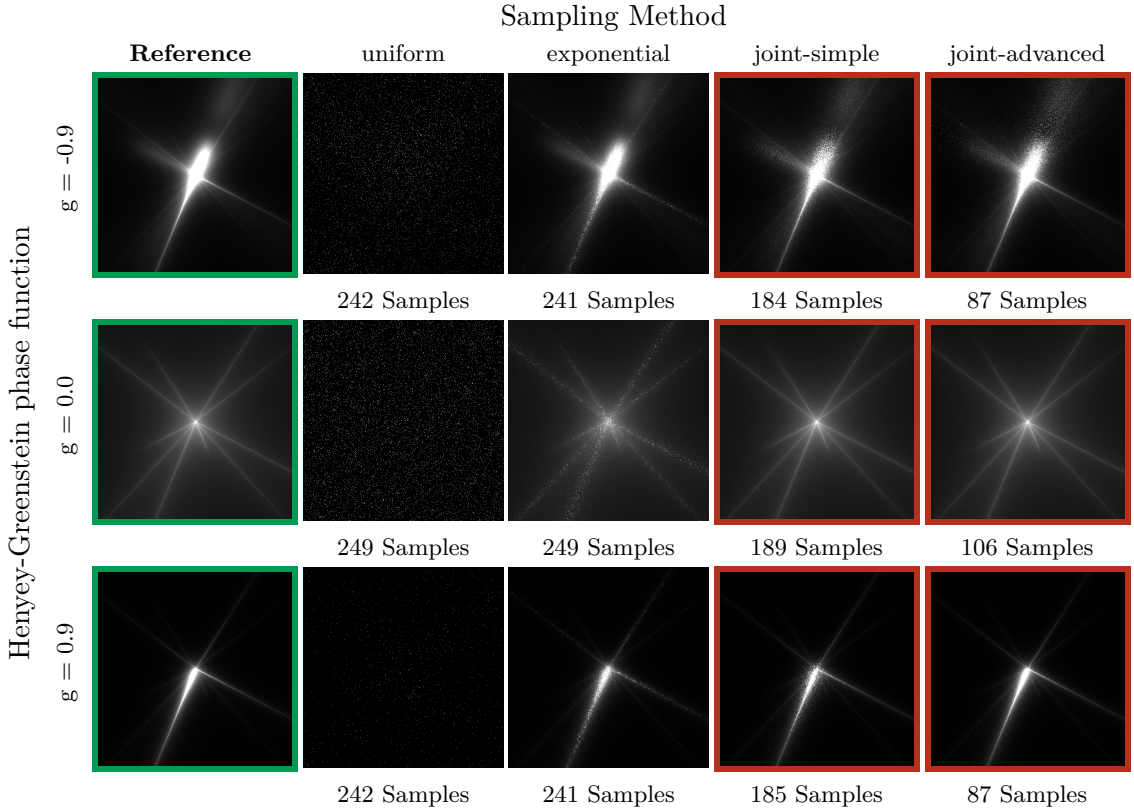


Table 9.23: Comparison of a scene with a point light (bidirectional, 10 light rays) which was rendered for 60 seconds.

9.4 Point Light (unidirectional)

Scene, Light & Medium Settings	
Light Position	(0, 0, -2)
Light Intensity	100
Medium (σ_a, σ_s)	0.1, 0.25

Converged Results

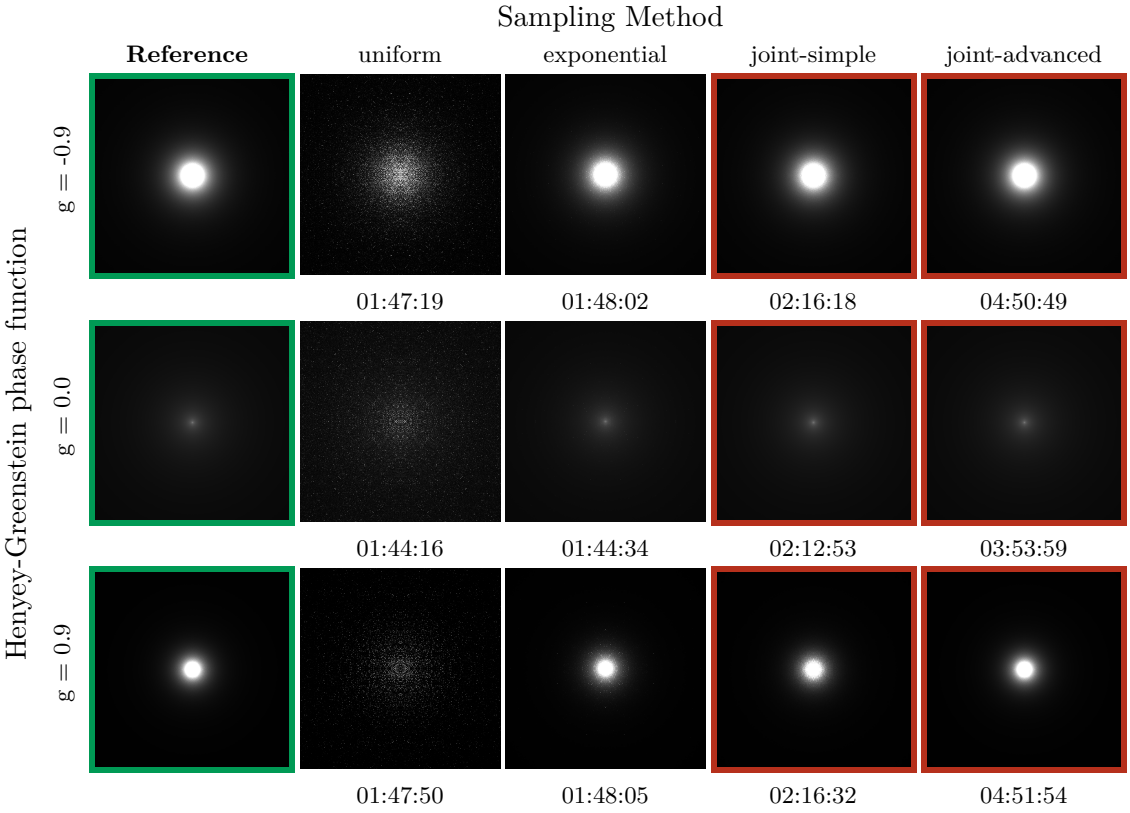


Table 9.24: Comparison of a scene with a point light (unidirectional) with 1 sample per pixel and 1 000 000 light rays per sample.

The tests in Table 9.24 show the results of a scene with a point light, where the light rays are cast unidirectionally. Only one quarter was rendered and repeated for the other quarters, as the other three quarters do not hold any additional information. As is shown in this table, this test is difficult to converge, as the light ray are not a good fit for unidirectional distribution. In this case, the benefits of the advanced joint distribution is therefore even more noticeable, as after 1 million samples, it is the only one that comes close to convergence. Table 9.25 shows the RMSE and the PSNR for both joint distributions.

	simple/advanced	
HG	RMSE	PSNR
-0.9	1.5214	44.4859
0.0	0.2171	54.4806
0.9	2.7353	39.3908

Table 9.25: Comparison of the RMSE and PSNR of Table 9.24.

Convergence Comparison

As can be seen from Table 9.26, Table 9.27 and Table 9.28 where the same scene has been rendered with 1 light ray, 10 light rays and 100 light rays, respectively, the light rays are not a good choice for unidirectional distribution. The single ray segments are still visible after many samples, therefore convergence is slow.

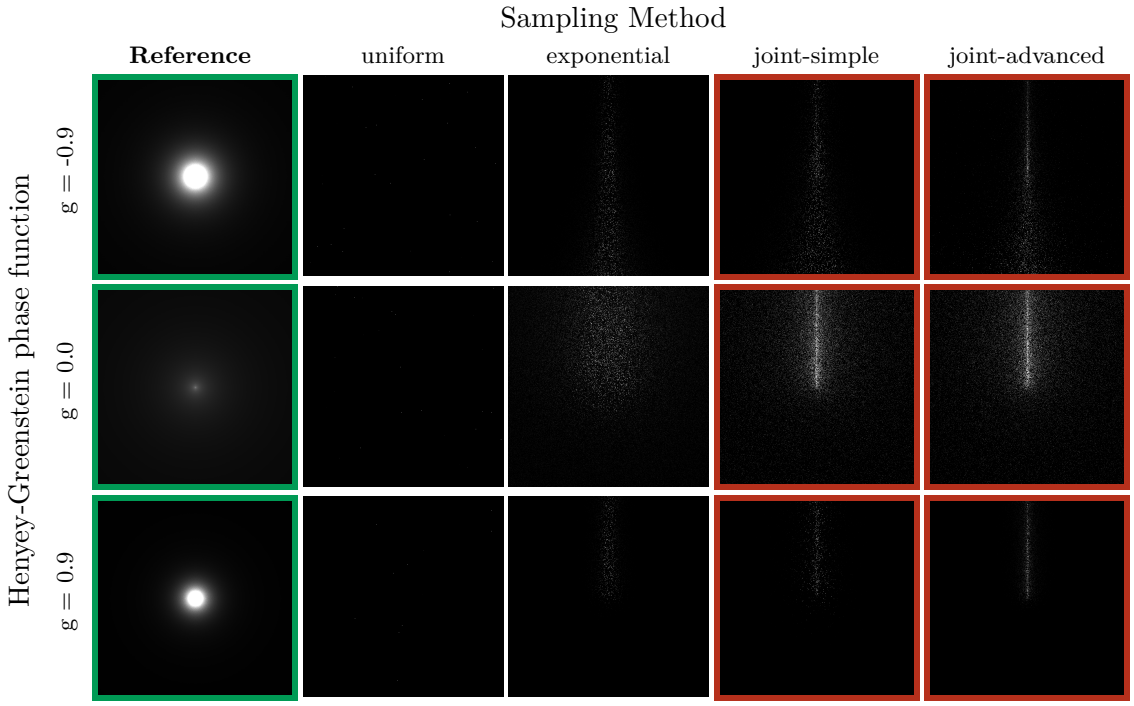


Table 9.26: Comparison of a scene with a point light (unidirectional) with 1 sample per pixel and 1 light ray per sample.

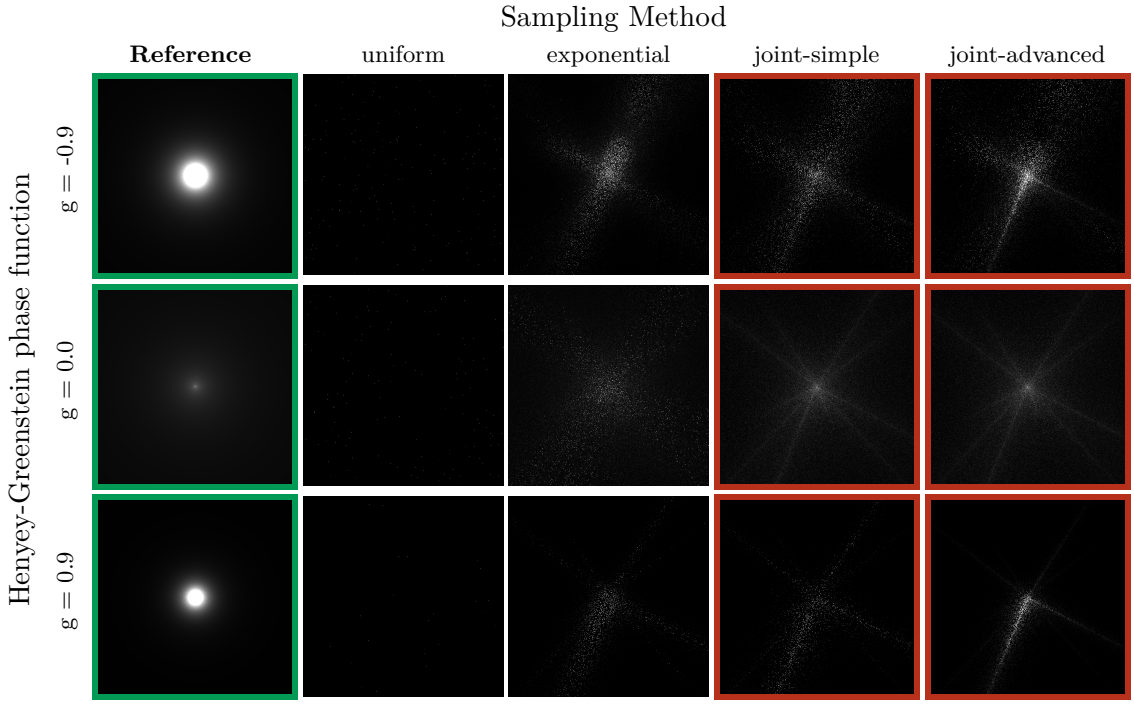


Table 9.27: Comparison of a scene with a point light (unidirectional) with 1 sample per pixel and 10 light rays per sample.

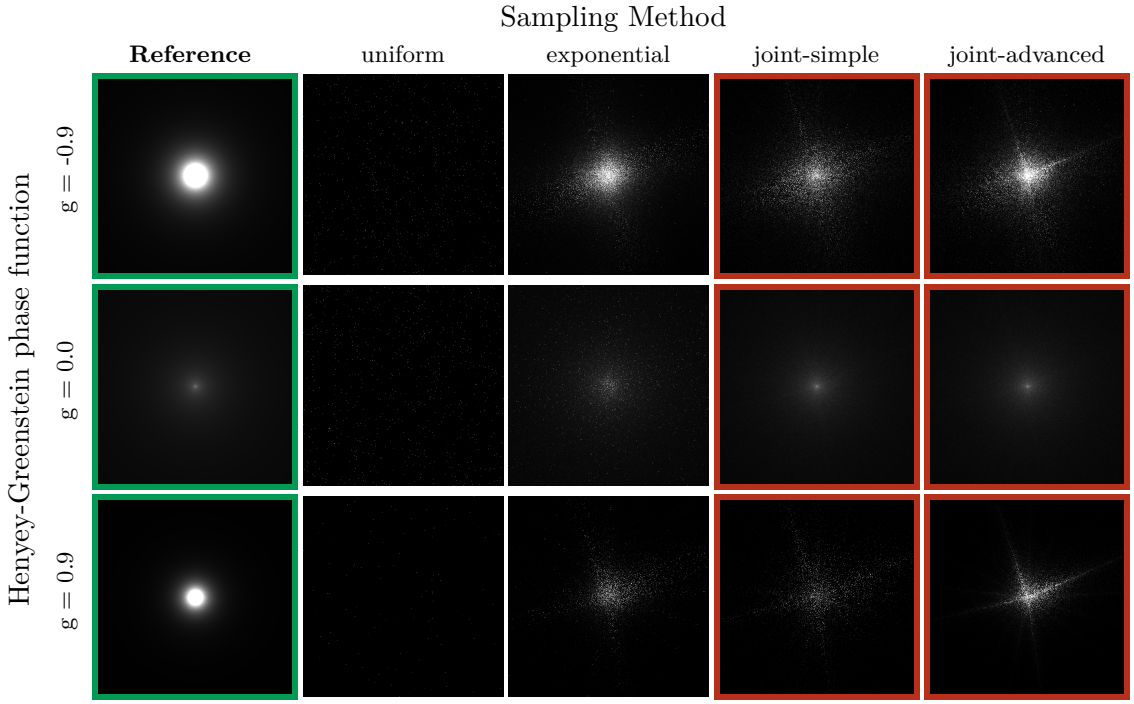


Table 9.28: Comparison of a scene with a point light (unidirectional) with 1 sample per pixel and 100 light rays per sample.

9.5 Comparison to Virtual Ray Lights [NNDJ12b]

Cornell Box - Sampling Differences

This test shows the sampling improvements of the different techniques, a similar test was made by Novák et al. in their paper. On the top, an isotropic medium is rendered by sampling the light ray uniformly and the camera ray equi-angularly (left) and by sampling with the simple joint distribution (right). On the bottom, the simple joint distribution is compared to the advanced joint distribution in anisotropic media. As can be seen from these images, with 100 samples per pixel and 10 light rays per sample, the new approaches perform better compared to their predecessors.

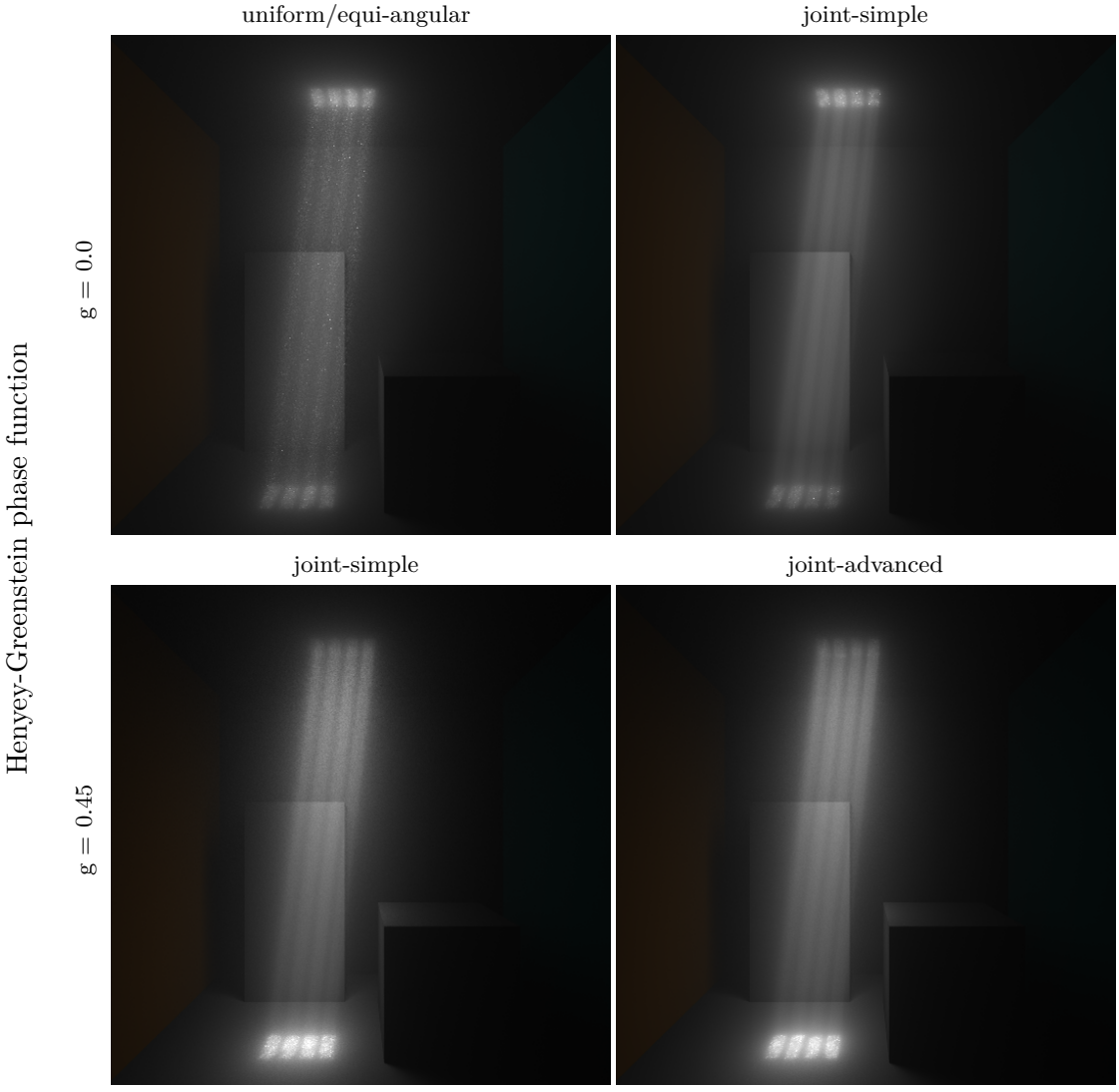


Table 9.29: Comparison of the Cornell box scene with different sampling strategies.

Sphere in Participating Media

Table 9.30 shows a sphere and the volume-to-surface contribution (top), as well as the volume-to-volume contribution (bottom). The left column, contains images from Novák et al. [NNDJ12b], whereas the right column shows our reconstruction of the scene. Due to the limited information, our reconstruction is only vaguely similar, although the issues mentioned for the original figures can clearly be seen. For the volume-to-surface contribution, the hit points of the light rays, as well as there path along the surfaces can clearly be seen. The volume-to-volume examples both show the convergence of the results without any artefacts.

Reference [NNDJ12b] | Our Reconstruction

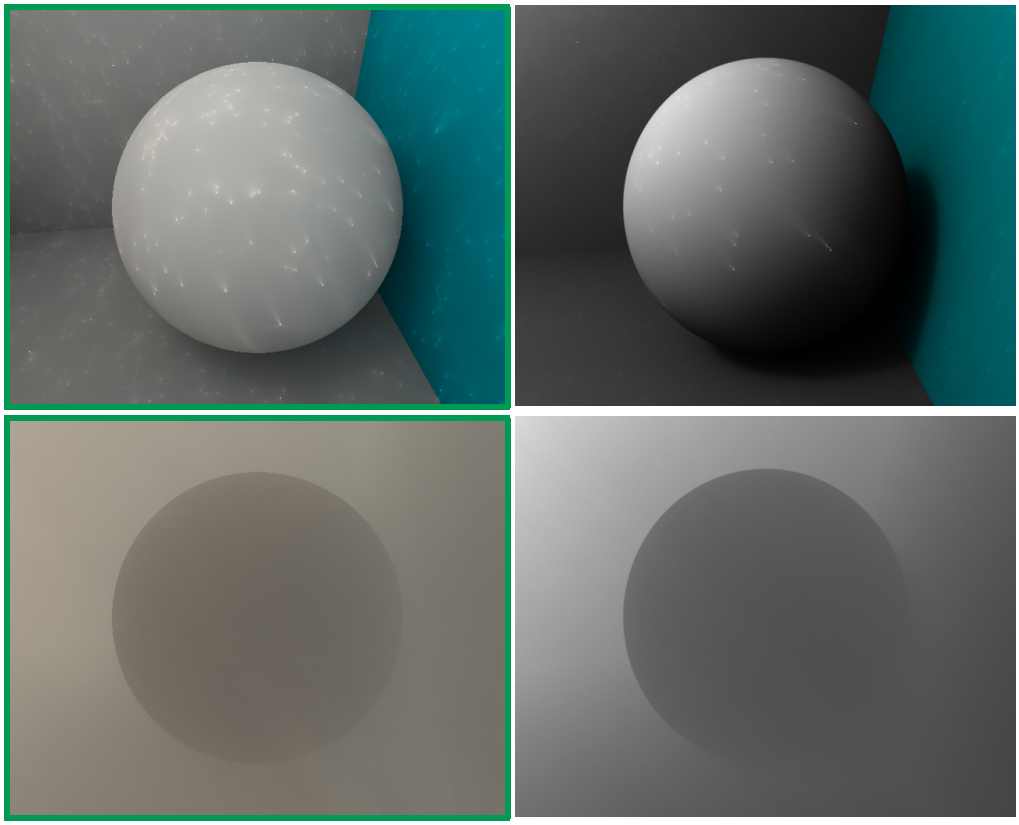


Table 9.30: Comparison of the Virtual Ray Lights sphere in participating media scene [NNDJ12b], to our reconstruction of that scene that was rendered with 100 samples and 100 light rays per sample.

9.6 Comparison Joint Importance Sampling [GKH+13]

Point Light (bidirectional, 10 light rays)

As Georgiev et al. [GKH+13] provide a results with around 10 light rays, this scene is chosen for a comparison. Some images that are discussed earlier are taken for the comparison. The only change that was applied is a horizontal flip. As is shown in Table 9.31 for the simple joint distribution and in Table 9.32 for the advanced joint distribution, the convergence behaves similarly in both cases in the examples provided with 4 samples per pixel. Additionally, the converged cases are in agreement. Other differences can be explained by different scene settings and color corrections.

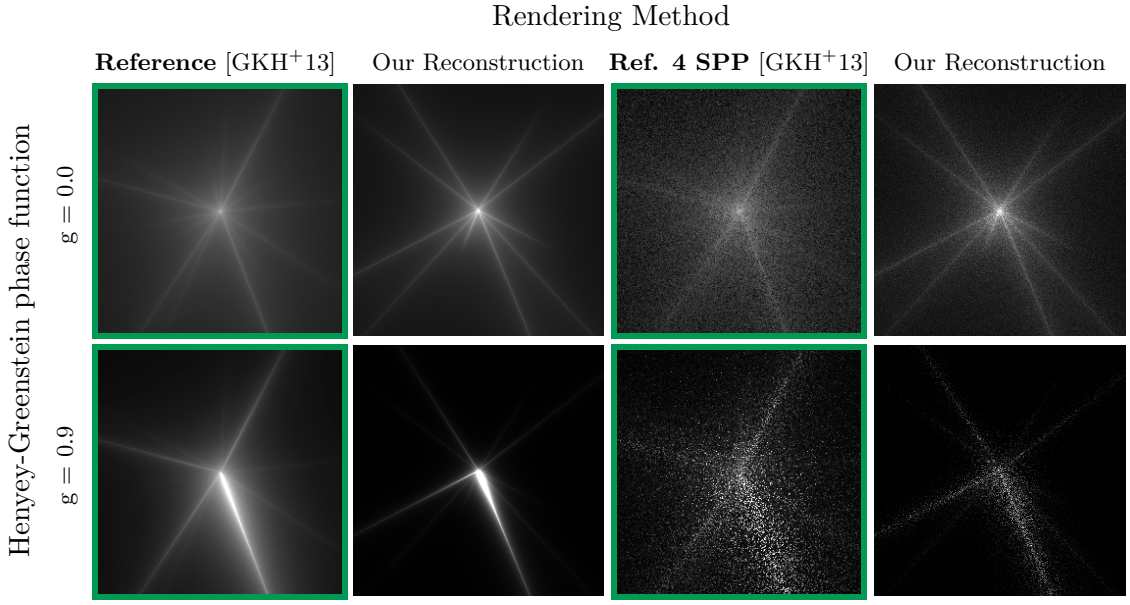


Table 9.31: Comparison of the Virtual Ray Lights bidirectional point light scene [GKH+13], to our reconstruction of that scene with the simple joint distribution sampling method.

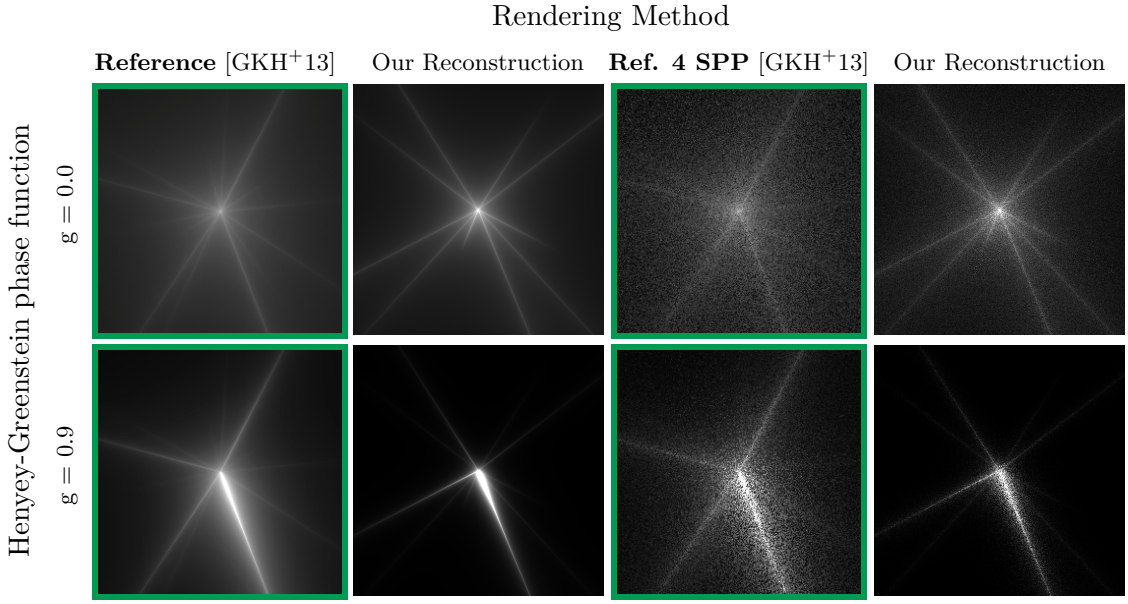


Table 9.32: Comparison of the Virtual Ray Lights bidirectional point light scene [GKH+13], to our reconstruction of that scene with the advanced joint distribution sampling method.

CHAPTER 10

Conclusion

This thesis documents the full implementation of the unbiased progressive method, Virtual Ray Lights for Rendering Scenes with Participating Media. Through the process of this documentation, the starting chapters describe foundations and methods that are of importance in the development of this approach. The analysis of different approaches that use different types of photon tracing show the general trend of increasing the dimensionality to improve the density of the data and the area of the estimators.

For Virtual Ray Lights, camera rays as well as light rays are evaluated to be able to compute the volume-to-volume and the volume-to-surface contribution of the light. A joint importance sampling technique is introduced to make it possible to sample from a shared domain. A simple version already works well for isotropic media, but the variance from anisotropic media cannot be accounted for. Therefore, Novák et al. developed a new technique for anisotropic media to create a probability density function that is proportional to the product of the phase functions, as well as the inverse-squared distance like in the isotropic case. A detailed mathematical derivation of the probability density function, the cumulative density function as well as its inverse, which was absent from the original paper, is a contribution of this thesis, along with the documentation of all functionality used in Virtual Ray Lights from a theoretical standpoint as well as an implementation standpoint.

The implementation of Virtual Ray Lights performed well in our testing scenarios and provided the expected results in all scenes. The method is an improvement for the computation of the volume-to-volume contribution as singularities are greatly reduced with a substantial increase in convergence speed. Many different tests are examined and the comparison to similar results from other publications has proven to be successful. As this implementation was done with educational purposes in mind, it can be followed easily with the help of this thesis and the provided source material. Appendix B provides information on how to access this material as well as further information on preexisting frameworks and contributions.

This thesis was designed to help bring an understanding from the ground up and develop a knowledge on this topic in a theoretical and practical sense. The goal of this thesis is to make it possible to implement Virtual Ray Lights into other projects. Furthermore, a focus has been on providing an understanding of complex importance sampling techniques and their usage in the field of rendering participating media.

APPENDIX A

Monte Carlo Integration & Importance Sampling

When a Monte Carlo method is used for rendering that means that an integral is approximated using Monte Carlo integration. For this approach, first a Monte Carlo estimator has to be constructed to be able to evaluate the integral. This part is described first. The distribution of samples is a key to the performance and convergence of this method. In most cases, a uniform distribution of samples is slow in convergence, therefore, different techniques have been developed to improve sampling. While there are simple improvements like exponential sampling, there are also far more sophisticated ones like multiple importance sampling. This appendix will provide a simple introduction in the terminology of Monte Carlo integration and importance sampling and will explain a few simple methods with examples. The importance sampling methods that are relevant for the implementation of VRL are discussed in Section 7.3.

A.1 Monte Carlo Integration

For Monte Carlo integration, the goal is to approximate the result of a given integral as accurate and fast as possible. The evaluation of this integral can be done by the sampled-mean version of the Monte Carlo integration. The method works as follows: For a given function for which the integral is to be calculated in a defined interval, samples are chosen within this interval to evaluate the function at this point. The result of the evaluation is then adjusted by the probability that this sample is chosen. In the case of a uniform distribution, this probability is constant for all samples the inverse of the interval size. All of these results are summed up and divided by the amount of samples that were generated.

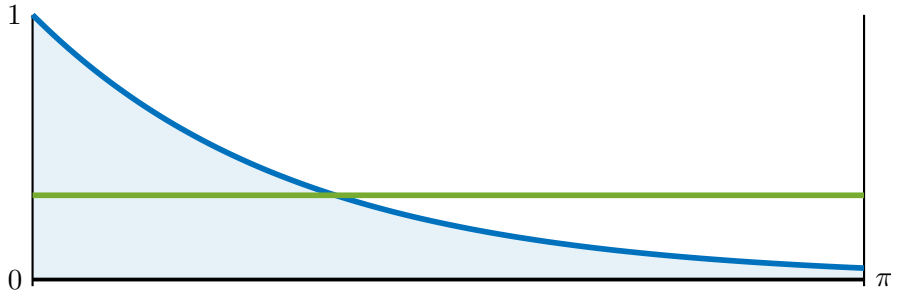


Figure A.1: The function e^{-x} in the interval $[0, \pi]$. This function is used as an example for the distribution of samples and is marked as blue. The green line is the PDF of the uniform distribution in this interval.

This process can be described with

$$F(x) \approx \frac{1}{N} \sum_{N}^{i=1} \frac{f(x)}{\text{pdf}(x)}, \tag{A.1}$$

where the term N of this equation stands for the number of samples, the function $f(x)$ is the function for which the integral should be approximated and the $\text{pdf}(x)$ is the probability of choosing this sample x.

A very simple example of an integral is

$$F(x) = \int_0^{\pi} e^{-x}\, dx, \tag{A.2}$$

which show the integral $F(x)$ of the function $f(x) = e^{-x}$ in the interval $x \in [0, \pi]$. This function is visualized in Figure A.1 as the blue line.

For the evaluation of this integral, uniform random samples are chosen and evaluated. The probability of choosing a sample in the interval $[0, \pi]$ uniformly is $1/\pi$ which can be seen as the green line of Figure A.1. The process for the first sample is described in detail to build a basic understanding of this principle. The following results are then shown in Table A.1. The first randomly chosen number is 2.4719 and the result of $f(2.4719)$ is 0.0844. This result is then divided by the probability $1/\pi$ and divided by the number of samples, which is 1 in this case. The first result is therefore 0.2652.

In Table A.1, the results after various samples is shown. The actual result of the integral is 0.9568 and as can be seen in the last row of this table, after 1 million samples, the error of the evaluation has decreased to 0.0004.

Samples	Error	Result
1	-0.6916	0.2652
10	0.2487	1.2055
100	-0.0014	0.9554
1000	0.0023	0.9591
10000	0.0022	0.9590
100000	-0.0020	0.9548
1000000	0.0004	0.9572

Table A.1: The error and convergence of the Monte Carlo integration of function e^{-x} in the interval $[0, \pi]$ with the uniform sampling.

A.2 Importance Sampling

As sampling with a uniform distribution is, in most cases, suboptimal, other distribution are used for importance sampling. For the understanding of these distributions, a basic understanding of a few fundamental statistical properties are necessary. The main terms for importance sampling are the probability density function (PDF) and the cumulative density function (CDF).

A.2.1 Probability Density Function

The PDF is a function that describes, for every point in the sample space, the relative likelihood for the sample being at the current position. It is often just described as the density of a continuous random variable. To make this concept clearer, Figure A.2 shows an example of the linear PDF $\text{pdf}(x) = x/2$. In this example, at 0.5 the function has a value of 0.25, which means that there is a 25% chance that a sample is chosen is at that position.

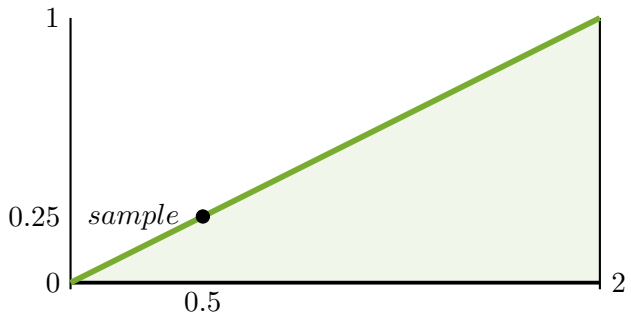


Figure A.2: An example of a linear PDF. The chosen sample at position 0.5 presents a 25% possibility that a sample at this position is chosen with this PDF.

An important property of the PDF is that it must be normalized which means that its integral, the CDF must be exactly 1. That means that in many cases, the CDF must be calculated to divide the PDF by it.

A.2.2 Cumulative Density Function

The CDF is the integral of the PDF. Therefore, it describes the area under the PDF and must always be exactly 1. The CDF to the PDF of the previous figure is shown in Figure A.3 as the orange line. The importance of the CDF comes mainly through its inversion. As the non-inverted CDF creates values in the interval $[0, 1)$ with the input of a value in the sample interval, the inverted CDF, denoted as CDF^{-1}, creates values in sample interval with the input of a value in the interval $[0, 1)$. By acquiring samples through the CDF^{-1}, the samples are distributed by the corresponding PDF. This method of generating random variables in a desired distribution is called inverse transform sampling or Smirnov transform.

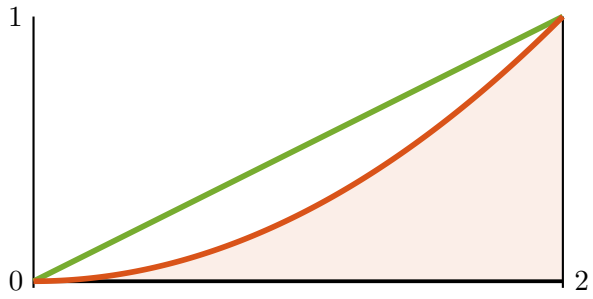


Figure A.3: The PDF of Figure A.2 as a green line with the corresponding CDF marked in orange.

With the knowledge gained in this section, the previous example of the integral of the function $f(x) = e^{-x}$ can now be expanded to be sampled with a more fitting distribution of samples. As can be seen in Figure A.4 as the blue line, the function is bigger at the start then at the end, it would therefore make sense to focus on the start of the function more than on the end. Therefore, the function $\frac{1}{(x+1)^2}$ is normalized in this interval, to create a

$$\text{pdf}(x) = \frac{\pi + 1}{\pi \cdot (x + 1)^2}, \tag{A.3}$$

with the corresponding

$$\text{cdf}(x) = -\frac{\pi + 1}{\pi \cdot (x + 1)} + \frac{\pi + 1}{\pi} \tag{A.4}$$

and

$$\text{cdf}(\xi)^{-1} = \frac{\pi \cdot \xi}{-\pi \cdot \xi + \pi + 1}. \tag{A.5}$$

This PDF is the green line of Figure A.4.

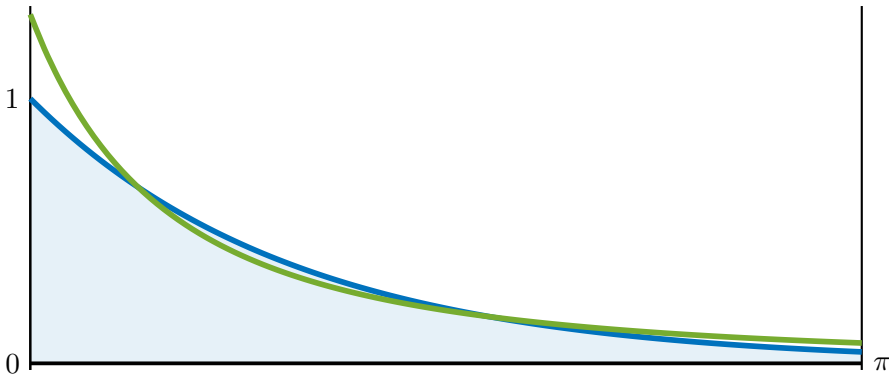


Figure A.4: The function e^{-x} in blue with a PDF as the green line, which fits the distribution better than a uniform distribution.

Samples	**Error**	**Result**
1	0.1051	1.0619
10	-0.0215	0.9353
100	-0.0035	0.9533
1000	-0.0004	0.9564
10000	-0.0001	0.9566
100000	0.0000	0.9567
1000000	0.0000	0.9567

Table A.2: The error and convergence of the Monte Carlo integration of function e^{-x} in the interval $[0, \pi]$ with the sampling of Equation A.3 which is visualized in Figure A.4.

The results for this evaluation with various sample numbers is shown in Table A.2. As can be derived from theses results, the convergence is faster than in the uniform sampled case, as the sampling distribution matches the function better. Even though this function is relatively evenly distributed, there are many cases where most of the target interval is not of major importance for the evaluation. Just expanding this example to a larger interval would decrease the performance of the uniform sampling marginally.

A.3 Importance Sampling in Rendering

To put all of this theory into the context of rendering, this example shows how sampling can be used for a light ray in a participating medium. In Figure A.5, a light ray can be seen. The light fades as more distance is traveled and this ray contributes less and less as the power of the light ray is affected by the transmission. When calculating an integral along this ray, it would therefore be advisable to use a PDF that is proportional to this property. The figure shows a linear PDF (green) and the actual underlying function (blue, flipped horizontally). As is shown in this illustration, a linear PDF would still take many samples at positions that are not relevant in this scenario.

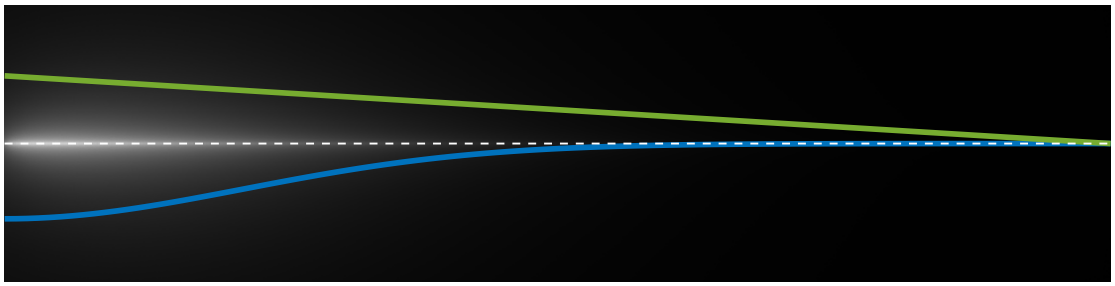


Figure A.5: A light ray in a participating media. An example of a linear PDF at the top (green) and the actual underlying function at the bottom (blue, flipped horizontally).

APPENDIX B

Source Material

The source material is available in two versions. The first version is a command line program that contains solely the implementation of Virtual Ray Lights. The second version is the implementation of Virtual Ray Lights, as well as Progressive Photon Mapping, into the educational path tracer Smallpaint by Károly Zsolnai-Fehér [ZF18b]. For Smallpaint, a user interface that provides progressive updates was created in the process of this thesis. The first option provides better adaptability as where the second option provides the better usability.

B.1 Command Line Program

The source code for the implementation of Virtual Ray Lights in a command line program is provided as a Microsoft Visual Studio 2017 solution. Before building, the option for OpenMP has to be set under `Project -> Properties -> C/C++ -> Language -> Open MP Support`. The project can then be built and the program `VirtualRayLight.exe`, that is either in the `Debug` or `Release` folder, can be moved anywhere.

The program can also be created without Microsoft Visual Studio with the command `g++ vrl.cpp -O3 -std=gnu++0x -fopenmp -static-libgcc -static-libstdc++`.

Multiple arguments can be provided as input to this program. A full list of commands can be seen with the command `VirtualRayLights.exe -h`.

B.2 Smallpaint

The source code for the the educational path tracer Smallpaint [ZF18b], which contains the implementations of Progressive Photon Mapping and Virtual Ray Lights, is provided as a Qt solution. The binaries for Windows and Linux are provided in the folders `binary_win32` and `binary_linux`. The Windows implementation works as long as Microsoft Visual Studio or Qt is installed on the system. When rebuilding this project, the instructions to do so are provided in the `README.txt` file.

The first contribution to Smallpaint is the user interface which makes progressive updates possible, as well as providing an improved usability. Furthermore, additions to the approaches can be made by following the guide `HOWTOAddYourOwnRenderer.txt`.

The second contribution is the implementation of Progressive Photon Mapping which was mentioned in Section 4.2. For this implementation, the source code by Toshiya Hachisuka was adapted to fit into the framework.

The third and main contribution of this thesis is the implementation of Virtual Ray Lights into this framework, which was discussed in the previous sections. As can be seen from Figure B.1, many properties can be adjusted. For the importance sampling, all options that were discussed in the previous chapters are available. Furthermore, all test scene can be selected from a menu and the settings are adjusted accordingly when doing so. Therefore, all examples that are provided can be recreated.

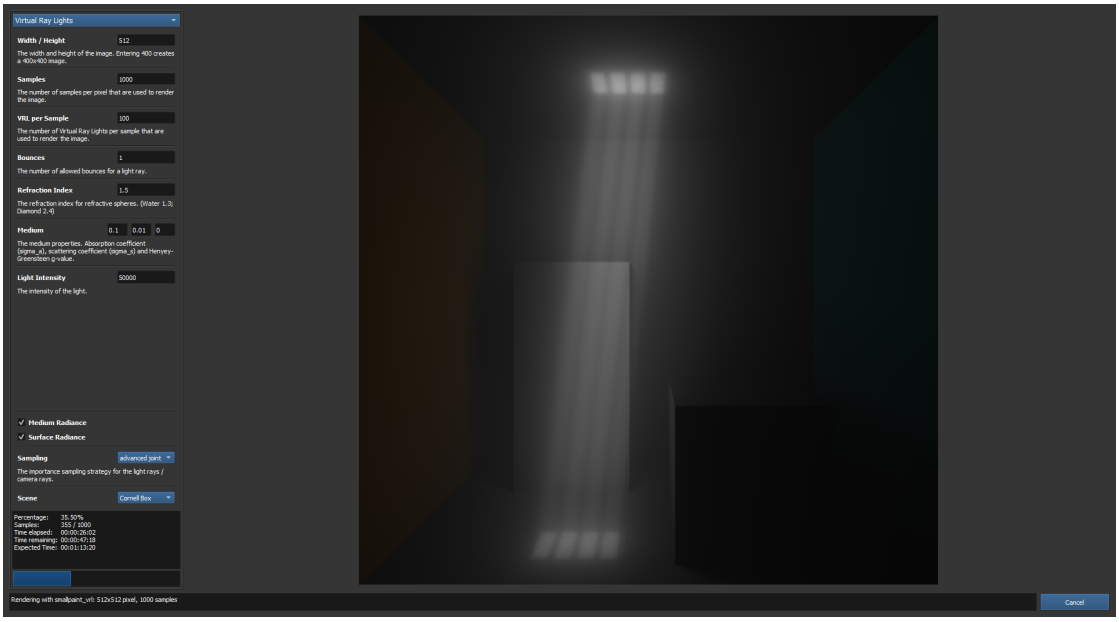


Figure B.1: A screenshot of the user interface of Smallpaint [ZF18b]. In this example the implementation of Virtual Ray Lights can be seen.

List of Figures

List of Tables

List of Algorithms

Acronyms

BPT Bidirectional Path Tracing. 46, 55, 56, 144

BRDF bidirectional reflectance distribution function. 1, 6, 8, 9, 26, 28, 32, 37, 51, 62, 87, 88

BRE Beam Radiance Estimate. 2, 45, 48, 49, 52–56, 61, 144

BTDF bidirectional transmission distribution function. 9

CDF cumulative density function. 67, 72–75, 89, 90, 92, 133, 137, 138, 145

HG Henyey-Greenstein phase function. 100, 112, 118, 124

JIS Joint Importance Sampling. xvi, 3, 59, 130, 131

MIS multiple importance sampling. 46, 56, 135

PB Photon Beams. 2, 52–56, 60, 144

PDF probability density function. 42, 43, 65–69, 72–76, 86–90, 92, 93, 133, 136–140, 145

PM Photon Mapping. 1, 2, 6, 7, 23, 24, 28–34, 45, 47, 49, 51, 53, 143, 144

PP Photon Planes. 60

PPB Progressive Photon Beams. 46, 53, 58, 61, 144

PPM Progressive Photon Mapping. 23, 30, 32–34, 53, 58, 100, 141, 142, 144

PSNR peak signal-to-noise ratio. 3, 100, 112, 118, 124, 147, 148

PV Photon Volumes. 60

RMSE root-mean-square error. 3, 100, 112, 118, 124, 147, 148

RRT Recursive Ray Tracing. 1, 6, 7, 13, 19–21, 143

RT Ray Tracing. 6, 13, 14, 20, 23–25, 27, 28, 30

SPPM Stochastic Progressive Photon Mapping. 23, 33, 34, 144

UPBP Unifying Points, Beams and Paths. 46, 55, 56, 144

VBL Virtual Beam Lights. 2, 46, 58, 59, 144

VPL Virtual Point Lights. 2, 45, 46, 50, 51, 56–59, 61, 62, 76, 77, 144, 145

VPM Volumetric Photon Mapping. 2, 45, 47–49, 52, 55, 56, 144

VPT Volumetric Path Tracing. 98

VRL Virtual Ray Lights. xvi, 1–4, 6, 38, 46, 56–59, 61, 62, 64, 76, 77, 128–131, 133–135, 141–145, 148, 149

VSL Virtual Spherical Lights. 45, 46, 50, 144

Bibliography

[AA92] Larry C Andrews and Larry C Andrews. *Special functions of mathematics for engineers*. McGraw-Hill New York, 1992.

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45. ACM, 1968.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BJ17] Benedikt Bitterli and Wojciech Jarosz. Beyond points and beams: Higher-dimensional photon samples for volumetric light transport. 2017.

[BSS93] Philippe Blasi, Bertrand Saec, and Christophe Schlick. A rendering algorithm for discrete volume density objects. In *Computer Graphics Forum*, volume 12, pages 201–210. Wiley Online Library, 1993.

[Buc95] Anthony Bucholtz. Rayleigh-scattering calculations for the terrestrial atmosphere. *Applied Optics*, 34(15):2765–2773, 1995.

[Cha13] Subrahmanyan Chandrasekhar. *Radiative transfer*. Courier Corporation, 2013.

[CPC84] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 137–145. ACM, 1984.

[CT82] Robert L Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (TOG)*, 1(1):7–24, 1982.

[DCH88] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *ACM Siggraph Computer Graphics*, volume 22, pages 65–74. ACM, 1988.

[Dor95] Julie Dorsey. Radiosity and global illumination. *The Visual Computer*, 11(7):397–398, 1995.

[FW59] Kenneth W Ford and John A Wheeler. Semiclassical description of scattering. *Annals of Physics*, 7(3):259–286, 1959.

[GKH+13] Iliyan Georgiev, Jaroslav Krivanek, Toshiya Hachisuka, Derek Nowrouzezahrai, and Wojciech Jarosz. Joint importance sampling of low-order volumetric scattering. *ACM Trans. Graph.*, 32(6):164–1, 2013.

[Gla89] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.

[GS10] Iliyan Georgiev and Philipp Slusallek. Simple and robust iterative importance sampling of virtual point lights. In *Eurographics (Short Papers)*, pages 57–60, 2010.

[Hea21] Thomas Little Heath. *A history of Greek mathematics*, volume 1. Clarendon, 1921.

[Hec87] Eugene Hecht. *Optik*. McGraw-Hill, 1987.

[Hen01] Klaus Hentschel. Das brechungsgesetz in der fassung von snellius. *Archive for history of exact sciences*, 55(4):297–344, 2001.

[HG41] Louis G Henyey and Jesse L Greenstein. Diffuse radiation in the galaxy. *The Astrophysical Journal*, 93:70–83, 1941.

[HJ09] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Trans. Graph.*, 28(5):141:1–141:8, December 2009.

[HKWB09] Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. Virtual spherical lights for many-light rendering of glossy scenes. In *ACM Transactions on Graphics (TOG)*, volume 28, page 143. ACM, 2009.

[HOJ08] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transactions on Graphics (TOG)*, 27(5):130, 2008.

[HSK89] John C Hart, Daniel J Sandin, and Louis H Kauffman. Ray tracing deterministic 3-d fractals. In *ACM SIGGRAPH Computer Graphics*, volume 23, pages 289–296. ACM, 1989.

[ICG86] David S Immel, Michael F Cohen, and Donald P Greenberg. A radiosity method for non-diffuse environments. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 133–142. ACM, 1986.

[Jar08] Wojciech Jarosz. *Efficient Monte Carlo methods for light transport in scattering media*. University Of California, San Diego, 2008.

[JC98] Henrik Wann Jensen and Per H Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320. ACM, 1998.

[JCKS02] Henrik Wann Jensen, Per H Christensen, Toshiaki Kato, and Frank Suykens. A practical guide to global illumination using photon mapping. *SIGGRAPH 2002 Course Notes CD-ROM*, 2002.

[Jen96a] Henrik Wann Jensen. Global illumination using photon maps. *Rendering techniques*, 96:21–30, 1996.

[Jen96b] Henrik Wann Jensen. The photon map in global illumination. 1996.

[Jen01] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. AK Peters, Ltd., 2001.

[JNSJ11] Wojciech Jarosz, Derek Nowrouzezahrai, Iman Sadeghi, and Henrik Wann Jensen. A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Transactions on Graphics (TOG)*, 30(1):5, 2011.

[JNT+11] Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike Sloan, and Matthias Zwicker. Progressive photon beams. *ACM Transactions on Graphics (TOG)*, 30(6):181, 2011.

[JWW76] J H_ Joseph, WJ Wiscombe, and JA Weinman. The delta-eddington approximation for radiative flux transfer. *Journal of the Atmospheric Sciences*, 33(12):2452–2459, 1976.

[JZJ08] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. In *ACM SIGGRAPH 2008 classes*, page 3. ACM, 2008.

[Kaj86] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[Kel97] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[KF11] Christopher D Kulla and Marcos Fajardo. Importance sampling of area lights in participating media. In *SIGGRAPH Talks*, page 55, 2011.

[KGH+14] Jaroslav Křivánek, Iliyan Georgiev, Toshiya Hachisuka, Petr Vévoda, Martin Šik, Derek Nowrouzezahrai, and Wojciech Jarosz. Unifying points, beams, and paths in volumetric light transport simulation. *ACM Transactions on Graphics (TOG)*, 33(4):103, 2014.

[LW93] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. 1993.

[Man77] Benoit B Mandelbrot. *Fractals*. Wiley Online Library, 1977.

[Mar12] Steve Marschner. Realistic image synthesis, 2012. http://www.cs.cornell.edu/courses/cs6630/2015fa/index.shtml.

[NGHJ18] Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. Monte carlo methods for volumetric light transport simulation. In *Computer Graphics Forum*, volume 37, pages 551–576. Wiley Online Library, 2018.

[Nic65] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied optics*, 4(7):767–775, 1965.

[NNDJ12a] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Progressive virtual beam lights. In *Computer Graphics Forum*, volume 31, pages 1407–1413. Wiley Online Library, 2012.

[NNDJ12b] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Virtual ray lights for rendering scenes with participating media. *ACM Transactions on Graphics (TOG)*, 31(4):60, 2012.

[Nov14] Jan Novák. *Efficient Many-Light Rendering of Scenes with Participating Media*. PhD thesis, Karlsruhe Institute of Technology", May 2014.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[Sch94] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246. Wiley Online Library, 1994.

[SIMP06] Benjamin Segovia, Jean Claude Iehl, Richard Mitanchey, and Bernard Péroche. Bidirectional instant radiosity. In *Rendering Techniques*, pages 389–397, 2006.

[SIP07] Benjamin Segovia, Jean Claude Iehl, and Bernard Péroche. Metropolis instant radiosity. In *Computer Graphics Forum*, volume 26, pages 425–434. Wiley Online Library, 2007.

[Vév15] Petr Vévoda. Robust light transport simulation in participating media. 2015.

[VG97] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76. ACM Press/Addison-Wesley Publishing Co., 1997.

[WBS03] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Interactive global illumination in complex and highly occluded environments. In *Rendering Techniques*, pages 74–81, 2003.

[Whi79] Turner Whitted. An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics*, volume 13, page 14. ACM, 1979.

[WJZ95] Lihong Wang, Steven L Jacques, and Liqiong Zheng. Mcml—monte carlo modeling of light transport in multi-layered tissues. *Computer methods and programs in biomedicine*, 47(2):131–146, 1995.

[ZF18a] Károly Zsolnai-Fehér. Tu wien rendering course, 2018. https://users.cg.tuwien.ac.at/zsolnai/gfx/rendering-course/.

[ZF18b] Károly Zsolnai-Fehér. Smallpaint: A global illumination renderer, 2018. https://users.cg.tuwien.ac.at/zsolnai/gfx/smallpaint/.