university of groningen

faculty of science and engineering

Visualizing Scattering in Virtual Ray Tracer

Jasper Trooster

Bachelor's Thesis
July 2025
Primary Supervisor: Steffen Frey
Secondary Supervisor: Jiří Kosinka

Abstract

Scattering is a vital component of realistic volumetric ray tracing in modern-day graphics. It enables accurate rendering of light interaction with volumetric media like clouds and smoke, by modeling the light bouncing between the particles they consist of. Current tools visualize the basic ray tracing process, but none are extensive enough to cover scattering. Only one, *Virtual Volume Raycaster* (VVRT), offers an interactive visualization of volumetric rendering, but from a scientific visualization perspective.

This thesis presents an extension of VVRT that supports an educative visualization of the concepts of scattering. It includes two algorithms: single and multiple scattering, implemented using ray marching and Monte Carlo methods respectively. These algorithms are applied on rays traversing the scene, illustrating their interaction with configurable volumetric objects. In combination with a tutorial, the user is guided visually and textually through the fundamental concepts. By enabling them to see the effects of the adjustments they make to the volume parameters, the tool offers an intuitive and engaging representation of the theory of scattering. This is confirmed by a user study, ensuring the extension remains valuable for students of computer graphics and lays a solid foundation for future work to build upon.

Contents

Chapter 1

Introduction

Computer graphics is a rapidly developing field of ever-increasing importance, with budgets and demand for realism in visual media rising year by year. In many applications, simulations and visualizations of the complex interplay of physical systems are required. Within the field, there are different approaches to rendering. The most prevalent are rasterization and ray tracing. While the latter allows for more realistic images because it simulates optical effects like reflection and refraction, it comes with a heavy performance impact. Because of this, historically rasterization has been used more widely, but recent advances in hardware support for ray tracing make it more feasible to use in real-time applications [1].

Rendering techniques for simple surface meshes are well-studied and widely taught. However, the space *between* those meshes in reality oftentimes plays an important role. Countless minuscule particles together make up *participating media* like flames, smoke, dust or clouds (Figure 1.1), as well as the atmosphere itself. Still, these media are only simulated by by advanced rendering engines, using techniques usually reserved to only advanced students.

The details of volumetric ray tracing are mathematically complex. However, the fundamental concepts can be taught effectively by visualizing the rays as they interact with the volumes in three-dimensional space. Therefore, it is beneficial to students of computer graphics to have a tool that visualizes the essential concepts, and to ease the process of grasping the underlying theory. This is achieved by providing a user friendly software application that supports adjusting the scene and its simulation parameters, in order to build an intuitive understanding of the developed theory and algorithms.

In the following chapters, this thesis answers the following Research Questions:

RQ1 What aspects of and concepts in the theory of scattering should be taught?

RQ2 How can these concepts best be represented visually?

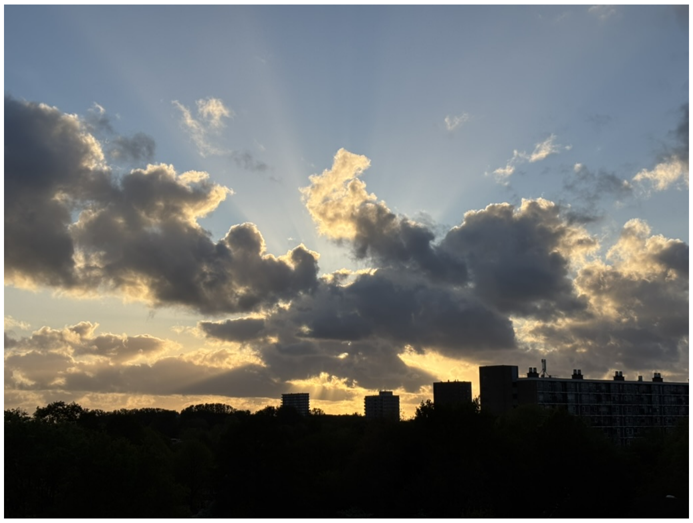


Figure 1.1: Clouds with lit edges and crepuscular rays.

RQ3 In what ways do we allow users to interact and adjust parameters?

RQ4 How can the tutorial teach these concepts most effectively?

RQ5 How valuable is the tool for future students of computer graphics?

In the next chapter, Chapter 2, we examine current related work. In Chapter 3, the relevant theory is explained, leading to an answer to RQ1. RQ2, RQ3 and RQ4 are covered in Chapter 4 on visualization, and the implementation is described in Chapter 5. Lastly, we describe and discuss the results of a user study in Chapter 6, and conclude in Chapter 7.

Chapter 2

Related Work

This chapter details prior work related to our aim of interactively visualizing scattering effects. First, existing applications that offer visual ray tracing capabilities are described, after which we focus on VVRT, the most advanced of them.

2.1 Prior Applications

Perhaps the first attempt at a behind-the-scenes ray tracing visualizer was by Russell in 1999 [2]. His application allows the user to see the ray-tracing process visualized interactively, by tracing rays as dotted lines through a scene containing the viewpoint. Figure 2.1 shows how clicking on a specific pixel in the render view draws lines through the scene, reflecting off the mirror and refracting in the spheres.

Another application developed to make understanding practical ray tracing easier to grasp for students is *Rayground* [3]. It is an online tool that introduces the underlying concepts in a shader-based programming interface, while simplifying the complex code that is typical of ray tracing implementations. Additionally, it offers a discussion on how it could enhance computer graphics courses. However, the ability to fly around the scene and inspect rays as they traverse it remains vital to a thorough and intuitive understanding of the concepts, and Rayground does not support this. Stewart partially addressed this in his thesis [4], developing a teaching tool of his own with features such as a preview of the final image and animated rays (Figure 2.2). However, this implementation too is flawed, since the objects in the scene and their properties cannot be adjusted.

Various other ways of interaction with ray tracing have been considered, such as through a virtual reality (VR) application. Eriksen created a VR environment including a so-called *ray gun*, with which the user can shoot rays anywhere and immediately see their interaction with the objects around them

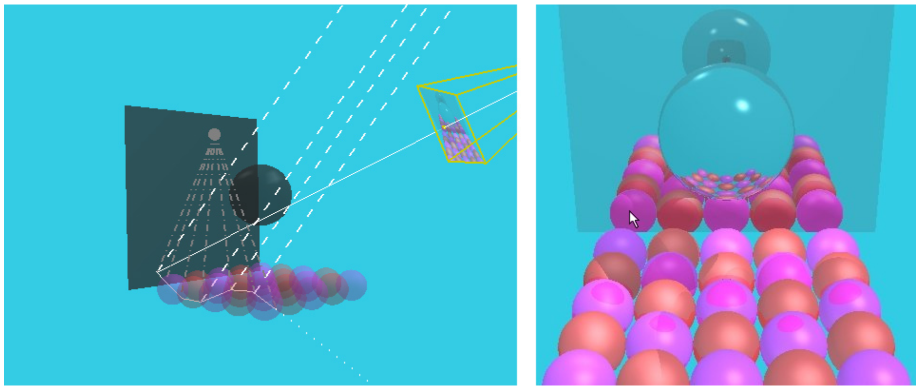


Figure 2.1: Rays can be fired into the scene through the mouse pointer by clicking [2].

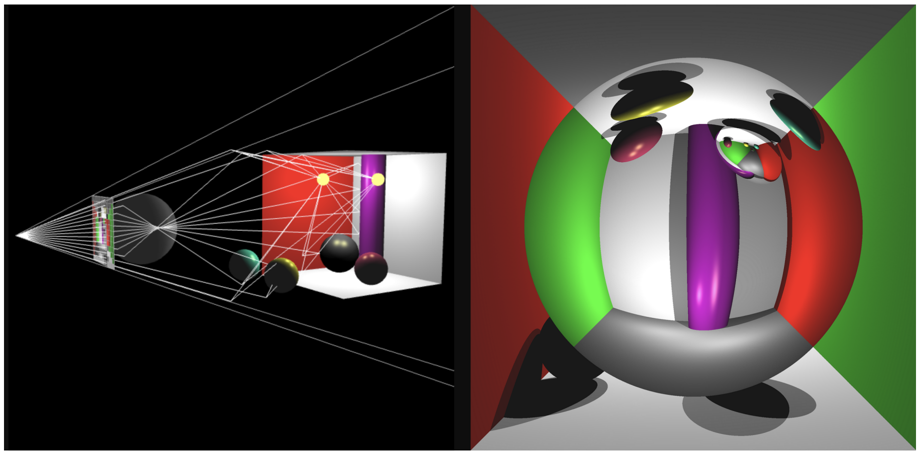
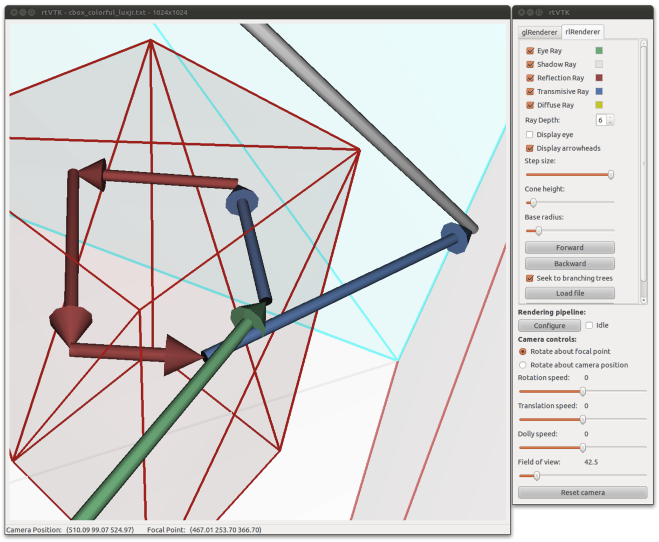


Figure 2.2: Screenshot of Stewart's visualization application interface, showing refraction of rays through a sphere [4].



[✓] Eye Ray 🟩
[✓] Shadow Ray ⬜
[✓] Reflection Ray 🟥
[✓] Transmisive Ray 🟦
[✓] Diffuse Ray 🟨
Ray Depth: 6
[ ] Display eye
[✓] Display arrowheads
Step size:
Cone height:
Base radius:
[✓] Seek to branching trees
Rendering pipeline:
[ ] Idle
Camera controls:
● Rotate about focal point
○ Rotate about camera position
Rotation speed: 0
Translation speed: 0
Dolly speed: 0
Field of view: 42.5

Camera Position: (510.09 99.07 524.97) Focal Point: (467.01 253.70 366.70)

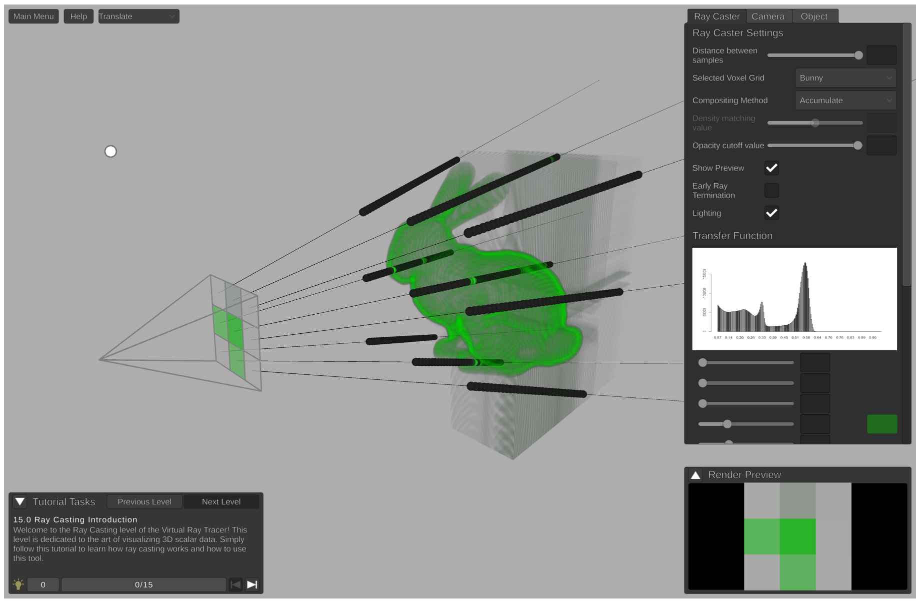
Figure 2.3: User interface of rtVTK [7].

[5]. This added dimension of control results in a very intuitive and engaging educational experience. A similar VR tool called *RayTracerVR* was developed, instead aiming at teaching ray tracing by stepping through a general ray tracing algorithm and tracking its variables [6].

In 2012, Gribble et al. created a graphics toolkit that helps in debugging existing ray tracing applications, named *Ray Tracing Visualization Toolkit* [7]. It offers a ray state recording library that integrates into a renderer, and a GUI that shows an interactive animation of a ray traversing the scene (Figure 2.3). This tool is unlike most in that instead of providing a native rendering engine, it integrates in already existing code to provide its functionality. Therefore, it could be used to teach realistic volume rendering by applying it to an existing physically-based volumetric ray caster, and showing the paths of rays interacting with a volume. However, limitations such as the complexity of the library layer over an already complex ray tracer make it less attractive than Virtual Volume Raycaster, described next.

2.2 VRT and VVRT

In 2022, a program named *Virtual Ray Tracer* was created by Verschoore de la Houssaije and Wezel [8]. It allows users to interact with the ray tracing process in real time, allowing them to see instantaneously how changes made to the scene or camera influence the final result. This version laid the groundwork for



Distance between samples
Selected Voxel Grid: Bunny
Compositing Method: Accumulate
Density matching value
Opacity cutoff value
Show Preview ✓
Early Ray Termination
Lighting ✓

15.0 Ray Casting Introduction
Welcome to the Ray Casting level of the Virtual Ray Tracer! This level is dedicated to the art of visualizing 3D scalar data. Simply follow this tutorial to learn how ray casting works and how to use this tool.

Figure 2.4: Interface of the current version of Virtual Volume Raycaster [13]. Spheres along primary rays represent scattering events.

Virtual Ray Tracer known today. It has gone through multiple iterations over the following years, culminating in VRT 2.0, published in 2023 [9]. Firstly, aiming to make the tool more engaging and easier to learn, Blok created a tutorial with levels and a step-by-step guide [10]. Following this, Rosema and Yilmaz [11, 12] converted the program into a web and mobile application and improved performance of the tool in order to be more capable and accessible, even on non-powerful hardware. Most importantly for our interest of scattering, Van der Wal and Blesinger [13] added support for visualizing rendering volumetric objects using direct volume rendering, naming their application *Virtual Volume Raycaster* (VVRT). The accompanying tutorial explains how rays interact with volumes, phase functions and compositing methods. The volume is rendered live in the scene. The user interface for this version of VVRT is shown in Figure 2.4.

However, even this advanced educational application does not offer support for the visualization of light propagation in volumes. On the contrary: when taking its physical equivalent, it only deals with absorption and emission. This may be enough for a simple demonstration of volumetric rendering, but is far from being realistic. Thus, the addition of an interactive physically-based volume rendering visualization to this tool is the logical continuation of its educational capabilities.

Chapter 3

Background

Volumetric ray tracing is a vast topic. It comprises similar challenges as traditional ray tracing, with its own set of effects to approximate. In this chapter, we go over the necessary theory and background in order to understand Chapter 4 and Chapter 5. First, we describe the real life processes that we aim to simulate, for which we derive mathematical models, including the volumetric counterpart of the well-known rendering equation. Lastly, we provide two methods that approximate this equation, by numerical and Monte Carlo integration.

3.1 Reality

In real life, photons emerge from light sources, like the Sun, car headlights or ceiling lamps. As they travel, they undergo various physical effects: they can be absorbed by objects, reflected off surfaces or refract, as they pass through different materials. Only a very small number actually makes it to the eye. Simulating these effects realistically would be infeasible due to the sheer number of photons to be simulated. Therefore, in computer graphics, we make simplifying assumptions that approximate reality to a model we *can* compute. In this case, instead of tracing the path of every single photon, we trace the rays *backwards*, starting at the viewpoint (see Figure 3.1).

This is the traditional ray tracing approach. One ray is cast through each of the pixels of the screen, determining the color of the corresponding pixel. At the point where a ray intersects a geometric object, a light ray is traced towards every light source in the scene to determine its contribution. Additionally, reflection rays are cast recursively in the case of a reflective material like a mirror, and refraction rays in a refractive material like glass (see Figure 3.2). All of these contributions are summed to obtain the pixel color.

In a simple ray tracer, this approximation will suffice. However, in reality, there are quite some effects that we cannot simulate with this approach, as mentioned earlier. Simplifications exist, and have been applied extensively,

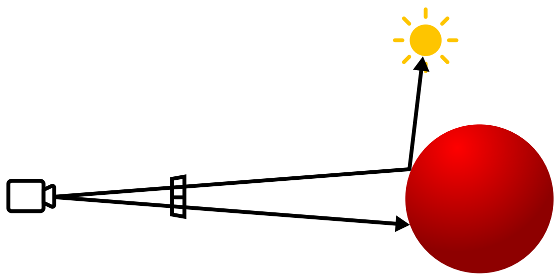


Figure 3.1: Ray tracing pixels of a two-pixel screen. The lower ray cannot reach the light as it is blocked by the sphere.

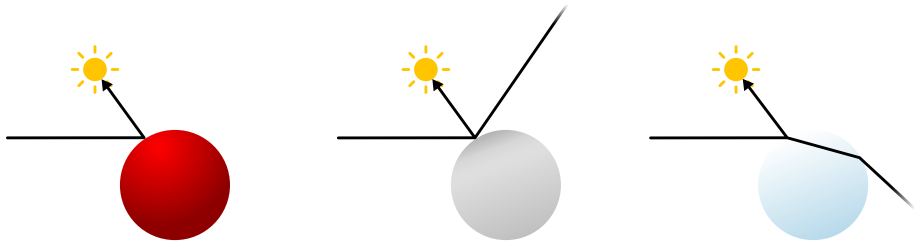


Figure 3.2: Different material types (opaque, mirror, transparent) and their interaction with rays.

especially in real-time applications. For instance, clouds can be modeled with billboards or meshes, fog can be based on the depth buffer and water can be a displaced texture. However, for realistic results in all situations, we turn to techniques that simulate the scattering of light inside volumes more accurately.

Before we do that, we need to know what happens to a photon inside a medium. Participating media consist of countless minuscule particles, like water droplets for clouds and fog, and soot for both a flame and its smoke. A photon can enter this medium and hit a particle, and depending on the properties of that particle, it either gets absorbed by it, or changes direction by reflecting and refracting. Because the particles are so small and there are so many of them, after such an event, it's almost impossible to determine exactly in which direction a photon will end up going. Thus, the light appears to scatter in a random direction, which is why this phenomenon is called *scattering*.

3.2 Modeling Reality

Before we can accurately simulate the aforementioned effects, we need to form a mathematical model. To do this, we assume we have a beam of light traveling across a medium, as in Figure 3.3.

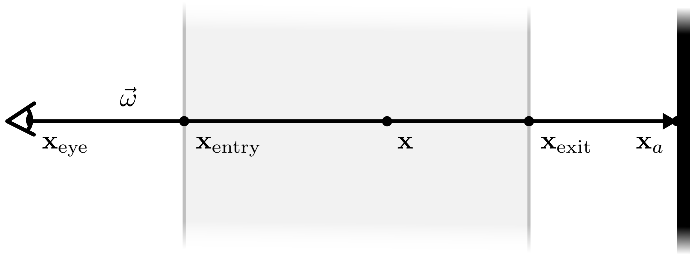

$\vec{\omega}$
$\mathbf{x}_{\text{eye}}$
$\mathbf{x}_{\text{entry}}$
$\mathbf{x}$
$\mathbf{x}_{\text{exit}}$
$\mathbf{x}_a$

Figure 3.3: A beam of light traveling across a medium.

3.2.1 Absorption

If we let $L(\mathbf{x}, \vec{\omega})$ be the radiance that reaches position $\mathbf{x}$ from direction $\vec{\omega}$, we can express absorption mathematically as

$$\mathrm{d}L(\mathbf{x}, \vec{\omega}) = -\sigma_a(\mathbf{x})L(\mathbf{x}, \vec{\omega})\mathrm{d}t. \tag{3.1}$$

Here, σ_a is the absorption function. It depends on the absorption coefficient of this volume, along with its density ρ at point $\mathbf{x}$:

$$\sigma_a(\mathbf{x}) = \sigma_a \rho(\mathbf{x}). \tag{3.2}$$

The same principle applies to the scattering and extinction functions σ_s and σ_t below.

3.2.2 Emission

Emission can increase the intensity of the light ray. It does not depend on the light from behind the volume L_a nor the incoming radiance L_i, only the emitted radiance L_e at that point. In practice, we let L_e be a constant color, instead controlling its intensity with the emission coefficient:

$$\mathrm{d}L(\mathbf{x}, \vec{\omega}) = \sigma_e(\mathbf{x})L_e(\mathbf{x}, \vec{\omega})\mathrm{d}t. \tag{3.3}$$

For our purposes, we assume this emission to be isotropic and have the same value everywhere, only attenuated by density and σ_e. In this case, the emission contribution will be the emitted radiance E at point $\mathbf{x}$ in that direction $\vec{\omega}$:

$$L_e(\mathbf{x}, \vec{\omega}) = \frac{1}{4\pi}E(\mathbf{x}) \tag{3.4}$$

3.2.3 Out-Scattering

Out-scattering is similar to absorption, and in fact indistinguishable from it from the perspective of the viewer: both effects result in decreased radiance. We still model both, as — unlike the absorption coefficient — the scattering coefficient influences not only the out-scattering component but the in-scattering component as well.

The difference in the equation here is that it depends on the scattering coefficient σ_s instead of the absorption coefficient:

$$\mathrm{d}L(\mathbf{x}, \vec{\omega}) = -\sigma_s(\mathbf{x}) L(\mathbf{x}, \vec{\omega}) \mathrm{d}t. \tag{3.5}$$

3.2.4 In-Scattering

Along the ray, photons from light sources scatter into the direction of the eye, in the same process that also scatters light away from it. This contribution is determined with the following equation:

$$\mathrm{d}L(\mathbf{x}, \vec{\omega}) = \sigma_s(\mathbf{x}) L_i(\mathbf{x}, \vec{\omega}) \mathrm{d}t. \tag{3.6}$$

It depends on the scattering coefficient σ_s, as well as the in-scattered radiance L_i. The latter is determined by taking the radiance of light coming from all directions of the unit sphere S^2:

$$L_i(\mathbf{x}, \vec{\omega}) = \int_{S^2} p(\mathbf{x}, \vec{\omega}, \vec{\omega_l})\, L(\mathbf{x}, \vec{\omega_l})\, \mathrm{d}\vec{\omega_l}. \tag{3.7}$$

To determine how much of this light is actually scattered in the direction $\vec{\omega}$, we employ a *phase function* p. A phase function is a probability density function that gives the probability that particles coming from a direction are scattered in another.

3.2.5 Phase functions

The particles a given volume consists of determine its scattering behavior. Some particles exhibit strong isotropic scattering, which scatters incoming light uniformly in all directions. Most particles however scatter light more in either the forwards or backwards direction, which is called anisotropic scattering.

The isotropic phase function is as follows:

$$p(\mathbf{x}, \vec{\omega}, \vec{\omega_l}) = \frac{1}{4\pi}. \tag{3.8}$$

Isotropic scattering is characteristic of Rayleigh scattering in, for example, the atmosphere.

In computer graphics, a commonly used phase function to approximate anisotropic scattering is the Henyey-Greenstein phase function, introduced in

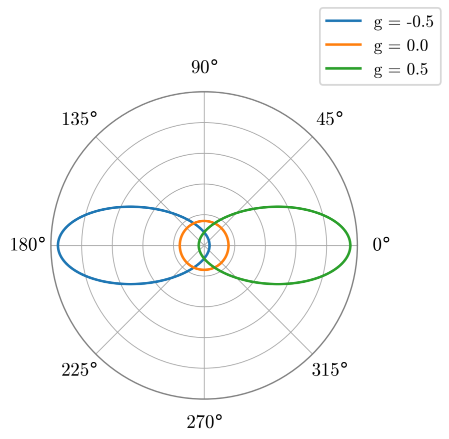


Figure 3.4: The Henyey-Greenstein function evaluated for different values of g.

1941 [14]. It was originally designed to apply on intergalactic dust, but due to its simplicity has since been used to simulate scattering of many other volumes, like clouds. It is defined as:

$$p(\mathbf{x}, \vec{\omega}, \vec{\omega_l}) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos(\angle(\vec{\omega}, \vec{\omega_l})))^{3/2}}, \tag{3.9}$$

where $\angle(\vec{\omega}, \vec{\omega_l})$ is the angle between the directions in radians, and g is the *asymmetry parameter*, which varies in $(-1, 1)$. See Figure 3.4 for plots of this function for different values of g.

3.2.6 Transmittance

The absorption and out-scattering components are often combined into extinction, which represents the total amount of radiance lost in the volume. Defining the extinction function as $\sigma_t(\mathbf{x}) = \sigma_a(\mathbf{x}) + \sigma_s(\mathbf{x})$, we get

$$\mathrm{d}L(\mathbf{x}, \vec{\omega}) = -\sigma_t(\mathbf{x})L(\mathbf{x}, \vec{\omega})\mathrm{d}t. \tag{3.10}$$

By integrating this differential equation over the length of the ray segment in the volume, spanning from the entry point $\mathbf{x}_{\text{entry}}$ to the exit point $\mathbf{x}_{\text{exit}}$, we get the *transmittance* $\mathcal{T}$. This is the fraction of light that gets transmitted through the subsection of the volume between two points, i.e. that does not get absorbed or out-scattered. It follows the Beer-Lambert law, expressing transmittance in terms of the extinction coefficient:

$$\mathcal{T}(\mathbf{x}_0, \mathbf{x}_1) = \exp(-\int_{\mathbf{x}_0}^{\mathbf{x}_1} \sigma_t(\mathbf{x}_s)\,\mathrm{d}\mathbf{x}_s). \tag{3.11}$$

A property of the transmittance that will be useful for approximation (Section 3.3.1) is that it is multiplicative. If $\mathbf{x}_1$ is a point on the line between $\mathbf{x}_0$ and $\mathbf{x}_2$, then

$$\mathcal{T}(\mathbf{x}_0, \mathbf{x}_2) = \mathcal{T}(\mathbf{x}_0, \mathbf{x}_1)\mathcal{T}(\mathbf{x}_1, \mathbf{x}_2). \tag{3.12}$$

3.2.7 The Volume Rendering Equation

When the contributions of each of the four scattering events are summed, we arrive at the *radiative transfer equation.* It describes the change in radiance of a light ray in the direction $\vec{\omega}$ at position $\mathbf{x}$:

$$\begin{aligned}\frac{\mathrm{d}L(\mathbf{x},\vec{\omega})}{\mathrm{d}t} &= \text{absorption} + \text{out-scattering} + \text{emission} + \text{in-scattering} \\ &= -\sigma_a(\mathbf{x})L(\mathbf{x},\vec{\omega}) - \sigma_s(\mathbf{x})L(\mathbf{x},\vec{\omega}) + \sigma_e(\mathbf{x})L_e(\mathbf{x},\vec{\omega}) + \sigma_s(\mathbf{x})L_i(\mathbf{x},\vec{\omega}) \\ &= -\sigma_t(\mathbf{x})L(\mathbf{x},\vec{\omega}) + \sigma_e(\mathbf{x})L_e(\mathbf{x},\vec{\omega}) + \sigma_s(\mathbf{x})L_i(\mathbf{x},\vec{\omega}).\end{aligned}$$

Like transmittance, this equation can be integrated from $\mathbf{x}_{\text{entry}}$ to $\mathbf{x}_{\text{exit}}$, which gives us the *volume rendering equation*:

$$\begin{aligned}L(\mathbf{x}_{\text{eye}}, -\vec{\omega}) = {} & \underbrace{\mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_{\text{exit}})\, L_a(\mathbf{x}_a, -\vec{\omega})}_{\text{reduced after-volume radiance}} \\ & + \underbrace{\int_{\mathbf{x}_{\text{entry}}}^{\mathbf{x}_s} \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s)\, \sigma_e(\mathbf{x}_s)\, L_e(\mathbf{x}_s, -\vec{\omega})\, \mathrm{d}\mathbf{x}_s}_{\text{accumulated emitted radiance}} \\ & + \underbrace{\int_{\mathbf{x}_{\text{entry}}}^{\mathbf{x}_s} \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s)\, \sigma_s(\mathbf{x}_s)\, L_i(\mathbf{x}_s, -\vec{\omega})\, \mathrm{d}\mathbf{x}_s}_{\text{accumulated in-scattering radiance}} .\end{aligned} \tag{3.13}$$

Here, the eye is positioned at $\mathbf{x}_{\text{eye}}$ and looks in direction $\vec{\omega}$ (Figure 3.3). The radiance L of light arriving at the eye, traveling in the opposite direction $-\vec{\omega}$, is determined by three components. The first describes the radiance L_a of the light from point $\mathbf{x}_a$ behind the volume, reduced by volume extinction through its transmittance $\mathcal{T}$. The second and third are the accumulated emitted and in-scattered radiance respectively, integrated over the length of the segment between the entry point and the variable of integration $\mathbf{x}_s$.

3.2.8 Uniformity

Until now, we have allowed the volume density to vary spatially. However, if we constrain the density to one value across the volume, we get a homogeneous volume. In real life, this still gives a good approximation for volumes with low variation in density, such as fog.

This simplifies certain equations. All functions such as the emission or the extinction function simplify to their coefficients multiplied by the uniform density ρ. In particular, because $\sigma_t(\mathbf{x}) = \sigma_t \rho$, the transmittance function $\mathcal{T}$ simplifies to

$$\mathcal{T}(\mathbf{x}_0, \mathbf{x}_1) = \exp(-\sigma_t\, \rho\, t), \tag{3.14}$$

where t is the distance between $\mathbf{x}_0$ and $\mathbf{x}_1$.

3.3 Solving the Volume Rendering Equation

Exact solutions of the volume rendering equation (Equation 3.2.7) are only possible in simple cases, such as homogeneous volumes. However, we require a solution that works for both homogeneous and heterogeneous volumes, so instead we look at methods to approximate the result with sufficient accuracy.

3.3.1 Single Scattering

To arrive at the first of the two approximation methods described in this thesis, we apply the Riemann sum approximation on the integrals in the volume rendering equation. The subsection of the ray between volume entry and exit is divided into a number of steps of size $\Delta\mathbf{x}$, and the ray's radiance is successively updated at each step along the ray. Each of these updates only accounts for one change in direction from the light source to the camera. This leads to the name *single scattering*: the light is scattered only once.

With a number of steps S, we define

$$\Delta\mathbf{x} = \frac{\mathbf{x}_{\text{exit}} - \mathbf{x}_{\text{entry}}}{S}, \tag{3.15}$$

$$\mathbf{x}_s = \mathbf{x}_{\text{entry}} + s\, \Delta\mathbf{x}, \tag{3.16}$$

and the full equation is

$$\begin{aligned} L(\mathbf{x}_{\text{eye}}, -\vec{\omega}) \approx\ & \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_{\text{exit}})\, L_a(\mathbf{x}_a, -\vec{\omega}) \\ & + \sum_{s=1}^{S} \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s)\, \sigma_e(\mathbf{x}_s)\, L_e(\mathbf{x}_s, -\vec{\omega}) \Delta\mathbf{x} \\ & + \sum_{s=1}^{S} \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s)\, \sigma_s(\mathbf{x}_s)\, L_i(\mathbf{x}_s, -\vec{\omega}) \Delta\mathbf{x}. \end{aligned} \tag{3.17}$$

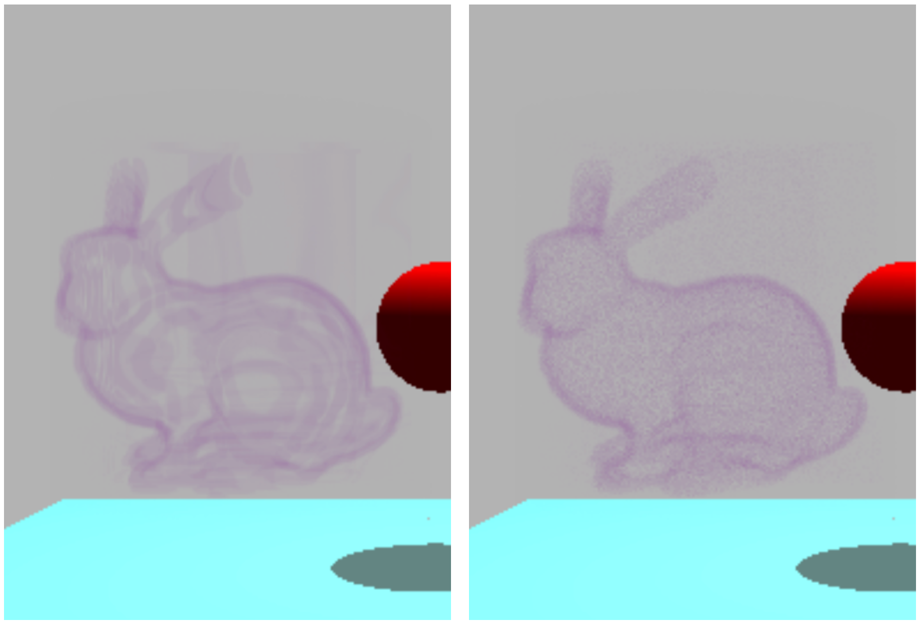

(a) Without jittering sample positions. (b) With jittering.

Figure 3.5: Banding when rendering a volume using single scattering. This is most noticeable in heterogeneous volumes.

This equation can be grouped into one sum as follows:

$$L(\mathbf{x}_{\text{eye}}, -\vec{\omega}) \approx \sum_{s=1}^{S} \Big(\mathcal{T}(\mathbf{x}_{s-1}, \mathbf{x}_s)\, L_a(\mathbf{x}_a, -\vec{\omega}) + \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s)\, [\sigma_e(\mathbf{x}_s)\, L_e(\mathbf{x}_s, -\vec{\omega}) + \sigma_s(\mathbf{x}_s)\, L_i(\mathbf{x}_s, -\vec{\omega})] \Big) \Delta\mathbf{x}. \tag{3.18}$$

Thanks to the multiplicative property of $\mathcal{T}$, we can write $\mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s) = \mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_1)\mathcal{T}(\mathbf{x}_1, \mathbf{x}_2)\cdots\mathcal{T}(\mathbf{x}_{s-1}, \mathbf{x}_s)$. This allows us to compute the transmittance only once per iteration and reuse the previous iteration's results to determine $\mathcal{T}(\mathbf{x}_{\text{entry}}, \mathbf{x}_s)$.

Avoiding Banding

To avoid banding as in Figure 3.5a, we randomly offset the sample points along the ray within their step. Even though this introduces noise to the image, as can be seen in Figure 3.5b, generally this is preferred over strong aliasing

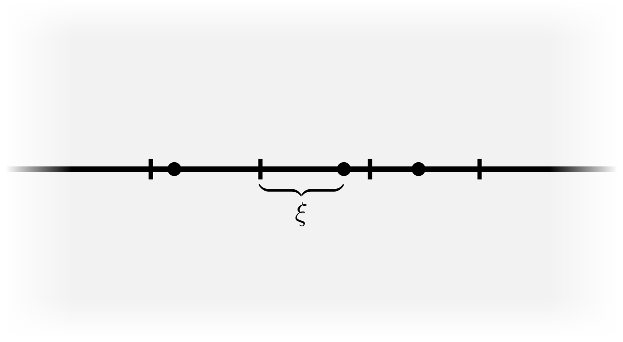


Figure 3.6: Jittering sample points within their step.

artifacts. To do this, we introduce a random variable ξ between 0 and 1, and set $\mathbf{x}_s = \mathbf{x}_{\text{eye}} + (s - \xi)\Delta\mathbf{x}$ for $0 < s \leq S$; see Figure 3.6.

3.3.2 Multiple Scattering

The second of the two methods uses Monte Carlo integration. In this method, a ray entering the volume does not continue along a straight line. Instead, we trace a reverse path through the volume, bouncing as many times as needed before exiting it. This is called a *random walk*: the new position and direction are determined stochastically. Due to the nature of the Monte Carlo method, as we simulate more and more of these walks and average their contributions, this average converges to the true solution.

The algorithm can be described as follows. The ray from the viewpoint at $\mathbf{x}_{\text{eye}}$ enters the volume. At this point, we simulate N random walks:

$$L(\mathbf{x}_{\text{eye}}, -\vec{\omega}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{W}(\mathbf{x}_{\text{entry}}, -\vec{\omega}). \tag{3.19}$$

$\mathcal{W}$ is the recursive random walk function, returning the radiance accumulated during its walk. It works similarly to the volume rendering function, but instead of attenuating the radiance from the subsection of the ray after the volume (L_a), we use the radiance from the next step of the walk:

$$\begin{aligned}\mathcal{W}(\mathbf{x}_i, -\vec{\omega_i}) = {} & \mathcal{T}(\mathbf{x}_i, \mathbf{x}_{i+1})\mathcal{W}(\mathbf{x}_{i+1}, \vec{\omega_{i+1}}) \\ & + \int_{\mathbf{x}_{\text{entry}}}^{\mathbf{x}_{i+1}} \sigma_e(\mathbf{x}_s) L_e(\mathbf{x}_s, -\vec{\omega_{i+1}})\mathrm{d}\mathbf{x}_s \\ & + \int_{\mathbf{x}_{\text{entry}}}^{\mathbf{x}_{i+1}} \sigma_s(\mathbf{x}_s) L_i(\mathbf{x}_s, -\vec{\omega_{i+1}})\mathrm{d}\mathbf{x}_s.\end{aligned} \tag{3.20}$$

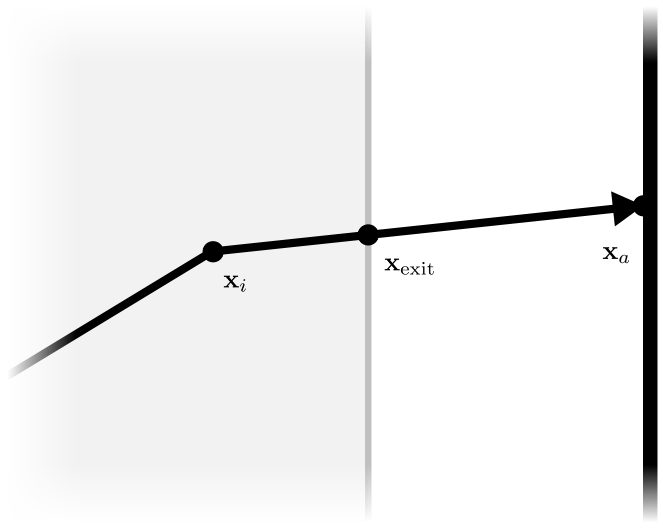


Figure 3.7: Determining the walk's exit point.

In this equation, the values of $\mathbf{x}$ and $\vec{\omega}$ are updated as follows:

$$\mathbf{x}_{i+1} = s(\mathbf{x}_i, \vec{\omega})\vec{\omega_i}, \tag{3.21}$$
$$\vec{\omega_{i+1}} = f_p(\vec{\omega_i}), \tag{3.22}$$

where s is the step function, returning the step size to the next scattering event, and f_p is the function that updates the direction based on the phase function p.

If $\mathbf{x}_{i+1}$ is positioned outside of the volume, then it is set to be the exit point, which is the point where the line between $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ intersects with the volume boundary. In this base case, illustrated in Figure 3.7, $\mathcal{W}$ simply returns the radiance from the ray that exited the volume:

$$\mathcal{W}(\mathbf{x}_{i+1}, -\vec{\omega_{i+1}}) = L_a(\mathbf{x}_a, -\vec{\omega_{i+1}}). \tag{3.23}$$

Delta Tracking

In homogeneous volumes, the step function s is calculated using the inverse probability distribution function of the transmittance:

$$s_o(\mathbf{x}, \vec{\omega}) = -\frac{\ln(1-\xi)}{\sigma_t}. \tag{3.24}$$

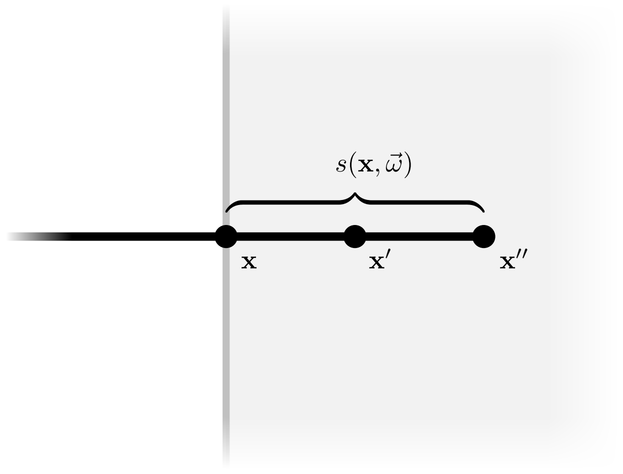


Figure 3.8: Example of delta tracking.

In heterogeneous volumes, we employ a method called *delta tracking* (Figure 3.8) to avoid oversampling in less dense areas. In this method, we also determine a step size using the homogeneous formula, but base it on the maximum extinction value $\max(\sigma_t)$ across the volume. We accept this step size to the new position $\mathbf{x}'$ with probability $\frac{\sigma_t(\mathbf{x}')}{\max(\sigma_t)}$; repeating the process if rejected. This way, we sample more often in areas with high extinction and relatively less often in sparse areas, which improves performance of the algorithm. The step size function now becomes

$$s_e(\mathbf{x}, \vec{\omega}) = \begin{cases} \text{distance to } \mathbf{x}' & \text{if } \xi < \frac{\sigma_t(\mathbf{x}')}{\max(\sigma_t)}, \\ s_o(\mathbf{x}', \vec{\omega}) & \text{otherwise,} \end{cases} \tag{3.25}$$

where $\mathbf{x}' = \mathbf{x} + s(\mathbf{x}, \vec{\omega})\,\vec{\omega}$.

3.4 Algorithms

The single and multiple scattering equations described above are written so that they can be solved algorithmically. These algorithms are described in this section. The first, Algorithm 1, corresponds to the equation of single scattering (Equation 3.18), and Algorithms 2 and 3 are derived from the the two equations of multiple scattering, Equations 3.19 and 3.20, respectively.

Algorithm 1: Single scattering

```
Input: Entry point $\mathbf{x}_{\text{entry}}$, exit point $\mathbf{x}_{\text{exit}}$, step size, direction $\vec{\omega}$
Output: The final color $c$
1  transmittance $T \leftarrow 1.0$
2  $c \leftarrow$ black
3  $t \leftarrow$ distance between $\mathbf{x}_{\text{entry}}$ and $\mathbf{x}_{\text{exit}}$
4  number of steps $S \leftarrow \left\lceil \frac{t}{\text{step size}} \right\rceil$
5  stride $\Delta s \leftarrow \frac{t}{S}$
6  for step $i \leftarrow 0$ to $S$ do
7      sample position $\mathbf{x}_s \leftarrow \mathbf{x}_{\text{eye}} + (i + \xi) \cdot \Delta s \cdot \vec{\omega}$
8      $c \leftarrow c + T \cdot \sigma_e \cdot \Delta s$
9      for light $l$ in lights in scene do
10         $c_l \leftarrow$ color of light $l$
11         $\vec{\omega_l} \leftarrow$ direction to light
12         $t_l \leftarrow$ distance to exit point in direction $\vec{\omega_l}$
13         number of steps towards light $S_l \leftarrow \left\lceil \frac{t_l}{\text{step size}} \right\rceil$
14         stride towards light $\Delta s_l \leftarrow \frac{t_l}{S_l}$
15         transmittance towards light $T_l \leftarrow 1.0$
16         for step towards light $i_l \leftarrow 0$ to $S_l$ do
17             light sample position $\mathbf{x}_{s_l} \leftarrow \mathbf{x}_s + (i_l + \xi) \cdot \Delta s_l \cdot \vec{\omega_l}$
18             $T_l \leftarrow T_l \cdot e^{-\Delta s_l \cdot \sigma_t(\mathbf{x}_{s_l})}$
19         end
20         phase at sample $p_s \leftarrow p(\mathbf{x}_s, -\vec{\omega}, \vec{\omega_l})$
21         $c \leftarrow c \cdot c_l \cdot T_l \cdot p_s \cdot \sigma_s(\mathbf{x}_s) \cdot T \cdot \Delta s$
22     end
23     $T \leftarrow T \cdot e^{-\Delta s \cdot \sigma_t(\mathbf{x}_s)}$
24 end
25 $c_a \leftarrow$ color of ray cast from $\mathbf{x}_{\text{entry}}$ in direction $\vec{\omega}$
26 return $c + T \cdot c_a$
```

Algorithm 2: Multiple scattering

```
Input: Number of walks $N$, maximum number of steps per walk $S$
Output: The final color $c$
1  $c \leftarrow$ black
2  for walk $i \leftarrow 0$ to $N$ do
3      $c \leftarrow c + \mathcal{W}(0, \mathbf{x}_{\text{entry}}, \vec{\omega}, S, 0, 0)$
4  end
5  $c \leftarrow \frac{c}{N}$
```

Algorithm 3: Random walk function $\mathcal{W}$

```
Input: Step $i$, segment origin $\mathbf{x}_i$, segment direction $\vec{\omega_i}$, maximum
       number of steps $S$, accumulated emission $\Sigma e$, accumulated
       scattering $\Sigma s$
Output: The color $c_i$ of this segment and the ones after it
1  if $i = S$ then
2      return $c_i \leftarrow$ black
3  end
4  step size $\Delta s \leftarrow s(\mathbf{x}_i, \vec{\omega_i})$
5  new position $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \Delta s\, \vec{\omega_i}$
6  new direction $\vec{\omega_{i+1}} \leftarrow f_p(\vec{\omega_i})$
7  $\Sigma e \leftarrow \Sigma e + \sigma_e(\mathbf{x}_i)\, L_e(\mathbf{x}_i, -\vec{\omega_i})$
8  $\Sigma s \leftarrow \Sigma s + \sigma_s(\mathbf{x}_i)\, L_i(\mathbf{x}_i, -\vec{\omega_i})$
9  transmittance $T \leftarrow e^{-\Delta s \cdot \sigma_t(\mathbf{x})}$
10 next step color $c_{i+1} \leftarrow \mathcal{W}(i+1, \mathbf{x}_{i+1}, \vec{\omega_{i+1}}, S, \Sigma e, \Sigma s)$
11 return $c_i \leftarrow T \cdot c_{i+1} + \Sigma e + \Sigma s$
```

3.4.1 Optimization Techniques

The described algorithms are not the most efficient currently available. However, the aim of the VVRT application is not to implement the most realistic nor the fastest rendering algorithm. Instead, what we are concerned with is educational value – explaining how reality is modeled, and the fundamental techniques and algorithms, visualized in an intuitive way. Therefore, we include only simple ray casting and Monte Carlo algorithms.

However, in the interest of improving performance to make the application more accessible to those with older hardware, some optimizations have still been implemented in those algorithms. They are not mentioned in the tutorial, as they are minor implementation details.

The first is that if the density at a point is zero, the extinction, emission and scattering values will also be zero. In this case, we can simply continue to the next iteration. This is shown in Algorithm 4.

Algorithm 4: Optimization: density is zero

```
... (calculation of sample position $\mathbf{x}_s$)
if $\rho(\mathbf{x}_s) = 0$ then
    continue to next iteration
end
...
```

Another optimization is termed early ray termination. We break out of the loop once the transmission is below a certain threshold ϵ, because iterating more does not provide enough change in radiance to visibly change the final color of the pixel. This is shown in Algorithm 5. In the code, ϵ is set to `0.001`.

Algorithm 5: Optimization: early ray termination

```
... (calculation of transmittance $T$)
if $T < \epsilon$ then
    break out of the for loop
end
```

Lastly, when a heterogeneous volume has density values of 0 across parts of its density field, we can skip that empty space. For this, efficient algorithms have been developed, such as octrees [15] and *SparseLeap* [16]. Implementing these would require quite some changes to the algorithm; specifically the iteration would be adjusted to jump to the next non-empty part of the volume. This would result in some speedup of the rendering time. This optimization has not been implemented, but could be covered in future work.

Chapter 4

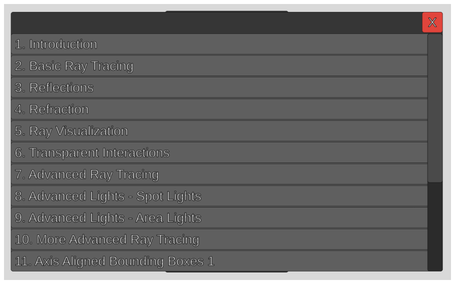
Interface

This chapter gives a brief overview of the user interface and ray tracing visualization in VVRT. Then, it describes the additions made in this extension, including visualization of scattering, additions to the user interface and the new levels and their tutorials.

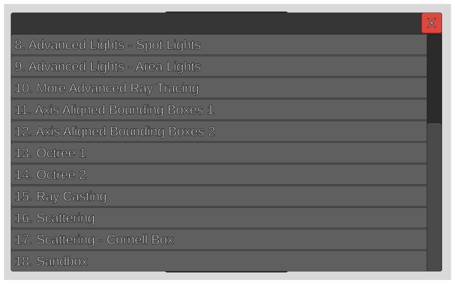
4.1 VVRT

VVRT consists of levels that the user plays through (Figure 4.1). Each level consists of a simple scene, that explains a particular aspect of or topic in ray tracing, like reflection and refraction [9], axis aligned bounding boxes [12], and of course, ray casting [17].

The interface, as seen in Figure 4.2, is akin to the Unity editor itself: a configuration panel on the right side of the screen allows the user to control certain aspects of the visualization or rendering. Also, each object can be translated, rotated and scaled with transform handles. Different from the editor


1. Introduction
2. Basic Ray Tracing
3. Reflections
4. Refraction
5. Ray Visualization
6. Transparent Interactions
7. Advanced Ray Tracing
8. Advanced Lights - Spot Lights
9. Advanced Lights - Area Lights
10. More Advanced Ray Tracing
11. Axis Aligned Bounding Boxes 1

(a) Levels 1 to 11.


8. Advanced Lights - Spot Lights
9. Advanced Lights - Area Lights
10. More Advanced Ray Tracing
11. Axis Aligned Bounding Boxes 1
12. Axis Aligned Bounding Boxes 2
13. Octree 1
14. Octree 2
15. Ray Casting
16. Scattering
17. Scattering - Cornell Box
18. Sandbox

(b) Levels 8 to 18.

Figure 4.1: The levels in the application, including those in the extension.

however, are the tutorial and render preview panels. The tutorial box displays each task and allows the user to navigate between tasks and even levels, and the render preview panel shows the color of each of the rays visualized in the scene.

These rays are generated by the ray tracer, and animated as they traverse the scene, starting at the camera. Upon hitting an object, the rays trace a light ray towards each light source, and reflect and refract if the material type allows it. This is all visualized accordingly in the scene, as shown in VVRT [13]. Rays also are of a certain type, depending on their function. For instance, a reflected ray has type `Reflect`. This is the same for refracted, light and shadow rays, among others. The different ray functions are colored differently to make them easier to distinguish [8].

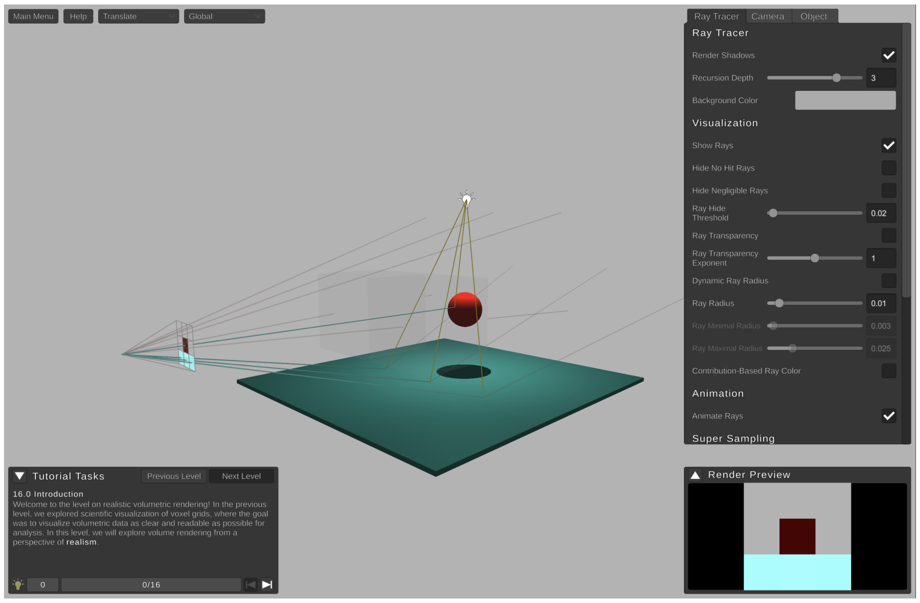
4.2 Contribution

We implemented a simple level with a scene consisting of only a few objects: a cyan floor, a red sphere, a simple point light and two new volume objects, as seen in Figure 4.2. The cyan floor sphere and light are familiar from earlier levels. They function both as visual anchor points and backgrounds for the volumes, allowing easy comparison between them. One of the volumes is heterogeneous, showing one of four available volumetric datasets, the other is homogeneous, with a uniform density throughout.

In this version of the ray tracer, the volumes are visualized as partly transparent gray boxes, like in the ray casting level. This indicates the boundaries of their density fields.

Like in previous levels, the render preview is set to be three pixels wide and three pixels high. This keeps the optimal balance of information density, being not too crowded while still giving enough different ray trajectories. The left and right rays in the middle row traverse the two volumes, and the middle one passes in between them and hits the sphere. This allows the user to easily select the volume-specific rays to see their sample points in more detail.

A second level was also added, displaying a Cornell box (Figure 4.3). This box is used to evaluate rendering algorithms by comparing them with a real-life photograph of the same box. The contents of the box and the color of its walls varies across texts, and we have chosen for a red left wall, green right wall, and two smaller boxes, standing on the floor. The larger of the two is a mirror, positioned so that it reflects light from the camera towards the smaller one, which is just opaque. Lastly, the space in the box is filled with a homogeneous volume, representing fog.



Ray Tracer

Render Shadows ✓
Recursion Depth 3
Background Color

Visualization

Show Rays ✓
Hide No Hit Rays
Hide Negligible Rays
Ray Hide Threshold 0.02
Ray Transparency
Ray Transparency Exponent 1
Dynamic Ray Radius
Ray Radius 0.01
Ray Minimal Radius 0.003
Ray Maximal Radius 0.025
Contribution-Based Ray Color

Animation

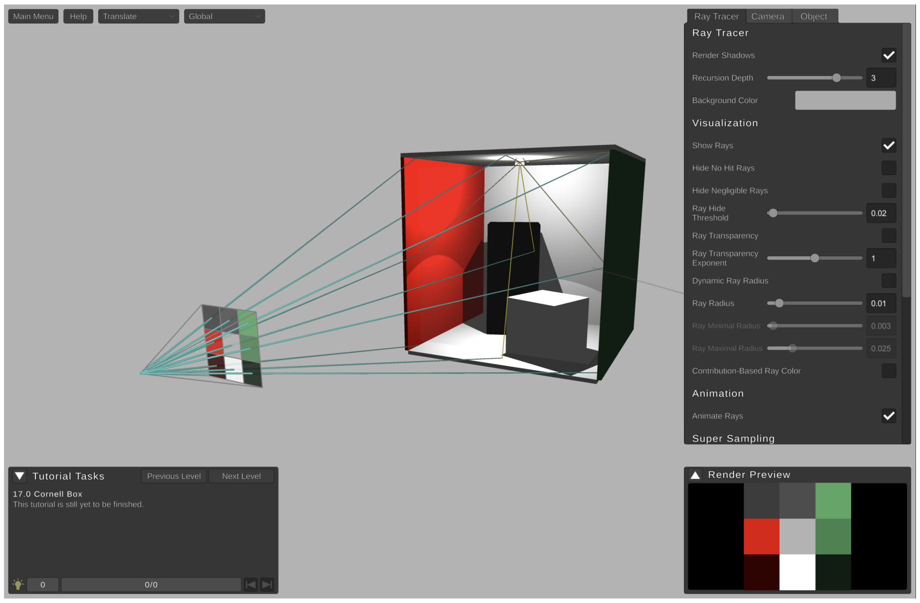
Animate Rays ✓

Super Sampling

16.0 Introduction
Welcome to the level on realistic volumetric rendering! In the previous level, we explored scientific visualization of voxel grids, where the goal was to visualize volumetric data as clear and readable as possible for analysis. In this level, we will explore volume rendering from a perspective of **realism**.

0 0/16

Figure 4.2: The scene in the scattering level.



Ray Tracer

Render Shadows ✓
Recursion Depth 3
Background Color

Visualization

Show Rays ✓
Hide No Hit Rays
Hide Negligible Rays
Ray Hide Threshold 0.02
Ray Transparency
Ray Transparency Exponent 1
Dynamic Ray Radius
Ray Radius 0.01
Ray Minimal Radius 0.003
Ray Maximal Radius 0.025
Contribution-Based Ray Color

Animation

Animate Rays ✓

Super Sampling

17.0 Cornell Box
This tutorial is still yet to be finished.

0 0/0

Figure 4.3: The scene in the Cornell Box level.

4.3 Visualization

Designing an appropriate visualization is a balance of information density. We aim to give the user an intuitive view of the algorithms, while taking care to not clutter the screen with superfluous information.

4.3.1 Single Scattering

The single scattering approximation as described in Section 3.3.1 is visualized as in Figure 4.4. The visualization follows the style of the previous levels, with rays represented by three-dimensional lines and identical ray colors as normal and light rays. A ray entering the volume is continued along a straight line to its exit, with light rays traced to all light sources from each sample. The steps are not distinguished; instead, they are hinted at by the light rays. While displaying each light ray does increase the amount of rays visible on the screen significantly, hiding them would remove valuable insight into whether this sample is illuminated by the light or blocked by an object.

Once the user switches to the single-pixel view, the sample points along the ray are shown, as in Figure 4.5. These points are rendered as black spheres, similar to those from the previous level on ray casting. They provide valuable information to the user, showing how each light ray is also attenuated as it traverses the volume towards the light source.

Additionally, the user is able to adjust the step size of the algorithm. When this is changed, the visualization updates accordingly. The minimum value is set at 0.1, which gives a result as in Figure 4.6a. As seen, this is an appropriate minimum bound; decreasing it further would severely hinder visibility as well as performance on older hardware. The corresponding upper bound does not have a similar issue, and thus it is set at 1 to have the default value of 0.5 be near the center of the slider (Figure 4.6b).

4.3.2 Multiple Scattering

When the rendering technique is set to multiple scattering, all random walks are visualized (Figure 4.7). The number of rays drawn in the scene equals the number of visualized walks the user has selected in the ray tracer control panel.

Each random walk is performed as described in Section 3.3.2: when a ray enters the volume, it takes a step of a probabilistic length before scattering in a different direction. At each of these steps, the volume is sampled, and a light ray are cast to every light source to determine its in-scattering contribution. As shown in Figure 4.7, the random walks oftentimes take only a small number of steps before exiting the volume or getting absorbed. While this might not correspond a student's traditional idea of what a random walk might look like, it is consistent with the random walk algorithm (Algorithm 3). This also means the render preview's pixels will be consistent with the rays in the scene, allowing

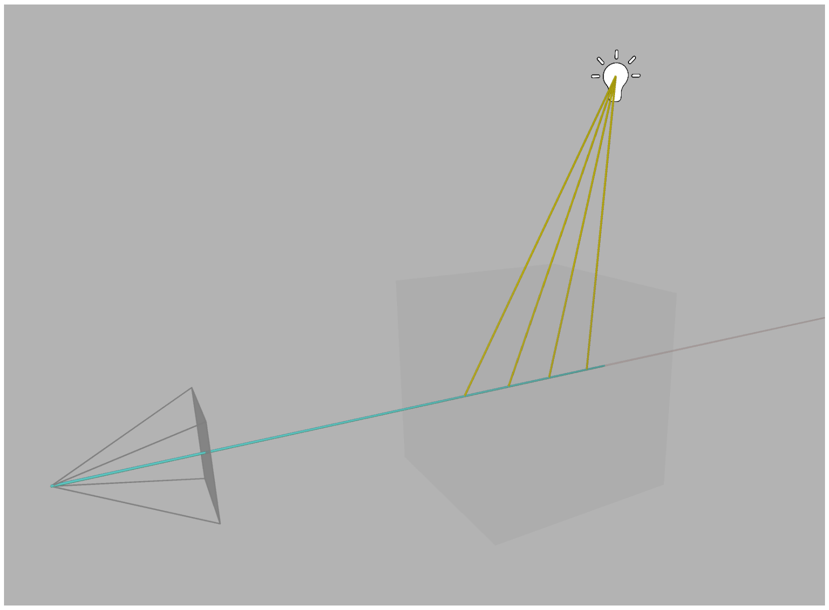


Figure 4.4: Visualization of single scattering, without single-pixel view enabled.

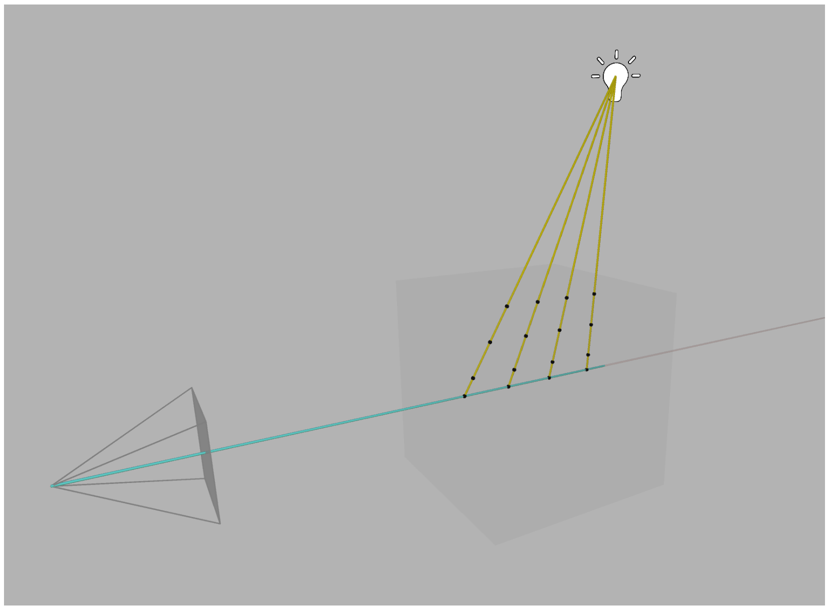


Figure 4.5: Visualization of single scattering, with single-pixel view enabled.

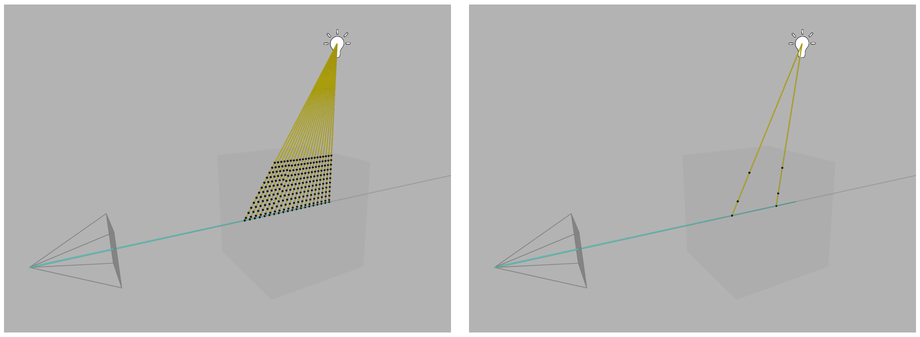

(a) A step size of 0.1. (b) A step size of 1.

Figure 4.6: Different step sizes in single scattering.

users to derive those pixels directly from the rays. Additionally, the tutorial explains the nature of this stochastic simulation, ensuring the user understands the concept.

This technique also has a corresponding single-pixel view, seen in Figure 4.8. Here, sample points are once again visualized as black spheres, this time conveniently also representing scattering events. When a walk terminates with an absorption event, this is shown as a larger dark red sphere. This makes them stand out from regular sample events, while still being small enough to not be distracting.

The number of walks visualized in the scene can be adjusted separately from the number of walks taken by the renderer. This is done to be able to freely increase the rendering fidelity without drawing too many walks in the scene. Both sliders have a minimum of one and a default of three, the latter of which was chosen to adequately convey the nature of Monte Carlo methods while preserving visual clarity. Additionally, setting both sliders to the same value means the render preview is consistent with a rendered image. Of course, the user may increase the number of walks drawn to ten, which is a sensible maximum for the amount of rays. On the other hand, the rendered amount can be set to at most 20. While usual Monte Carlo rendering simulates orders of magnitude more walks, our rendering is done on the CPU. Unfortunately, this means this number cannot be increased much before rendering takes too long.

When these settings are changed, or the rays are redrawn in general, all rays undergo a new random walk, meaning the ray paths will differ drastically. This was chosen to illustrate and keep true to the random nature of the algorithm.

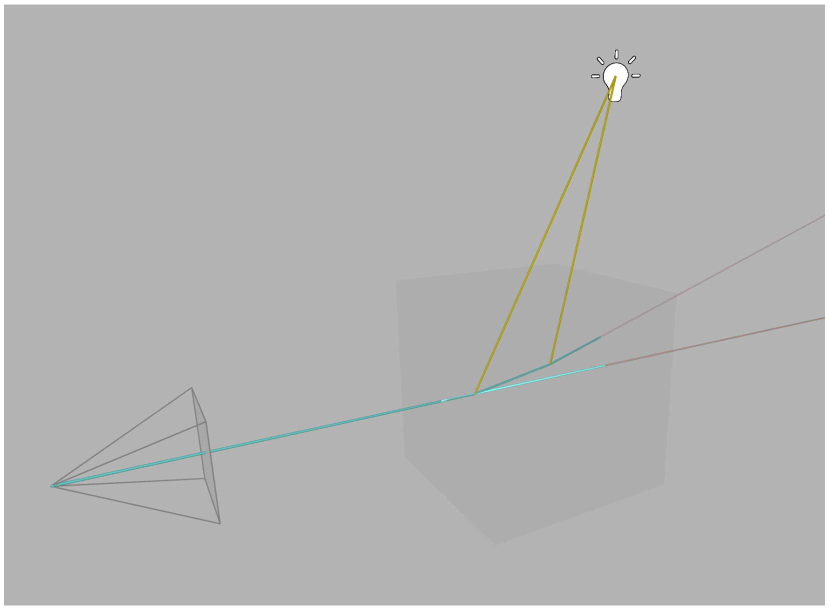


Figure 4.7: Visualization of multiple scattering, without single-pixel view enabled.

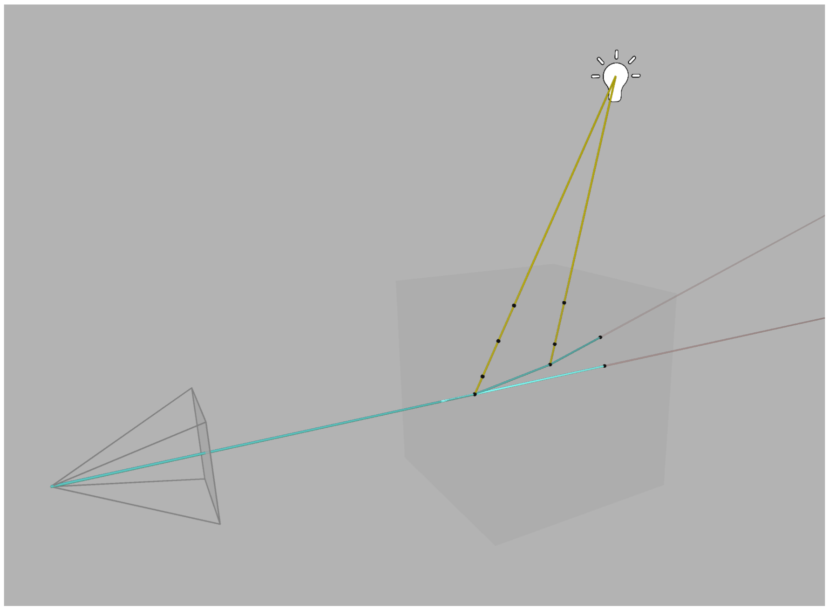
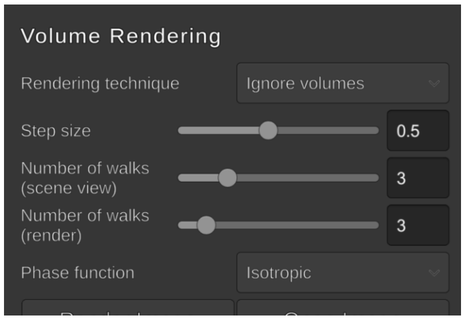


Figure 4.8: Visualization of multiple scattering, with single-pixel view enabled.


Volume Rendering

Rendering technique: Ignore volumes

Step size: 0.5

Number of walks (scene view): 3

Number of walks (render): 3

Phase function: Isotropic

Figure 4.9: Ray tracer configuration options.

4.4 User Interface

All options from earlier levels are kept, and a number of settings have been added. In particular, a new section has been added to the ray tracer configuration panel, and two configuration panels have been created for both types of volume.

4.4.1 Ray Tracer Configuration

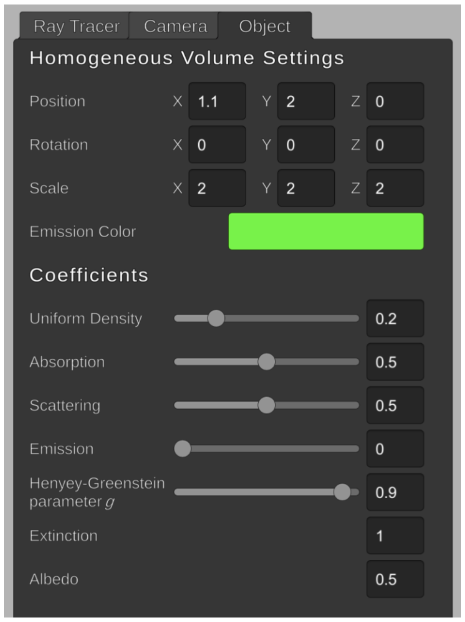
Underneath the familiar configuration options for the ray tracer, there is a new section on volume rendering, seen in Figure 4.9. It contains a dropdown menu for the volume rendering technique. Possible options are to ignore volumes entirely, or to simulate either single scattering or multiple scattering. By default, it is set to ignore volumes, disabling all per-technique configuration options. This is done to ease the user into the new visualization options, as they will be instructed by the tutorial what to expect and to explore the options at their own pace.

When single scattering is used, the step size slider becomes available, and the number of walks sliders are disabled. On the contrary, when multiple scattering is used, those become interactive and the step size slider is grayed out.

Lastly, both techniques make use of a phase function. The two options available are the isotropic phase function, and the Henyey-Greenstein phase function, as described in Section 3.2.5.

4.4.2 Volume Configuration

Heterogeneous and homogeneous volumes share most of their configurable parameters. Just like regular mesh objects like the sphere, they have editors for the position, rotation and scale. Both have sliders for the absorption coefficient σ_a, scattering coefficient σ_s and emission coefficient σ_t, along with a color picker for the emission color, L_e. In addition, there is a slider to adjust the



Homogeneous Volume Settings

Position	X 1.1	Y 2	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 2	Y 2	Z 2
Emission Color	🟩		

Coefficients

Uniform Density	0.2
Absorption	0.5
Scattering	0.5
Emission	0
Henyey-Greenstein parameter *g*	0.9
Extinction	1
Albedo	0.5

Figure 4.10: The homogeneous volume control panel.

Henyey-Greenstein asymmetry parameter g, and two fields simply displaying the extinction and albedo values. All parameters display a tooltip when hovered over that gives a short description.

Homogeneous Volumes

Specific to homogeneous volumes is their uniform density, and so they contain a Uniform Density slider to adjust it. The configuration panel for this type of volume is shown in Figure 4.10.

Heterogeneous Volumes

On the other hand, the density in heterogeneous volumes varies spatially. The data is arranged in a grid of voxels, each having the associated density value in the range $[0, 1]$. This data organization is the same as in the previous level by Van der Wal, as well as the available data to visualize. There are four available volume types: the Bucky ball, the Stamford Bunny, an engine part and hazelnuts on a branch. These can be selected via a dropdown menu in the volume properties panel, shown in Figure 4.11. The full configuration panel for heterogeneous volumes is shown in Figure 4.12.

For the emission color, we decided to not implement the transfer function as

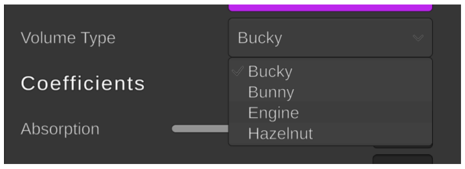
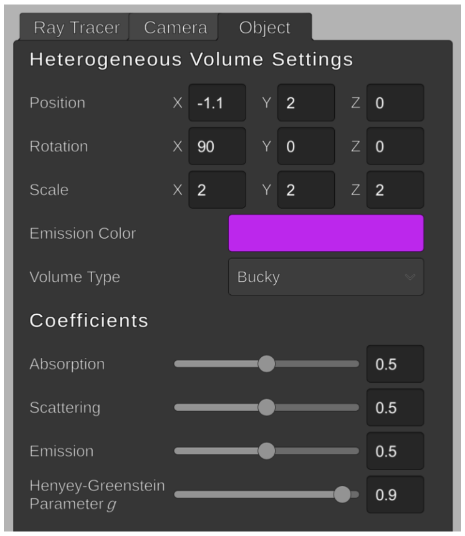


Figure 4.11: A dropdown allowing the user to select a different dataset.



Heterogeneous Volume Settings

Position X -1.1 Y 2 Z 0

Rotation X 90 Y 0 Z 0

Scale X 2 Y 2 Z 2

Emission Color

Volume Type Bucky

Coefficients

Absorption 0.5

Scattering 0.5

Emission 0.5

Henyey-Greenstein Parameter g 0.9

Figure 4.12: The heterogeneous volume control panel.

seen in the previous level on ray casting. This would add unnecessary complexity to the control panel, distracting the user from the parameters of importance to this level.

4.5 Tutorial

The two levels are accompanied by a tutorial that aims to guide the user through the added functionality of this VVRT extension, and the new visualization options. It is written for computing science students familiar with the concept of light transport in volumes, but not with the specifics, nor necessarily how the effects are usually approximated in rendering. The tutorial is divided into 17 tasks, in each of which a small concept is explained in about three to four sentences. Some of the tasks require the user to adjust a setting in the control panel, or render an image, before it proceeds to the next task.

4.5.1 Scattering Level

The tutorial starts by relating this level to the previous one, and how the techniques differ in goal and approach (Figures 4.13a and 4.13b). After this, the background knowledge required to understand the algorithms is explained conceptually, before we introduce those algorithms (Figures 4.13c, 4.13d and 4.13e). First single scattering is explained (Figure 4.13f), and the user is instructed to select this technique in the control panel (Figure 4.13g). Here, they adjust the step size and explore the single pixel view, guided by the tutorial (Figures 4.13g and 4.13h).

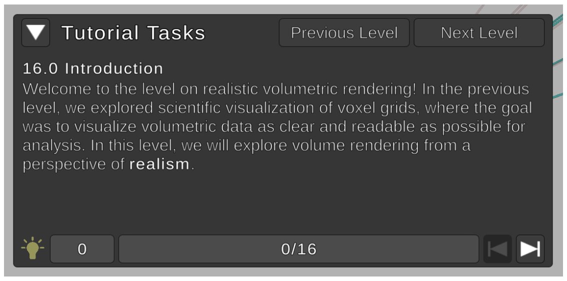
Before continuing to the multiple scattering, the user is told to render an image (Figure 4.13i). They will see the volume representations show up in the final render, with the bunny model in the left volume and the right volume representing a gray fog. The tutorial tells them to adjust the volume parameters and see how the preview and final render react to their changes (Figure 4.13j).

In the multiple scattering part of the tutorial (Figure 4.13k), we go through a similar path as in the single scattering part. This time, the user is walked through the multiple scattering controls, consisting of the two sliders adjusting the number of walks (Figure 4.13l). Once again, the user selects a single pixel to see the single-pixel view of multiple scattering, and renders the image (Figures 4.13m and 4.13n). They are told that the image will be noisier due to the nature of the stochastic simulation.

Lastly, the level covers the different phase function options. After selecting the Henyey-Greenstein phase function, the user is instructed to configure the asymmetry parameter g in the control panels of the volumes (Figures 4.13o and 4.13p) and to continue to the second level, where they will apply their newly acquired knowledge (Figure 4.13q).

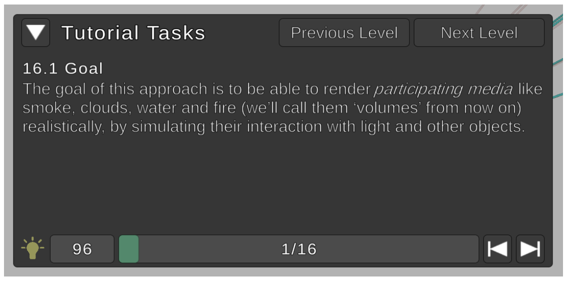
4.5.2 Cornell Box Level

The second level, containing the Cornell box, only has one tutorial task. It tells the user about the contents of the box, and encourages them to try out different configurations and see their effect on the rendered image (Figure 4.13r).



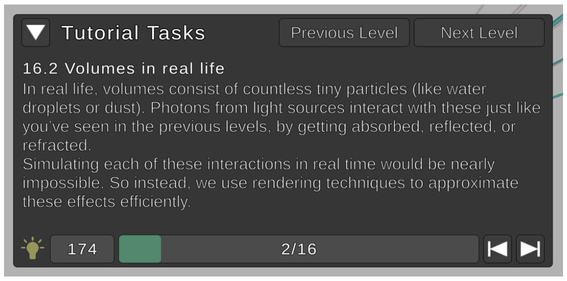
16.0 Introduction
Welcome to the level on realistic volumetric rendering! In the previous level, we explored scientific visualization of voxel grids, where the goal was to visualize volumetric data as clear and readable as possible for analysis. In this level, we will explore volume rendering from a perspective of **realism**.

(a) Task 0.



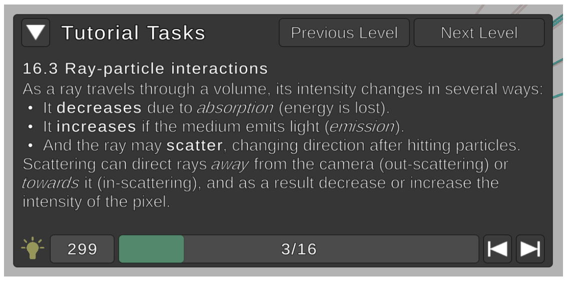
16.1 Goal
The goal of this approach is to be able to render *participating media* like smoke, clouds, water and fire (we'll call them 'volumes' from now on) realistically, by simulating their interaction with light and other objects.

(b) Task 1.



16.2 Volumes in real life
In real life, volumes consist of countless tiny particles (like water droplets or dust). Photons from light sources interact with these just like you've seen in the previous levels, by getting absorbed, reflected, or refracted.
Simulating each of these interactions in real time would be nearly impossible. So instead, we use rendering techniques to approximate these effects efficiently.

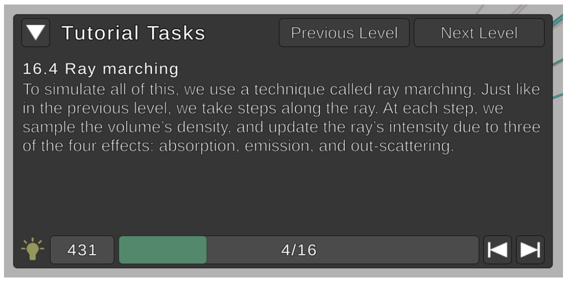
(c) Task 2.



16.3 Ray-particle interactions
As a ray travels through a volume, its intensity changes in several ways:
- It **decreases** due to *absorption* (energy is lost).
- It **increases** if the medium emits light (*emission*).
- And the ray may **scatter**, changing direction after hitting particles.

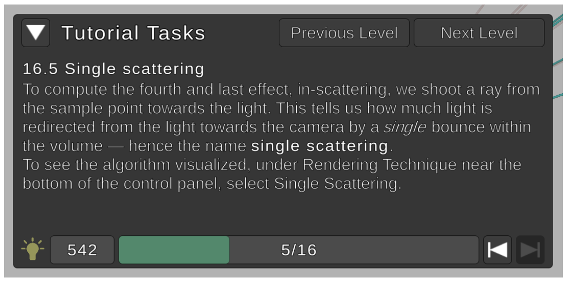
Scattering can direct rays *away* from the camera (out-scattering) or *towards* it (in-scattering), and as a result decrease or increase the intensity of the pixel.

(d) Task 3.



16.4 Ray marching
To simulate all of this, we use a technique called ray marching. Just like in the previous level, we take steps along the ray. At each step, we sample the volume's density, and update the ray's intensity due to three of the four effects: absorption, emission, and out-scattering.

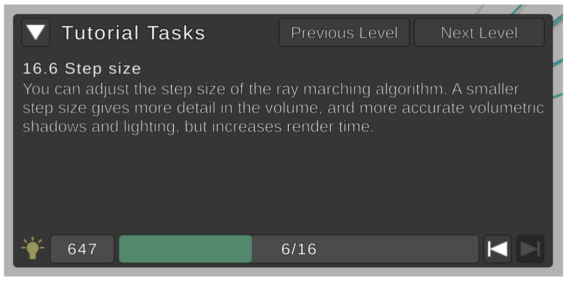
(e) Task 4.



16.5 Single scattering
To compute the fourth and last effect, in-scattering, we shoot a ray from the sample point towards the light. This tells us how much light is redirected from the light towards the camera by a *single* bounce within the volume — hence the name **single scattering**.
To see the algorithm visualized, under Rendering Technique near the bottom of the control panel, select Single Scattering.

(f) Task 5.

Figure 4.13: Tutorial tasks (1 of 3).

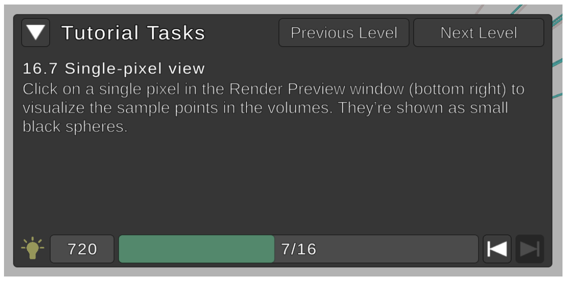

Tutorial Tasks
Previous Level
Next Level

16.6 Step size

You can adjust the step size of the ray marching algorithm. A smaller step size gives more detail in the volume, and more accurate volumetric shadows and lighting, but increases render time.

647 6/16

(g) Task 6.

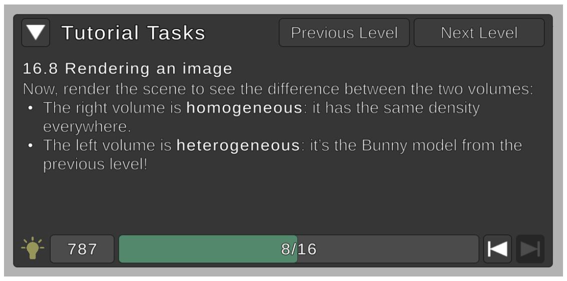

Tutorial Tasks
Previous Level
Next Level

16.7 Single-pixel view

Click on a single pixel in the Render Preview window (bottom right) to visualize the sample points in the volumes. They're shown as small black spheres.

720 7/16

(h) Task 7.


Tutorial Tasks
Previous Level
Next Level

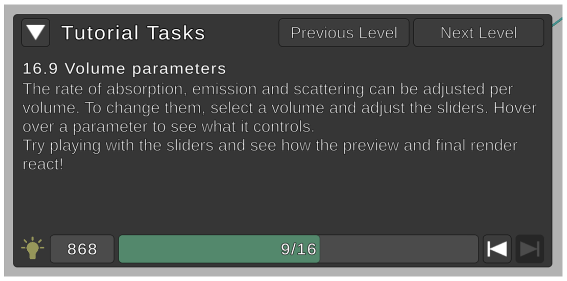
16.8 Rendering an image

Now, render the scene to see the difference between the two volumes:

- The right volume is **homogeneous**: it has the same density everywhere.
- The left volume is **heterogeneous**: it's the Bunny model from the previous level!

787 8/16

(i) Task 8.

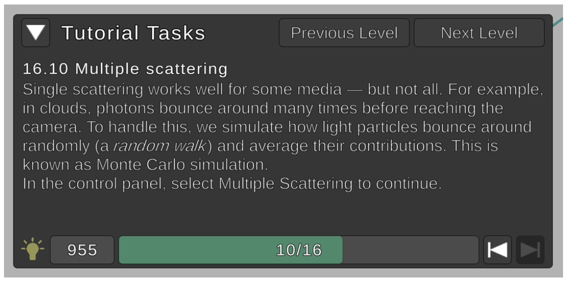

Tutorial Tasks
Previous Level
Next Level

16.9 Volume parameters

The rate of absorption, emission and scattering can be adjusted per volume. To change them, select a volume and adjust the sliders. Hover over a parameter to see what it controls.
Try playing with the sliders and see how the preview and final render react!

868 9/16

(j) Task 9.

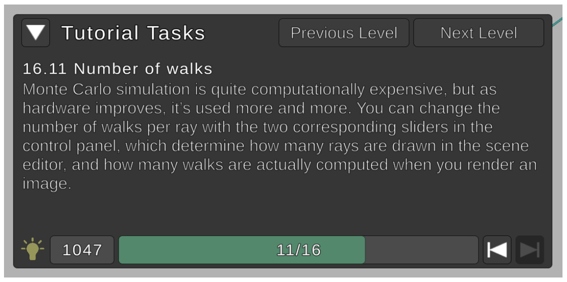

Tutorial Tasks
Previous Level
Next Level

16.10 Multiple scattering

Single scattering works well for some media — but not all. For example, in clouds, photons bounce around many times before reaching the camera. To handle this, we simulate how light particles bounce around randomly (a *random walk*) and average their contributions. This is known as Monte Carlo simulation.
In the control panel, select Multiple Scattering to continue.

955 10/16

(k) Task 10.


Tutorial Tasks
Previous Level
Next Level

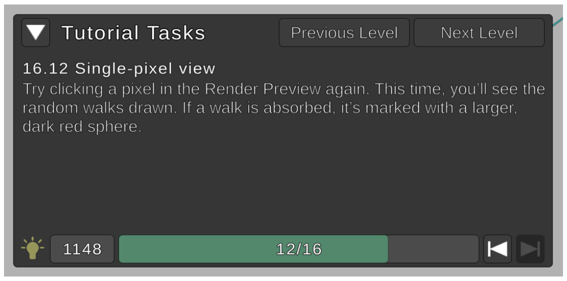
16.11 Number of walks

Monte Carlo simulation is quite computationally expensive, but as hardware improves, it's used more and more. You can change the number of walks per ray with the two corresponding sliders in the control panel, which determine how many rays are drawn in the scene editor, and how many walks are actually computed when you render an image.

1047 11/16

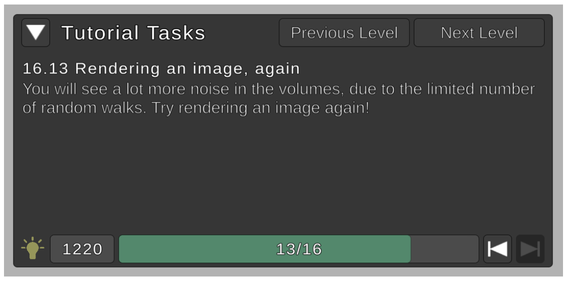
(l) Task 11.

Figure 4.13: Tutorial tasks (2 of 3).



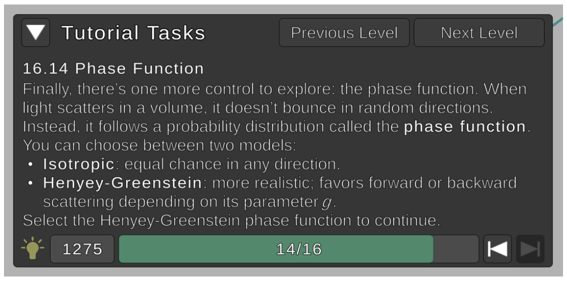
16.12 Single-pixel view
Try clicking a pixel in the Render Preview again. This time, you'll see the random walks drawn. If a walk is absorbed, it's marked with a larger, dark red sphere.

(m) Task 12.



16.13 Rendering an image, again
You will see a lot more noise in the volumes, due to the limited number of random walks. Try rendering an image again!

(n) Task 13.

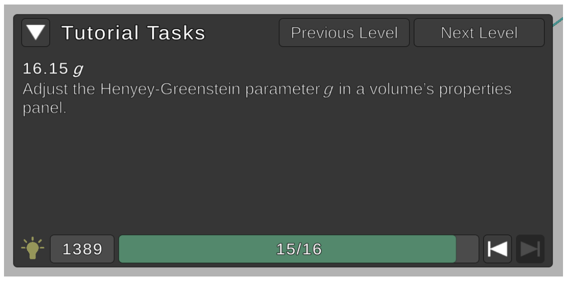


16.14 Phase Function
Finally, there's one more control to explore: the phase function. When light scatters in a volume, it doesn't bounce in random directions. Instead, it follows a probability distribution called the **phase function**. You can choose between two models:

- **Isotropic**: equal chance in any direction.
- **Henyey-Greenstein**: more realistic; favors forward or backward scattering depending on its parameter g.

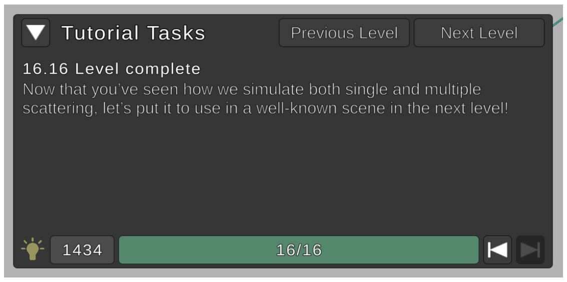
Select the Henyey-Greenstein phase function to continue.

(o) Task 14.



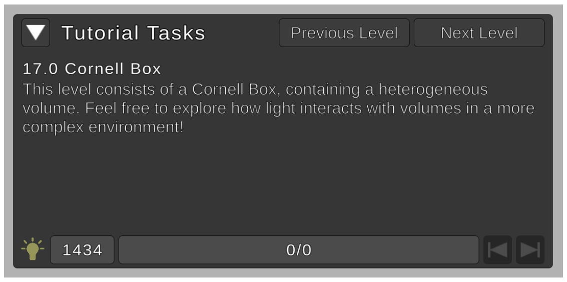
16.15 g
Adjust the Henyey-Greenstein parameter g in a volume's properties panel.

(p) Task 15.



16.16 Level complete
Now that you've seen how we simulate both single and multiple scattering, let's put it to use in a well-known scene in the next level!

(q) Task 16.



17.0 Cornell Box
This level consists of a Cornell Box, containing a heterogeneous volume. Feel free to explore how light interacts with volumes in a more complex environment!

(r) Task 0 of the second level.

Figure 4.13: Tutorial tasks (3 of 3).

Chapter 5

Implementation

This chapter describes the low-level implementation details, including an overview of the structure of the codebase and our contributions to it. The codebase can be found at the following link: `https://github.com/Japsert/VVRT-Scattering`.

5.1 Architecture

VRT and its successor VVRT evolved over a number of contributions, all focused on improving educational ability and accessibility. The underlying codebase[1] has grown and matured too. In this section, we provide an overview of its general structure and how different parts work, individually and together. Then, we describe our contributions and how they fit into this structure.

5.1.1 Structure

The structure of most Unity applications, including VVRT, resembles a Model-View-Controller architecture, as is common for applications following the object-oriented paradigm. Unity itself provides a broad abstraction layer that handles most of the controller logic, providing a programming interface to implement custom logic. In our case, this is the camera's ability to pan, orbit and zoom, similar to the built-in Unity editor.

The model part of the architecture works through *GameObjects*. These can represent anything in the scene, from physical objects like spheres to cameras and lights, and even user interface (UI) components. These GameObjects have components attached to them. Each component provides specific functionality to the associated object, and can be reused across different objects. In VVRT, these are often used to add custom C# scripts to objects like the Scene Manager,

[1]This can be found at `https://github.com/LukkeWal/Virtual-Ray-Tracer-Ray-Casting-Support`.

which keeps track of all ray traced objects in the scene. The functions these provide are called when UI elements are interacted with.

The View part, responsible for rendering of the scene and UI, is also mostly taken care of by Unity. Only the UI design and their corresponding scripts are custom to the application. The first, UI design, is done using *prefabs* in the Unity editor. These are reusable templates for GameObjects and their components. All of the UI, including the control panels, tutorial box and render preview panels, is defined in terms of prefabs.

Integration into Ray Tracer

In VVRT, the ray casting level was implemented by inheriting from the ray tracer developed prior, as described in Van der Wal's thesis [17]. They reasoned that while they were reusing a lot of the ray tracer's code for the volume visualization, they did not want to modify the original code. In our extension, we have chosen not to follow this approach, and instead have chosen to integrate new features directly into the existing ray tracer. This is because our volumes are meant to coexist with other objects in the scene, as opposed to being the primary object to visualize. Thus, we can create more complex scenes to show the potential of scattering, such as the Cornell Box level.

5.1.2 Unity Ray Tracer

Perhaps most vital to the tool is the Unity Ray Tracer prefab and script. Together with the scene and ray managers, it stores all settings from the ray tracer configuration panel. Additionally, it contains functions to render the ray tracing scene from the perspective of the virtual camera. These functions can be divided in two categories: those that generate the visualized rays in the scene, and those that render the final image. While these functions contain mostly similar logic and could be merged, they are kept separate to improve the rendering performance, as generating the rays involves some expensive operations.

The most high-level functions are `Render` and `RenderImage`, which are called when the visual rays need to be redrawn and when the user renders an image, respectively. For each of the camera's pixels, they call their corresponding `Trace` and `TraceImage` functions. Both functions are recursive, calling themselves when e.g. a ray hits a reflective surface, to determine the path of the ray after it reflects. The former returns a tree of `RTRay` objects, with as its root the ray originating at the camera. The latter returns a color, which is the color that is obtained by combining all contributions after tracing the ray through the scene, as in a basic ray tracer.

It is these functions where our contributions start. Every time a ray is cast into the scene, and we hit an `RTVolume` object (Section 5.1.3), we either ignore it or call the `SingleScattering` or `MultipleScattering` function, depending on the current ray tracing algorithm setting in the configuration panel. These implement the algorithms as described in section 3.4, as well as some helper functions, like the phase functions.

5.1.3 Volumes

The `RTVolume` object mentioned earlier is an abstract script class. It extends `RTMesh`, which is attached to prefabs such as `Sphere` or `Cube`; inheriting its position, rotation and scale properties. `RTVolume` defines common properties across all volume types, like `Absorption` and `Scattering`, as well as other useful functions.

`RTVolume` has in turn two derived classes. The first, `RTHomogeneousVolume`, contains simple implementations of `RTVolume`'s functions for homogeneous volumes. On the other hand, `RTHeterogeneousVolume` is itself also an abstract class and implements more complicated logic for its data, stored as a voxel grid. This class is extended for each of the available volume types. All of these define their loading procedure in the Unity event function `Start`, which is called when the script is instantiated.

In addition to the existing prefabs (`RT Ray`, `RT Mesh`, `RT Camera` among others), we created a prefab for each of the volume-related scripts. In particular, there is a `RT Homogeneous Volume` prefab, and four heterogeneous volume prefabs: `Bucky`, `Bunny`, `Engine` and `Hazelnut`. To show the event points along the rays, we also need two prefabs, called `RT Absorption Event` and `RT Volume Sample`.

We also implemented a Volume Manager prefab that is currently only used for utility functions related to volumes, but could be used for asynchronous loading of models (with a `VolumeLoader`).

5.1.4 Rays

Before, rays were represented in the ray tracer by small simple cylinders that were elongated as far as the ray's magnitude. To support events on any ray, we have wrapped this cylinder object, instead making the `Ray` prefab contain the cylinder, which is now in `RayBody`. This means that we can attach other prefabs to the `Ray` prefab — in this case, the sample and absorption prefabs.

Rays that are generated inside the scattering algorithms are of a new ray type, namely `Volume`. Another new ray type is `VolumeLight`, which is used to keep volume rendering logic separate from all light rays cast in the scene. For an edge case free implementation, all light rays should account for volumes, but for our simple scenes, this is not necessary.

5.1.5 Levels

All these new prefabs are put into use in two new Unity levels, that add onto the previous 17 levels. They both get their own tutorial text, defined in the

`Game Manager` prefab, which is added to every level by Unity. Both levels are placed after the ray casting level and before the sandbox level. In that last level, users are free to create their own scenes by adding or removing objects [18]. Naturally, we have extended the scene manager to add support for creating and removing volumes, as well as set the number of required points to be in line with the previous objects [10].

5.1.6 User Interface

Lastly, a number of changes and additions have been made to the UI:

- The ray tracer control panel has new controls, as in Figure 4.9.
- `HeterogeneousVolumeProperties`: control panel for heterogeneous volumes displayed when one is selected (Figure 4.12).
- `HomogeneousVolumeProperties`: control panel for homogeneous volumes displayed when one is selected (Figure 4.10).
- `DropdownEdit`: editor for a dropdown menu in the control panel (Figure 4.11).
- `FloatDisplay`: non-interactive editor for floating point numbers, with configurable precision like `FloatEdit`. This is used for the extinction and albedo values in the heterogeneous volume control panel (Figure 4.12).
- `IntEdit`: editor for integer values, used for the number of random walk steps in the ray tracer control panel (Figure 4.9).

5.2 Hardware Support

The application has been developed on a MacBook Air M3 from 2024. It also has been tested on a small number of devices, including recent and more dated Windows PCs.

Both on Mac and Windows, rendering an image takes about three seconds for both single and multiple scattering when the default settings are selected. While multiple scattering takes more resources to produce a similar image to single scattering, we prioritize keeping the rendering times low, and the default settings are set to achieve this.

Chapter 6

Evaluation

A user study was conducted to evaluate the effectiveness of the visualization and tutorial. Seventeen participants played through the two new levels by following the tutorial's instructions. Afterwards, they filled in a digital survey about their experience. In this section, we summarize and analyze the results. The full response data per question, as well as the textual responses, can be found in Appendix A.

The study consists of three parts, each evaluating a different aspect of the tool's educational value. First, we ask about the tutorial to determine how well written it is. Then, we get to the visualization, including both the visualized rays and configurable volumes. The last part covers usefulness and general educational value. Finally, participants are given an opportunity for additional comments, suggestions and feedback, and any encountered difficulties or bugs.

6.1 Study population

Around 40% of participants have taken a course on graphics (either Computer Graphics or Introduction to Computer Graphics and Visualization) before, while the remaining 60% have no experience in any of the courses. Unfortunately, none of the participants have taken the Scientific Visualization course. Interestingly, six out of nine of those with experience with a graphics course indicated 4 out of 5 familiarity with the concepts, with only two responding with 1 (see Figure 6.1).

6.2 Tutorial

Most participants were positive about the tutorial, saying that the concepts (coefficients and algorithms) were explained well, as seen in Figure 6.2. In addition, the pacing was reported to be right, if not a little too fast (Figure

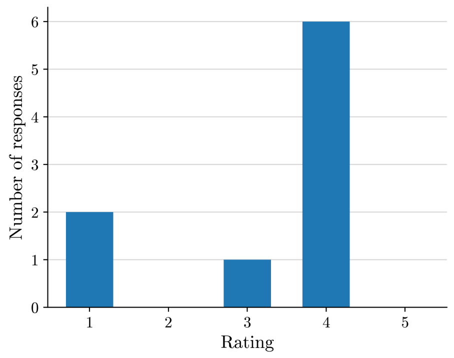


Figure 6.1: Familiarity with concepts of light scattering, of those with prior experience in a computer graphics course

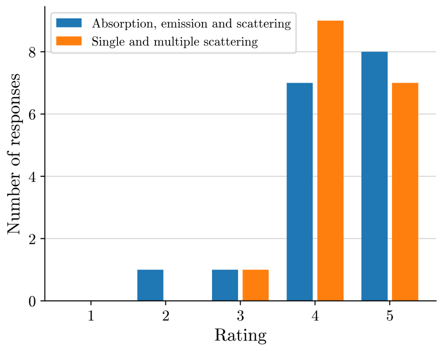


Figure 6.2: Clarity of concept explanation

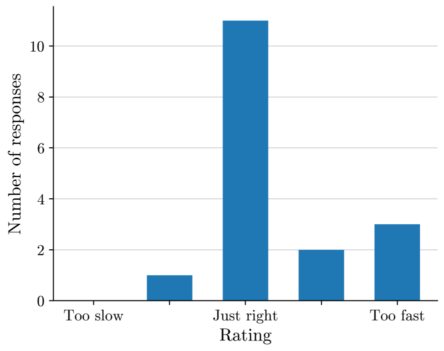


Figure 6.3: Tutorial pacing

6.3). A minor improvement suggested was giving more real-life examples to accompany the coefficient explanations.

6.3 Visualization

As shown in Figure 6.4, all participants rated the visualization highly, with an average of 8.5 out of 10. Multiple positive points were named, such as the random walk, light ray, and sample point visualizations, as well as the tooltips for the configuration options. The users also did not miss any volume parameters that they could not control. The existing parameters were predominantly intuitive (Figure 6.5), but the effects of changing them were somewhat unclear (Figure 6.6).

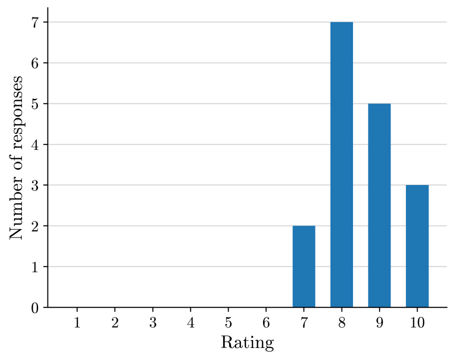


Figure 6.4: Quality and clarity of visualization

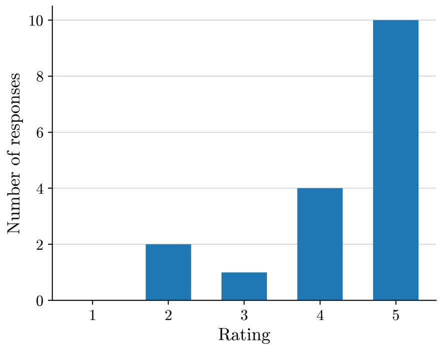


Figure 6.5: Intuitiveness of volume configuration options

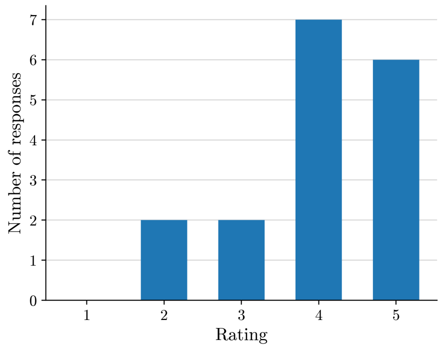


Figure 6.6: Clarity of effects of changing volume parameters

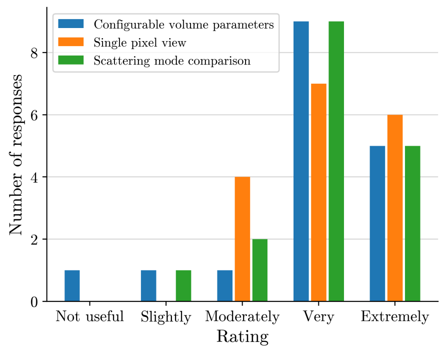


Figure 6.7: Usefulness of features

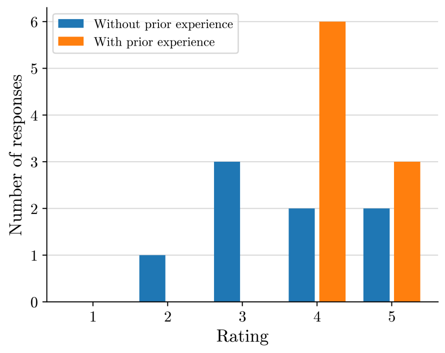


Figure 6.8: Improvement in confidence of explaining scattering in a volumetric renderer

One of the main suggestions for improvement was to adjust the user interface to reduce the amount of interaction needed between adjusting a parameter and rendering an image, adjusting it more, and so on. Currently, users have to interact with the UI six times to complete one of these cycles. Additionally, it was reported that the visualization of multiple scattering was unclear and seemingly did not reflect adjustments in the provided parameters. Lastly, three participants said they would have preferred to see more pixels in the render preview, to see it updating more clearly when parameters were changed.

6.4 Educational value

Finally, participants were asked to evaluate the usefulness and educational value of the levels. All of configurable volume parameters, the single pixel view and the different scattering modes were found useful, and equally so (see Figure 6.7). Users were also asked to what extent the tutorial improved their confidence in explaining how scattering works in a volumetric renderer. Those with experience gave themselves a 4.3 on average (Figure 6.8), showing the tool's promise in education, in particular for more advanced students. Generally, Figures 6.9 and 6.10 show that the new visualization features and tutorials were thought to be engaging and valuable for future students of computer graphics.

6.5 Discussion

The study aimed to investigate whether all important aspects and concepts were taught well (RQ1 and RQ4), if they were visualized well (RQ2), whether users

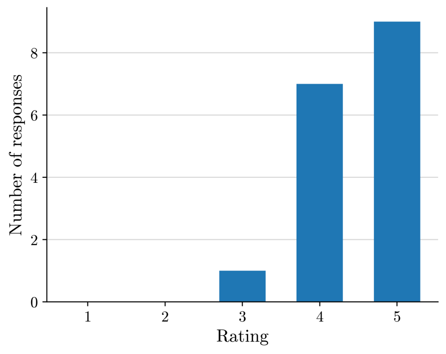


Figure 6.9: Engagement level of new visualization features and tutorials

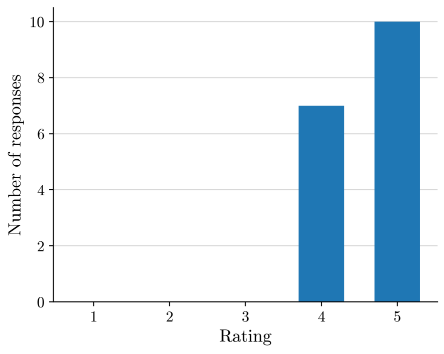


Figure 6.10: Value of new visualization features as educational tool for students of computer graphics

were satisfied with the interaction and the adjustment of parameters (RQ3) and if the tool is valuable for future students of computer graphics (RQ5).

Unfortunately, none of the participants of the study had followed a course on scientific visualization. Persons with prior experience in this field are likely more familiar with ray casting, and ray tracing in general, which results in less insight on these levels compared to the ray casting level. However, since the extension is primarily targeted at students of computer graphics, the study population is still representative.

The results of the user study show that the tutorial's explanation was predominantly clear and paced right, answering RQ1 and RQ4. In addition, participants said the concepts were visualized well overall, but some had difficulties understanding multiple scattering. This likely stems from the fact that the visualization of the random walks, although accurate, are unfamiliar to most other visualizations of similar processes in the field, as mentioned in Section 4.3.2. This is an opportunity for future work to improve on. However, the other participants mentioned no issues with understanding multiple scattering (see Survey Question 9), and we can conclude that the concepts are visualized well (RQ2).

As mentioned in Section 6.3 above, users were not as satisfied with how clear the effects of changing volume parameters were (RQ3). We believe the three separate concerns mentioned to stem from the same issue: a lack of immediate feedback of the effect of changing ray tracer and volume parameters. In fact, all three could be solved at once, by showing the volume not only in the rendered image, but in the scene as well instead of the gray cubes. They would be updated instantly when parameters change. More on this can be read in Section 7.1.

From the responses to the last survey section, starting from Survey Question 14, we see that the application has significant educational value, primarily to students of computer graphics but also to those without prior experience. This answers the final research question, RQ5.

While the feedback reveals opportunities for future work, the tool has received positive feedback, and we can conclude that it can be employed to teach the concepts of scattering in volumetric rendering to future students of computer graphics.

Chapter 7

Conclusion

In this thesis, we have presented a physically based and educational volumetric ray tracing visualization, aimed at students of computer graphics. To facilitate this, we have investigated the most important concepts needed to explain light transport in participating media, and how these concepts can be represented visually, interacted with, and explained in the form of a tutorial. Lastly, a user study shows our extension to address most research questions in a satisfying manner, and that it has sufficient educational value. While the visualization and user interface generally resulted in positive remarks, users experienced some usability difficulties regarding interactivity, which is covered by RQ3. These will largely be addressed by the ideas for future work in the next section.

7.1 Future work

This thesis lays the groundwork for future expansions in volumetric rendering education. Some suggested improvements and additions to the current work are described below.

- The addition that would make the most impact is to render the volumes in real-time inside the scene, as described in Blesinger's thesis [19]. This would involve writing a shader for the GPU, but in turn would greatly improve the user experience and resolve a large part of the friction users have encountered, as the user study has shown.
- Currently, rays entering a volume undergo their paths, depending on the currently selected volume rendering technique. However, these algorithms do not take into account any geometry until the rays exit the volume again, resulting in geometry being cut off at the boundaries of the volume. One approach to solve this could be rendering the volumes and geometry separately and combining the results. Another, more proper approach is to account for geometry inside of the volumetric algorithm.

- VVRT as well as VRT includes additional visualization options, such as contribution-based color and opacity of rays or samples. Similar features can be added to this extension, with a corresponding toggle in the control panel. For example, the samples could be colored based on the combined effects at that point in the volume, and the rays could have color and opacity based on their contribution to the pixel.

- Further optimizations could be developed to improve performance and thus accessibility to students with older hardware. These optimizations could be visualized along with the current ray representations. Some possible additions are octrees [15] or *Sparseleap* [16].

Chapter 8

Acknowledgments

First of all, I would like to thank my supervisors, Steffen Frey and Jiří Kosinka, for their extremely valuable feedback and support. I also want to thank all participants of the user study for devoting their time to it and giving me helpful responses. I hope to participate in yours at some point! Lastly, thank you, Ola, for everything you've done to support me and to keep me grounded throughout this journey. I couldn't have done this without you.

Bibliography

[1] Yangdong Deng et al. “Toward Real-Time Ray Tracing: A Survey on Hardware Acceleration and Microarchitecture Techniques”. In: *ACM Comput. Surv.* 50.4 (Aug. 2017), 58:1–58:41. ISSN: 0360-0300. DOI: `10.1145/3104067`.

[2] Jake A Russell. “An Interactive Web-Based Ray Tracing Visualization Tool”. In: (June 1999).

[3] Nick Vitsas et al. *Rayground: An Online Educational Tool for Ray Tracing.* ISSN: 1017-4656. The Eurographics Association, 2020. ISBN: 978-3-03868-102-1. URL: `https://doi.org/10.2312/eged.20201027` (visited on 06/25/2025).

[4] Michael Stewart. “Ray Tracing Teaching Tool”. In: (Aug. 2022). URL: `https://hdl.handle.net/1969.1/196523` (visited on 04/23/2025).

[5] Rikard Storheil Eriksen. “Visualizing the Principles of Ray Tracing Algorithms in Virtual Reality”. MA thesis. NTNU, 2022. URL: `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3019913` (visited on 04/23/2025).

[6] Nuno Verdelho Trindade et al. “Improving Ray Tracing Understanding With Immersive Environments”. In: *IEEE Transactions on Learning Technologies* 17 (2024), pp. 1921–1934. ISSN: 1939-1382. DOI: `10.1109/TLT.2024.3436656`.

[7] Christiaan Gribble et al. “Ray tracing visualization toolkit”. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* I3D ’12. New York, NY, USA: Association for Computing Machinery, Mar. 2012, pp. 71–78. ISBN: 978-1-4503-1194-6. DOI: `10.1145/2159616.2159628`.

[8] Willard A. Verschoore de la Houssaije et al. *Virtual Ray Tracer.* ISSN: 1017-4656. The Eurographics Association, 2022. ISBN: 978-3-03868-170-0. URL: `https://doi.org/10.2312/eged.20221045` (visited on 06/25/2025).

[9] Chris S. van Wezel et al. “Virtual Ray Tracer 2.0”. In: *Computers & Graphics* 111 (Apr. 2023), pp. 89–102. ISSN: 0097-8493. DOI: `10.1016/j.cag.2023.01.005`.

[10] Peter Jan Blok. "Gamification of Virtual Ray Tracer". Bachelor's thesis. 2022. URL: https://fse.studenttheses.ub.rug.nl/27596/ (visited on 06/25/2025).

[11] Roan Rosema. "Adapting Virtual Ray Tracer to a Web and Mobile Application". Bachelor's thesis. 2022. URL: https://fse.studenttheses.ub.rug.nl/27894/ (visited on 06/25/2025).

[12] Bora Yilmaz. "Acceleration data structures for Virtual Ray Tracer". Bachelor's thesis. 2022. URL: https://fse.studenttheses.ub.rug.nl/27838/ (visited on 06/25/2025).

[13] Lukke Van Der Wal et al. "VVRT: Virtual Volume Raycaster". In: *EuroVis 2025 - Education Papers* (2025). ISBN: 9783038682738 Publisher: The Eurographics Association. DOI: 10.2312/EVED.20251021.

[14] L. G. Henyey and J. L. Greenstein. "Diffuse radiation in the Galaxy." In: *The Astrophysical Journal* 93 (Jan. 1941). Publisher: IOP ADS Bibcode: 1941ApJ....93...70H, pp. 70–83. ISSN: 0004-637X. DOI: 10.1086/144246.

[15] Jiaze Li et al. "Real-time volume rendering with octree-based implicit surface representation". In: *Computer Aided Geometric Design* 111 (June 2024), p. 102322. ISSN: 0167-8396. DOI: 10.1016/j.cagd.2024.102322.

[16] Markus Hadwiger et al. "*SparseLeap* : Efficient Empty Space Skipping for Large-Scale Volume Rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 974–983. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2017.2744238.

[17] Lukke van der Wal. "Virtual Ray Tracer: Ray Casting Support". Bachelor's thesis. 2023. URL: https://fse.studenttheses.ub.rug.nl/31739/ (visited on 06/26/2025).

[18] Chris van Wezel. "A Virtual Ray Tracer". Bachelor's thesis. 2022. URL: https://fse.studenttheses.ub.rug.nl/26455/ (visited on 07/17/2025).

[19] Philip Blesinger. "Real-time Visualisation of Volume Raycasting in Virtual Ray Tracer". Bachelor's thesis. 2024. URL: https://fse.studenttheses.ub.rug.nl/34382/ (visited on 04/23/2025).

List of Figures

List of Algorithms

Appendix A

All Survey Responses

The following is an exhaustive list of all responses to the user study questionnaire. Quantitative questions are grouped by response, and open-ended questions are ordered by participant.

1. Have you taken any of the following courses?

Course	**Count**
Introduction to Computer Graphics and Visualization, Computer Graphics	4
Introduction to Computer Graphics and Visualization	2
Computer Graphics	3
None of the above	8

2. If you did, before going through the tutorial, how familiar were you with the concepts of light scattering?

Rating	**Count**
1: not familiar at all	2
2	0
3	1
4	6
5: very familiar	0

3. How well did the tutorial explain the concepts of absorption, emission, and scattering?

Rating	**Count**
1: badly	0
2	1
3	1
4	7
5: very well	8

4. How well did the tutorial explain the concepts of single and multiple scattering?

Rating	**Count**
1: badly	0
2	0
3	1
4	9
5: very well	7

5. Was the pacing of the tutorial appropriate?

Rating	**Count**
1: too slow	0
2	1
3: just right	11
4	2
5: too fast	3

6. How would you describe your general experience with the tutorial levels?

P	Response
1	no problems and quite educational
2	no experience with previous levels
3	It was fun to play around with the parameters and concepts where generally explained well.
4	Good the instructions were clear
5	I enjoyed the way concepts were explained and that it waited until you completed a step until it allowed you to move forward. It might be nice if it told you what values to choose and then explain the effects of those values afterwards. Or specific things in the result that are particularly interesting. What pixel to choose to see the best result also.
6	The tutorial levels were nicely paced and split into individual "chapters". There was quite a bit of reading before actually getting to single and multiple scattering which I appreciated. I only wish the multiple scattering part was a bit more in-depth as I wasn't completely sure what what individual settings did. Furthermore, the default model was rather sparse which diminished the render preview.
7	Single scattering was easy to understand, multiple scattering was a little bit harder to understand without any prior knowledge of the concept.
8	Misschien kijken of je minder tasks in de tutorials per level kunt doen, of het tutorial panel groter. Ik vond het vervelend dat de tutorial al gelijk naar de volgende task ging zonder dat ik dat deed. Ik vond het ook leuk dat je wel een beetje zelf moest zoeken in de interface. En ik vond het konijn leuk, die zou ik erin houden.
9	Ik vond het fijn zo in stappen erdoorheen te gaan, minder intimiderend. de opdrachten gingen gelijk verder terwijl ik soms nog niet alles had gelezen of gedaan, omdat ik dan iets aanpaste om uit te proberen, dus het is denk ik fijner als je zelf doorklikt.
10	Bij het benoemen van 'emission' (maar ook de andere concepts), zou een voorbeeld handig zijn ("bijv. vuur" bij emission, of "bijv. stoomtreinrook" bij absorption). En bij single vs multiple kon ik zien, maar werd bij multiple niet zozeer benoemd dat je dan alle kanten op kunt gaan.
11	Everything was as I expected, had fun
12	Good explanation of different important things. Overall clear and good experience
13	Amazing, perfect for people with attention disorders

14	It had a good pacing, and it was relatively easy to follow. Some concepts could maybe use a more in-depth explanation but other than that, it went relatively smoothly
15	Tutorial is quite good however the user panel should change because the Ray has too many options.
16	Few kinks left to make it good
17	I found the tutorial useful. However, to make it more user-friendly, I would suggest a few improvements that may help users better understand its content: Clearly highlight the user's required actions and indicate that the next level will be unlocked only after completing the current task. Fix the page numbering when navigating backward. Currently, the user can navigate to previous tasks within a level, but the page number does not update accordingly. For example, after completing task 7, the user is taken to page 8 of 16. If they go back to task 7, the page number still shows 8 of 16 instead of updating to 7 of 16, which can be confusing. Other than that, the application looks great. In my opinion, just a few small improvements would make it feel more like a game, where users can clearly see their progress and feel more motivated to continue.

7. How would you rate the quality and clarity of the visualization?

Rating	**Count**
1	0
2	0
3	0
4	0
5	0
6	0
7	2
8	7
9	5
10	3

8. What aspects of the visualization did you particularly like?

P	**Response**
1	I liked seeing the light rays and how they travel from source to wherever they end up
2	-
3	The visualized ray marching and non uniform volume (bunny).
4	Liked how the random walk was visualized and the preview
5	I enjoyed how clearly everything was shown and that I can actually see the effects.
6	I enjoyed the visualization of single scattering and how changing the step size affected the render preview. I thought this was lacking in the multiple scattering section.
7	Animated single pixel view was quite useful in understanding the process.
8	Je kon de rays goed volgen, waar ze heen gingen. Het was goed uitgelegd waar er sample points waren.
9	het feit dat je het in andere situaties kon visualiseren, ook al waren ze simpel.
10	To be able to look at different angles and to zoom, to see what's happening.
11	How easy it is to interact with the scene + the slowmo ray simulation
12	The visibility of the light rays and it was clear
13	the amount of rays
14	The animation, the option panels, the accurate and quick rendering
15	I can rotate.
16	dynamic fast changes
17	I feel comfortable with the application because it reminds me autocad and other similar tools

9. What would you change or improve about the visualization?

P	**Response**
1	Maybe have different types of surfaces in the environment to see more about the scattering process in practice
2	-
3	Not much, rendering on the GPU is not necessary however it would be a nice addition. Aside from that everything is clear :)
4	Nothing
5	Not major, but be able to change the speed of the ray projection.
6	As previously mentioned, I would improve the multiple scattering section and make the visualization reflect adjustments in the provided parameters.
7	It's quite hard to compare the rendered images of different volume parameter configurations.
8	Ik snapte niet wat multiple scattering was, in de visualizatie.
9	misschien iets meer uitleg bij het laatste level dat het een mirror was want het duurde even (bij mij) om te begrijpen waar ik naar keek.
10	Maybe standard not a 9 pixel preview but a bit more :)
11	maybe put the render image button somewhere accesable from any tab
12	I would like to be able to have more pixels for rendering so the image becomes more clear
13	More pixels
14	I dont really have any suggestions for improvement, it was done well
15	N/A
16	Visualization no but the ui and tutorial
17	Sometimes, after moving the camera and other objects, I lose track of their position along the axes. For example, when I want to return the camera to its initial position, it takes some time to figure out which axis values need to be adjusted. It would be helpful to have a "reset" action to quickly return objects to their default positions.

10. How intuitive were the volume configuration options (e.g., absorption, emission, scattering controls)?

Rating	**Count**
1: very unintuitive	0
2	2
3	1
4	4
5: very intuitive	10

11. How clear were the effects of changing the volume parameters?

Rating	**Count**
1: very unclear	0
2	2
3	3
4	7
5: very clear	6

12. Were there any volume coefficients or parameters you wanted to control but couldn't?

P	**Response**
1	No
2	idk
3	Not that I'm aware of... (unless there are hidden coefficients)
4	None
5	i think it covered the basics nicely
6	No, the provided coefficients and parameters were sufficient
7	Animation Speed
8	no
9	no / g (ik kon geen effect merken maar wel aanpassen)
10	No
11	no
12	None
13	Being able to draw a field where scattering can occur
14	not really, essentially anything that I wanted to control or change was possible to adjust
15	N/a
16	No
17	No

13. What did you like/dislike about configuring the volumes?

P	**Response**
1	It was all pretty standard so nothing really stands up in a positive or negative way
2	missing units
3	I did not always see a big change, however that is quite subjective, it might be the parameters that I changed.
4	I disliked that I had to click on the volume to configure
5	it was a bit hard to find at first but once you do it it is fine
6	I very much enjoyed seeing the explanations of what each parameter/coefficient does upon hovering above it, I thought this was lacking in other levels of the Virtual Ray Tracer (VRT). I wish the parameters could somehow be visualized rather than having to render the image each time after changing a parameter and then trying to remember the difference from the previous render off the top of my head.
7	It was enjoyable to explore different homogeneous volume configuration options in the Cornell Box level after finishing the first level.
8	Absorption en emission waren duidelijk, maar scattering kon ik minder goed het effect van zien.
9	ik vond het leuk wat te veranderen en daar het effect van te zien, waardoor je dus kan leren hoe het werkt door zelf uit te proberen.
10	Only question was: why the limitation in e.g. the number of walks?
11	the responsiveness
12	N.A.
13	How the rays scatter because of it
14	I would have liked having the possibility to compare renders in a more efficient way
15	N/a
16	They were a bit hidden in the ui
17	Nothing

14. How useful were the following features in helping you understand scattering?

Feature	**Rating**	**Count**
Configurable volume parameters	1: not useful	1
	2: slightly	1
	3: moderately	1
	4: very	9
	5: extremely	5
Single pixel view	1: not useful	0
	2: slightly	0
	3: moderately	4
	4: very	7
	5: extremely	6
Scattering mode comparison (zero, single and multiple scattering)	1: not useful	0
	2: slightly	1
	3: moderately	2
	4: very	9
	5: extremely	5

15. To what extent did the tutorial improve your confidence in explaining how scattering works in a volumetric renderer?

Rating	**Count**
1: not confident at all	0
2	1
3	3
4	8
5: very confident	5

16. How engaging did you find the new scattering visualization features and tutorials?

Rating	**Count**
1: not engaging at all	0
2	0
3	1
4	7
5: very engaging	9

17. How valuable do you think these new scattering visualization features would be as an educational tool for students of computer graphics?

Rating	**Count**
1: not valuable at all	0
2	0
3	0
4	7
5: very valuable	10

18. Do you have any additional comments, suggestions, or feedback?

P	**Response**
1	I guess there wasn't really anything indicating that i finished the level
2	Good work Jasperek!
3	
4	no
5	very nice!!
6	I think the VRT is a great tool for learning Computer Graphics topics and seeing them visualized. The volumetric rendering section fits nicely in the list of available levels. Apart from the suggestions mentioned in the prior questions (improved volume visualization and multiple scattering visualization), I don't have any additional suggestions.
7	
8	fun experiment, would do again. make sure the program works !!!!!!
9	je hebt hard gewerkt ->beloon jezelf
10	Brilliant :)
11	provide a mouse
12	
13	Love it
14	Aside from small comments like adding a toggable feature to compare renders, or maybe a bit more in-depth explanations, I think the project was done well and it succeeds in its mission to educate perfectly.
15	n/a
16	
17	If the goal is for it to be an educational tool, it would be better to include more tests or quizzes related to the terminology and information presented in the tutorial. The current tutorial helps users better understand the application and partially covers the theory checking. It would also be helpful to present the tutorial in a document or full-page mode, and to highlight or bold key terminology.

19. If you experienced any technical issues (performance problems, usability difficulties, bugs), please describe them below.

P	Response
1	
2	everything works perfect
3	
4	Phase function had a bug with its parameter
5	
6	I had some miner issues that can be applied to the entire VRT, not only the scattering levels. I did not like the "Tutorial Tasks" automatically progressing without me explicitly pressing the forward button. The Cornell Box had a weird bug where the color of the volume was stuck to black when using multiple scattering.
7	
8	sometimes the rendering stopped at 29% and then the program crashed.
9	
10	Wat mij betreft mag de legenda standaard in beeld.
11	no tech issues
12	
13	No vsync. Also make it run on the gpu
14	This isn't necessarily a bug but I was informed that I wasn't able to check the emission with multiple scattering and that would have added to the value of the program in educating its user with these phenomenon, so I look forward to the final product with the potentially added functions
15	n/a
16	
17	