university of groningen / faculty of science and engineering / computing science

Virtual Ray Tracer: Ray Casting Support

Bachelor's thesis
November 15, 2023
Student: Lukke van der Wal
Primary supervisor: Steffen Frey
Secondary supervisor: Jiří Kosinka

Contents

Abstract

An application called *Virtual Ray Tracer* was developed to visualize the ray tracing process. Its goal is to help students of the computer graphics bachelor course at the Rijksuniversiteit Groningen, or anyone else interested, learn about this rendering technique.

In this thesis, we expand the Virtual Ray Tracer by adding support for ray casting. Ray casting is an important technique in computer graphics visualisation. We present its principles to the user by drawing the volume as a transparent grid. Through this grid, we illustrate the rays and the samples that they take. Components such as the transfer function and compositing methods are visible and changeable. Our goal is to help students of the masters' course Scientific Visualization at the Rijksuniversiteit Groningen gain a more detailed understanding of volume ray casting.

We did a user study to evaluate how much the application helped users understand ray casting. Overall these results were positive with many users saying that they learned more about ray casting from the application.

1 Introduction

Within Computer Graphics, ray tracing and ray casting are rendering techniques to turn 3- dimensional objects and scenes into 2D images. They both simulate a ray of light travelling through 3D space and use the interactions of the ray with objects in the scene to calculate a colour value for a pixel. Ray tracing only calculates interactions with objects' surfaces. Volume ray casting, on the other hand, interacts with the object at every point along the ray that intersects with the object. It uses this extra information to visualize objects with volumetric information, like an MRI scan.

Ray casting works on object data that not only has information about its surface but also about its interior. The data is usually represented as a grid of voxels. Each voxel contains a scalar value, in the case of an MRI this scalar can represent the density of the object. For every pixel in the final image, a ray is cast through the grid. As the ray travels through the grid it considers the scalar values of the voxels it travels through. These scalar values are transferred into corresponding RGBA values by the transfer function. While traveling along a ray the RGBA values are combined into a single RGBA value using a compositing method. The compositing method can for example be the maximum or average. This single RGBA value is finally displayed on the pixel corresponding to the ray cast by the camera.

Virtual Ray Tracer by Chris van Wezel and Willard Verschoore de la Houssaije [1, 2, 3] was developed to help students learn about ray tracing, see Figure 1. It has been expanded multiple times. Support for more complex ray tracing techniques was added by Jesper van der Zwaag [4], acceleration data structures were added by Bora Yilmaz [5], gamification of the application was done by Peter Jan Blok [6], and adaptation to web and mobile was done by Roan Rosema [7]. Together they created Virtual Ray Tracer 2.0 [8] that this project builds upon.

The goal of this thesis is to expand Virtual Ray Tracer 2.0 by adding support for volume ray casting. This technique will allow those interested in it to learn how volumetric objects with varying materials below their surface, represented as voxel grids, can be visualized using ray casting.

In this thesis, we first provide relevant background (Chapter 2). We present ray tracing, Virtual Ray Tracer, and ray casting. We then introduce the method we used to visualize ray casting within the Virtual Ray Tracer (Chapter 3). After this, we demonstrate how we implemented these changes (Chapter 4). In the user study (Chapter 5), we evaluate the project, and we end with a conclusion (Chapter 6).

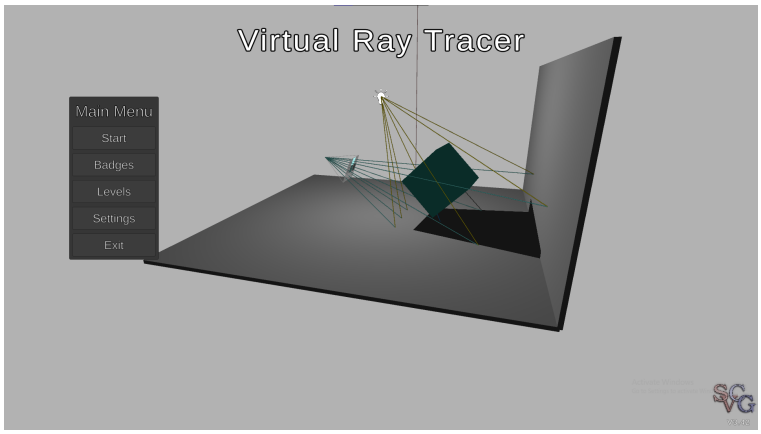


Figure 1: The main menu of Virtual Ray Tracer

2 Background

2.1 Ray Tracing

Within Computer Graphics, ray tracing is a technique used to make a 2D image out of 3D objects and scenes. It is done by simulating a ray of light travelling through this 3D scene. A grid of pixels is placed in front of a camera, this camera acts as the eye. For every pixel, a ray is shot from the camera and through its corresponding pixel. This ray then travels through the scene whenever it intersects with an object a calculation is made, see Figure 2. Whether the ray bounces, in which direction and if it is scattered into multiple rays depends on the material of the object it encounters. All objects a ray and its subrays encounter will determine the final colour of the pixel that the ray passes through.

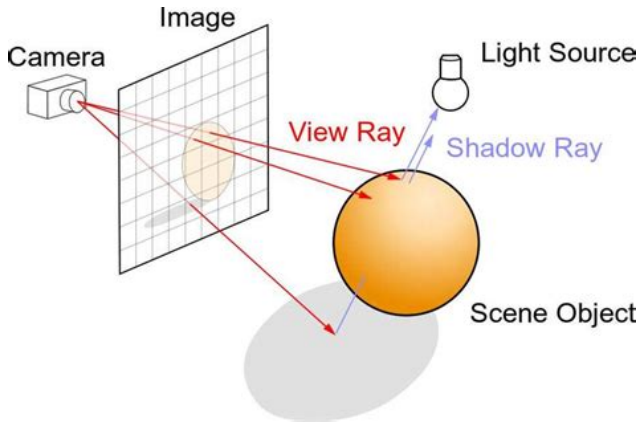


Figure 2: Drawing of the Ray Tracing algorithm, image adopted from [9]

2.2 Virtual Ray Tracer

There are many applications that aid in visualizing ray tracing techniques, such as [10, 11, 12, 13]. These are made to analyze ray tracing implementations and improve them. They are, however, not aimed at education. Some applications do focus on being educational [14, 15], but they have no visualization of the ray tracing process. Nevertheless, the idea of a ray tracing educational tool is not new. There is an old java implementation [16] from 1999, but it is no longer up to date with today's standards and is no longer available. A newer application that is aimed at developers and students [17] exists, but it requires users to plug in their ray tracing programs, which is no trivial task; this makes it difficult to use for educational purposes.

This is where *Virtual Ray Tracer* [1], commonly abbreviated as *VRT* comes in. It aims to improve the learning process by providing a visualization of ray tracing techniques. It was created to help students of the computing science bachelor at the Rijksuniversiteit Groningen study for the Computer Graphics course. However, it is available to anyone and aims to help those interested in ray tracing learn about the technique. It is created with the Unity engine.

Virtual Ray Tracer has multiple levels demonstrating different techniques used in ray tracing. Level 1 is an introduction to the application and guides users on how to use it. Level 2 shows the basics of ray tracing. Levels 3 through 14 explain specific techniques within ray tracing, getting more advanced as you go further through them. We will be adding level 15: ray casting. Level 16 is a sandbox level where the user is free to

create scenes and experiment with the application.

2.3 Ray Casting

Ray casting is similar to ray tracing, it also turns 3-dimensional objects and scenes into 2d images [18]. They both simulate a ray of light traveling through 3D space and use the interactions of the ray with objects in the scene to calculate a colour value for a pixel. However, ray tracing only calculates interactions with objects' surfaces. Volume ray casting, on the other hand, interacts with the object at every point along the ray that intersects with the object. It uses this extra information to visualize objects with volumetric information, like an MRI scan.

2.3.1 Data Representation

Data for ray casting can come from simulations or making a scan of the density of an object. The data representation that we will use will be voxel grids. A voxel grid is a 3-dimensional grid with scalar values. Every scalar value represents a density value of the volumetric object. These densities range from 0 being low density, to 1 being high density. Every cell of the voxel grid contains a single density value and is called a voxel.

2.3.2 Casting a Ray & Taking Samples

Just as with ray casting a ray is shot from the camera and through its corresponding pixel. When this ray enters a voxel grid it starts taking samples. These samples are a set distance apart. When taking samples the sample coordinates rarely fall exactly on a single voxel. To account for this the 8 voxels surrounding the sample coordinates are taken into account, and their density values are trilinearly interpolated. After which we are left with the density value of the sample.

2.3.3 Transfer Function

This sample density is then translated into a colour using a transfer function. The transfer function we will use is a colour lookup table. This table has 2 columns: the first is an ascending list of density values, and the second has their corresponding RGBA value. Just as with taking a sample, the sample density will rarely be the same as a density value in the colour lookup table. So here linear interpolation is once again used. Note however that not the densities, but their corresponding colours are interpolated, giving us the sample colour.

2.3.4 Compositing Method

As a ray travels it composites the sample colours into a single composited colour. It does this using a compositing method. In our algorithm, we will use front-to-back ray casting, so the samples close to the camera are considered first during compositing. We provide the user with four compositing methods: Accumulate, Average, Maximum, and First. In the following formulas let:

C_n = Composited colour at sample number n
$C_0 = (0, 0, 0)$
O_n = Composited opacity at sample number n
$O_0 = 0$
c_n = colour of sample number n
o_n = opacity of sample number n
d_n = density of sample number n
$f(d)$ = transfer function colour of density d
T = Termination point for early ray termination
X = Opacity cutoff value
Y = Density matching value

T is used for early ray termination. It is a boolean expression. When it evaluates to true the current sample and all further samples are disregarded. The opacity cutoff value is a pre-set value between 0 and 1 that is used in the formulas for Accumulate and First.

- Accumulate

$$C_n = C_{n-1} + (1 - O_{n-1}) \cdot c_n$$
$$O_n = O_{n-1} + (1 - O_{n-1}) \cdot o_n$$
$$T = O_n >= X$$

- Average

$$C_n, O_n = f(\frac{\sum_{i=1}^{n} d_i}{n})$$

- Maximum

$$C_n, O_n = f(\max(d_1, ..., d_n))$$

- First

$$C_n, O_n = \begin{cases} f(D), \exists n \in [1, n], d_n = D \\ f(0) \end{cases}$$
$$T : C_{n-1} \neq C_0$$

In the end the last C value will be the final composited color, and therefore the final pixel colour of that ray.

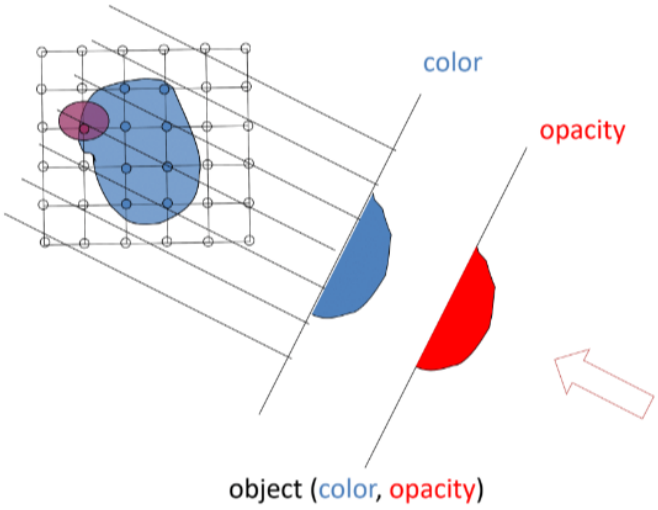


Figure 3: 2D representation of ray casting. Note that the rays in this diagram are parallel, in our application, they will originate from a single point and spread out. Image adopted from [19]

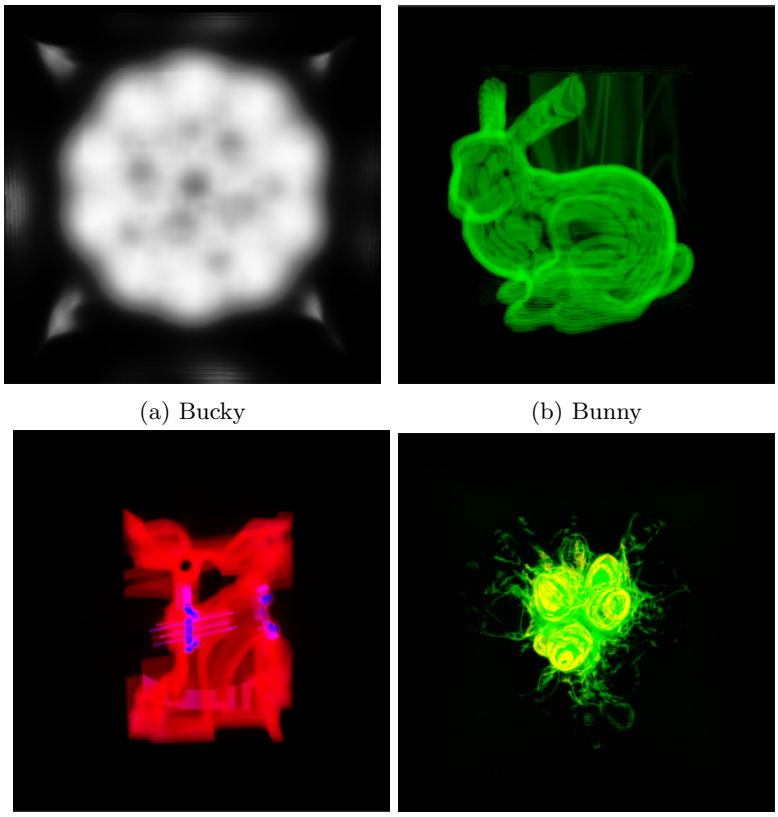
3 Method

In this chapter, we present the methods we used to visualize ray casting. We start by giving an overview of the methods for the basics of ray casting, then show additional controls that we added.

3.1 Basic Ray Casting

3.1.1 Voxel Grid

The voxel grid is visualized using a transparent gray box. What is inside these boxes can only be seen in the render preview window, or when rendering a high-resolution image. We made 4 voxel grid types available: Bucky, Bunny, Engine and Hazelnut; see Figure 4. While playing, the user can select a different voxel grid with a dropdown menu. When changing voxel grids, the Transfer Function will be set to the recommended transfer function for this type.

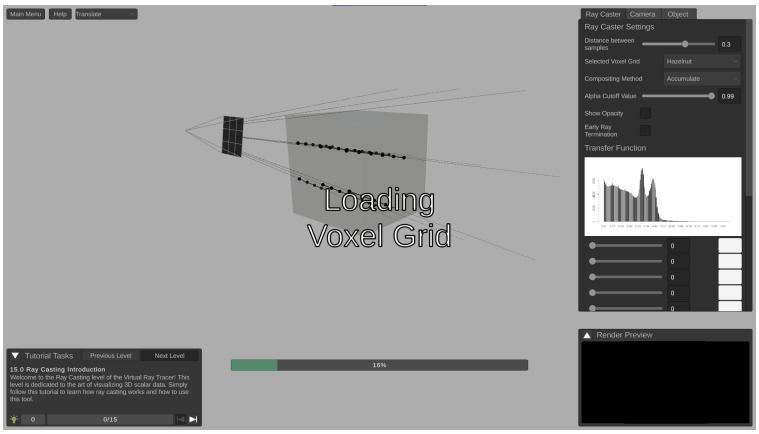


(a) Bucky (b) Bunny

(c) Engine (d) Hazelnut

Figure 4: The 4 voxel grids with default transfer table, compositing method accumulate and distance between samples=0.1

As loading a voxel grid can take quite some time, see Figure 5, we added a loading bar so users are not left wondering why their application is not responding.



H

Figure 5: Loading in the hazelnut voxel grid type, the largest of the four assets

3.1.2 Samples

Taking the samples is visualized by drawing a ray through the voxel grid. Along this ray samples are visualized using opaque spheres whose colour corresponds to the sample colour. The colour of the ray is the same as its final composited colour. When a ray is drawn the samples of that ray will be drawn with it. When the "Animate" setting is set to *true*, the samples are drawn as if they are taken as the ray travels through the grid.

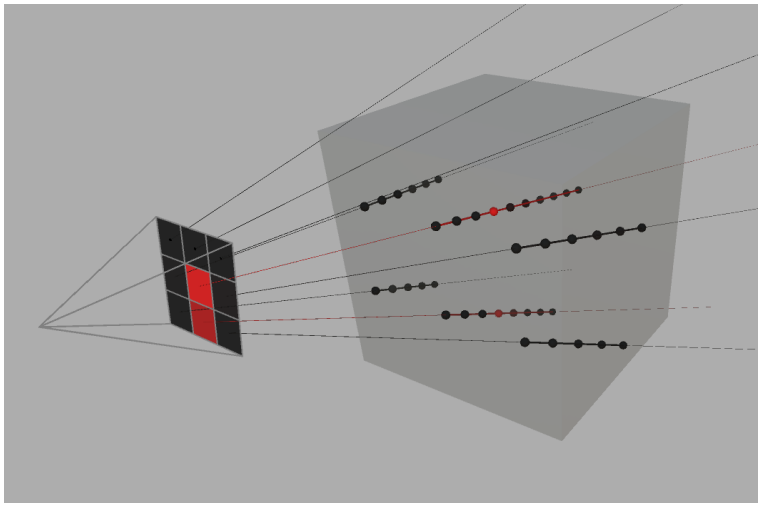


Figure 6: Samples of the engine voxel grid

3.1.3 Transfer Function

The transfer function is visualized in the control panel. There the color lookup table can be viewed and edited. A suggested transfer function is loaded every time the voxel grid type is changed, see Figure 7. These suggested transfer functions are based on histograms of the voxel grid data. These histograms are also visible

in the control panel.

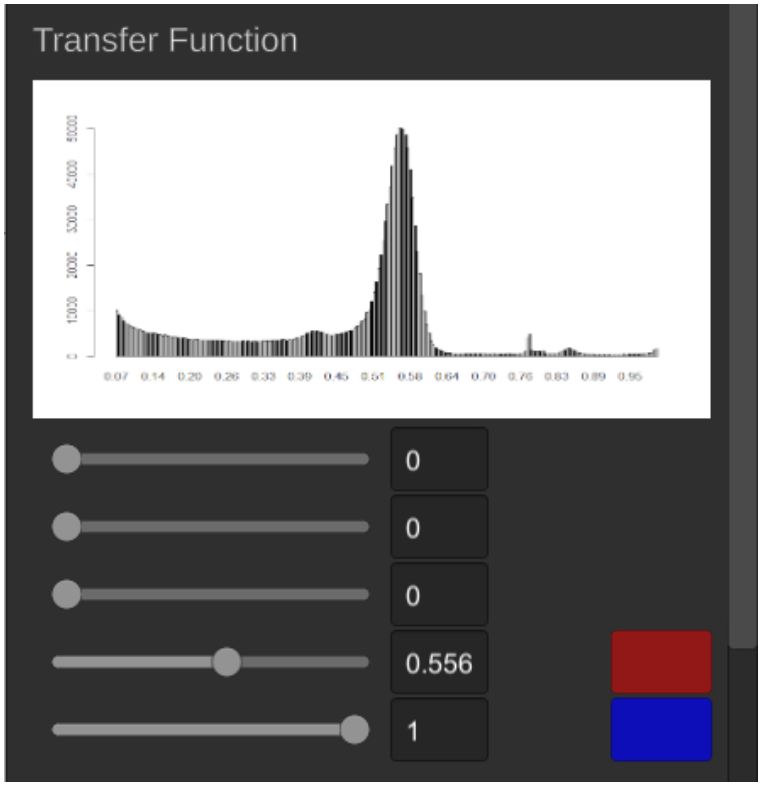
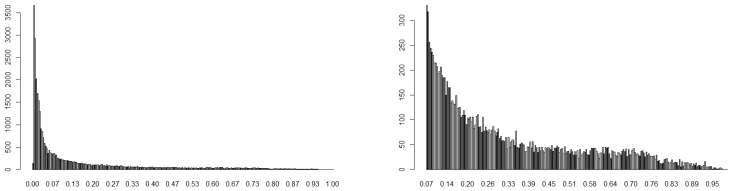


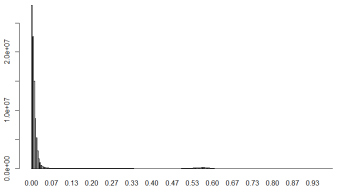
Figure 7: Reccomended transfer function for the engine voxel grid

Most voxel grids will have a lot of space around the object that they represent. This can also be seen in the full histograms shown in Figure 8. This makes it hard to see which densities have spikes and might therefore be interesting to give an entry in the transfer functions color lookup table. To account for this, we made histograms that ignore any density under 0.07. As can be seen in the cropped histograms in Figure 8, these cropped histograms are much more informative.

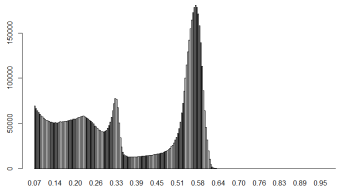


(a) Bucky full

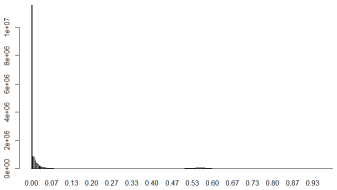
(b) Bucky cropped



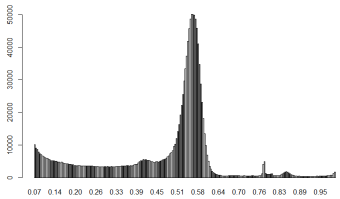
(c) Bunny full



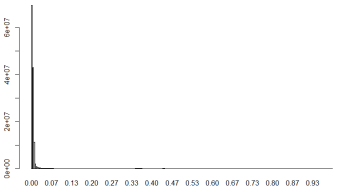
(d) Bunny cropped



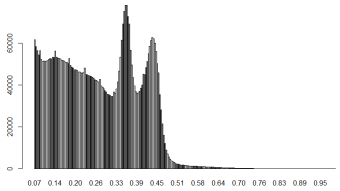
(e) Engine full



(f) Engine cropped



(g) Hazelnut full



(h) Hazelnut cropped

Figure 8: Histograms of the 4 voxel grids. On the left are the full histograms, on the right are the cropped histograms where values under 0.07 are ignored

3.1.4 Compositing Method

The compositing method is visualized through the ray calculation window, see Figure 9. The ray and its pixel will also have the same colour as the final composited colour. To view how the composited colour is calculated one can click on any pixel in the Render Preview window. This will open the ray calculation window for that pixel.

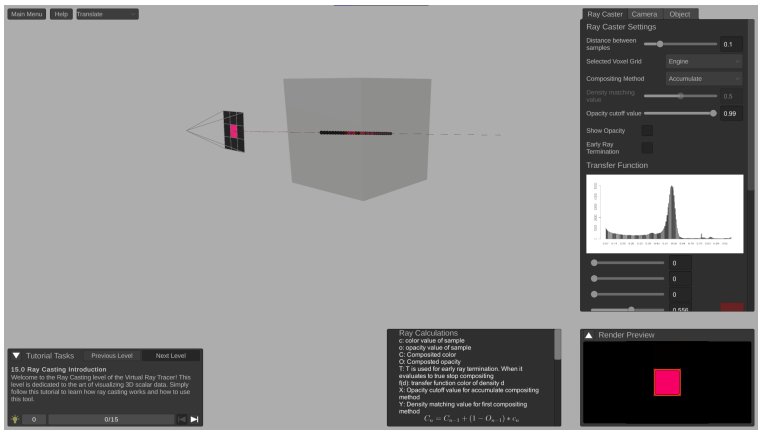


Figure 9: Ray Calculation window is opened at the bottom right next to the render preview window

At the top of the ray calculation window, the variables used in the formulas and the table below are explained, see Figure 10a. Then the formula for the composited colour C_n and the composited opacity O_n are given, as can be seen in Figure 10b. If the compositing method is 'accumulate' or 'first', the formula for early ray termination T is also given. Subsequently, a table begins. The table has 4 columns:

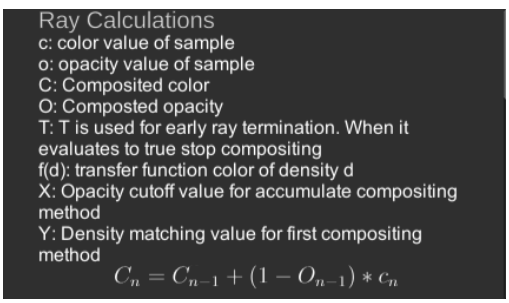
d = Density of sample

$c\&o$ = Sample RGBA

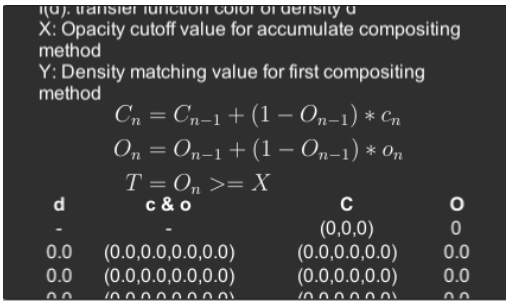
C = Composited Color

O = Composited Opacity

As shown in Figure 10c, the table has a row for every sample taken on the selected ray. The composited colour changes as more samples are recorded. When ray termination occurs a row is inserted displaying the text "Ray Termination". From this point on C and O will no longer change.



(a) Variables



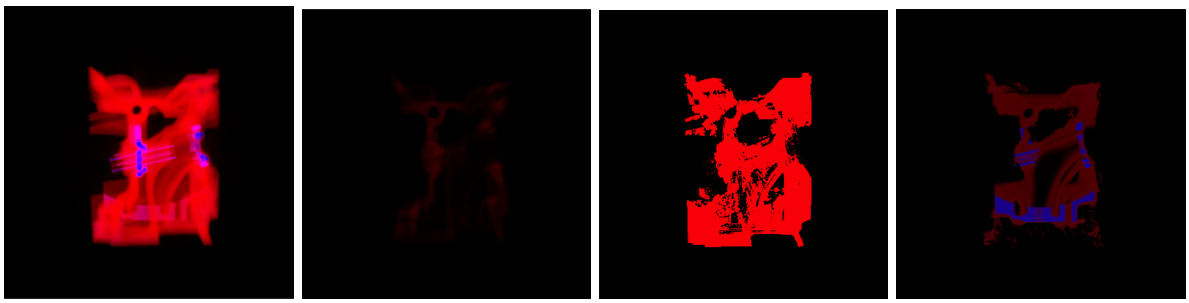
(b) Formulas for Accumulate

d	c & o	C	O
-	-	(0,0,0)	0
0.0	(0.0,0.0,0.0,0.0)	(0.0,0.0,0.0)	0.0
0.0	(0.0,0.0,0.0,0.0)	(0.0,0.0,0.0)	0.0
0.0	(0.0,0.0,0.0,0.0)	(0.0,0.0,0.0)	0.0
0.0	(0.0,0.0,0.0,0.0)	(0.0,0.0,0.0)	0.0
0.0	(0.0,0.0,0.0,0.0)	(0.0,0.0,0.0)	0.0
0.2	(0.4,0.0,0.0,0.1)	(0.4,0.0,0.0)	0.1
0.9	(0.3,0.0,0.7,0.6)	(0.7,0.0,0.6)	0.6
0.5	(0.9,0.0,0.0,0.3)	(1.0,0.0,0.6)	0.7
Ray	Ray Termination	Ray Termination	Ray
0.1	(0.2,0.0,0.0,0.1)	(1.0,0.0,0.6)	0.7

(c) Table with samples and activated early ray termination

Figure 10: All the information in the ray calculation window

Of course, selecting a different compositing method will also affect how the resulting image looks. The effects of the four available compositing methods can be see in Figure 11.



(a) Accumulate (b) Average (c) First at 0.56 (d) Max

Figure 11: Compositing method results on the engine voxel grid

3.2 Controls

We expanded the control panel of the ray tracer with new settings and hid some controls that are irrelevant or incompatible with ray casting. The controls that the Ray Tracer and Ray Caster do share are:

- Hide no hit rays

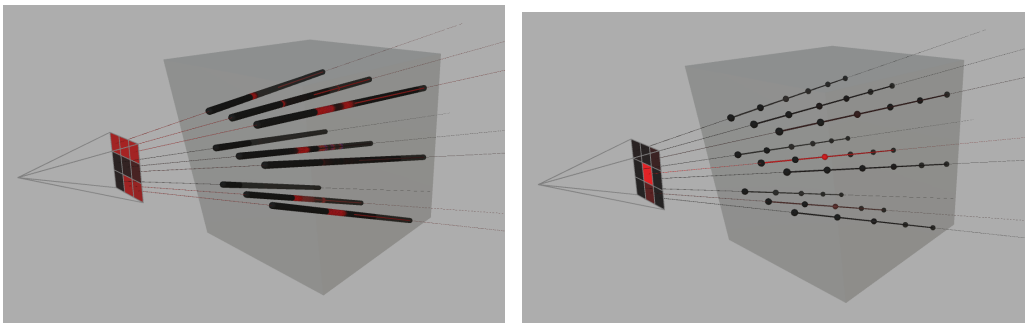
- Animate rays
- Animate sequentially
- Loop animation
- Animation speed
- Supersampling
- Supersampling animation
- Render image
- Open image
- Fly to virtual camera

Some of the new controls have already been mentioned: selecting a voxel grid, selecting a compositing method, and changing the transfer function. These are the bare controls that are needed for ray casting. To allow for experimenting with the application and different visualization styles we added four more settings described below.

3.2.1 Distance Between Samples

The distance between the samples taken by a ray can be edited here. The slider allows for values between 0.005 and 0.5. The best visualizations are usually between 0.1 and 0.4. Between these values the individual samples are visible and the application runs smoothly.

There has to be a lower bound that is bigger than 0 because a sample distance of 0 would break the algorithm. We have chosen 0.005 to prevent users from going too low and causing lag in the application. Some systems might already have trouble with a sample distance of 0.005, so a warning is included in the tutorial to carefully slide the value down. The upper bound of 0.5 is less important, We chose this value because it makes the most useful part of the scale, 0.1 to 0.4, fall nicely in the middle of the slider.

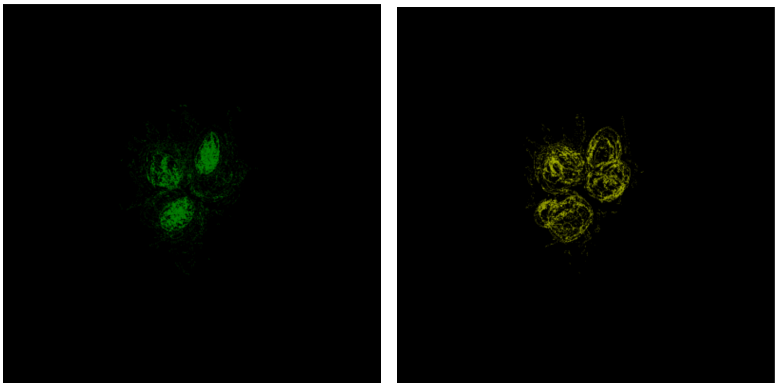


(a) Distance = 0.005 (b) Distance = 0.5

Figure 12: The difference between the maximum and minimum sample distance using the engine voxel grid

3.2.2 Density Matching Value

The density matching value is used for the 'first' compositing method. It is denoted as Y in the formulas in Section 2.3.4. This slider has a minimum value of 0 and a maximum value of 1, which is the same as the range of the sample densities. The effects of changing the density matching value can be see in Figure 13.



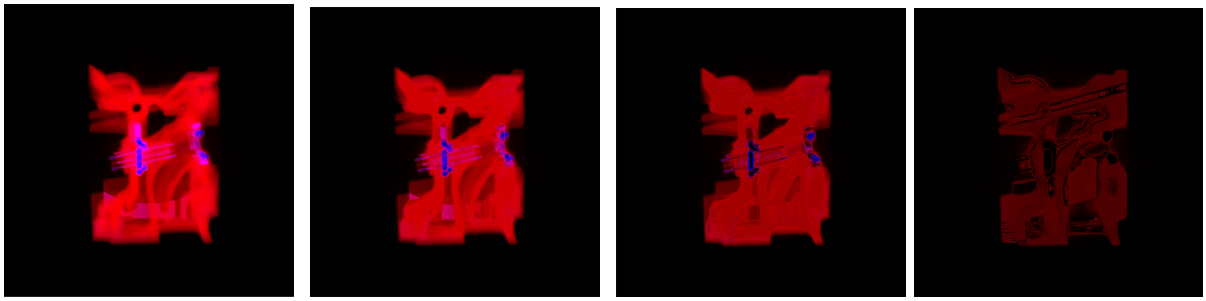
(a) Hazelnut with density matching = 0.35

(b) Hazelnut with density matching = 0.43

Figure 13: The effects of altering the density matching value

3.2.3 Opacity Cutoff Value

The opacity cutoff value is used for the 'accumulate' compositing method. In 'accumulate', compositing stops when the composited opacity exceeds the opacity cutoff value. This slider also has the same range as the sample densities: between 0 and 1.



(a) Opacity Cutoff = 1 (b) Opacity Cutoff = 0.85 (c) Opacity Cutoff = 0.7 (d) Opacity Cutoff = 0.45

Figure 14: The effects of altering the opacity cutoff value

3.2.4 Show Opacity

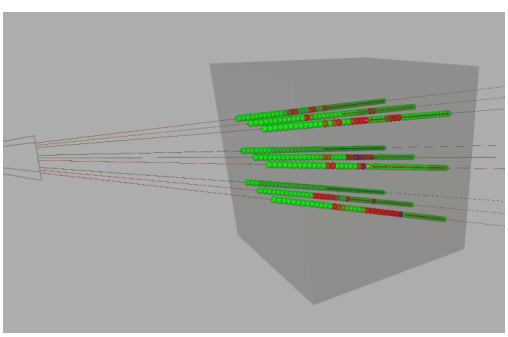
When the show opacity checkbox is checked, the visualized rays and samples will become transparent, see Figure 15. The sample will have the same opacity as its sample colour and the ray will have the same opacity as its composited opacity. This way it is more clear which samples of a ray are contributing to its final colour.

To better show this change we change the recommended transfer function for the engine voxel grid as shown in table 1.

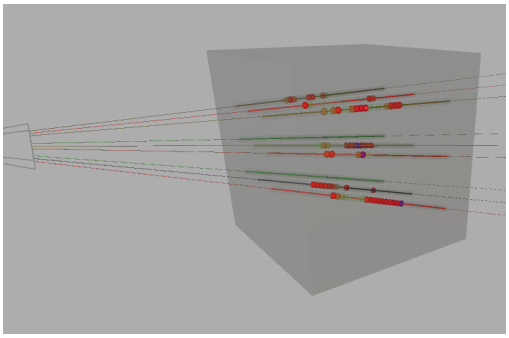
Original		New	
Density	RGBA	Density	RGBA
0	0 0 0 0	0	0 1 0 0
0	0 0 0 0	0	0 1 0 0
0	0 0 0 0	0	0 1 0 0
0.556	1 0 0 0	0.556	1 0 0 0
1	0 0 1 0	1	0 0 1 0

Table 1: Altered version of the recommended transfer function for the engine voxel grid where the "clear" colours are green instead of black

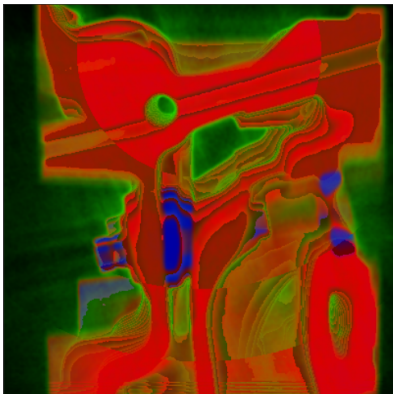
With this altered transfer function the samples with a density close to 0 will show up as green. Despite these densities having an opacity value close to 0, this green colour does not show in the final picture. Note that there is a green hue from low-density values that are not quite 0. But it is not bright green as Figure 15a would suggest.



(a) Show Opacity Off



(b) Show Opacity On

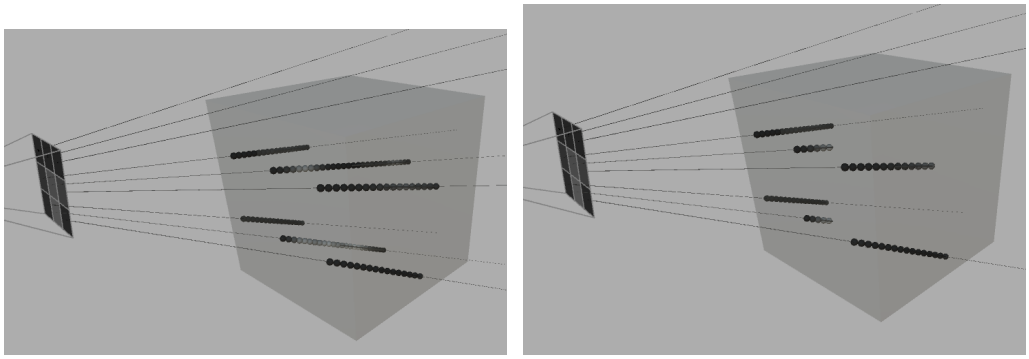


(c) The resulting image

Figure 15: Effects of the Show Opacity checkbox

3.2.5 Early Ray Termination

When early ray termination is set to 'on', the end of compositing in the accumulate and first compositing methods will become visible in the visualized rays. These rays will be cut short at the moment that T becomes true.



(a) Early Ray Termination Off (b) Early Ray Termination On

Figure 16: Effects of Early Ray termination

4 Implementation

In this chapter, we demonstrate how we implemented the changes. First, we go over some relevant features from the Virtual Ray Tracer [8]. Then we highlight new additions we made to support ray casting in the Virtual Ray Tracer.

4.1 Relevant Virtual Ray Tracer Features

In this chapter, we go over two notable features in the Virtual Ray Tracer that we changed or used differently than the original.

4.1.1 Ray Trees

Virtual Ray Tracer uses RayTrees to visualize rays. A RayTree consists of the 0-length base ray and all its subrays. In a standalone application, this structure would not be needed for the Ray Caster, since ray casting rays never reflect or refract. To avoid duplicate code with the Virtual Ray Tracer, the datatype for a ray is kept the same.

We have chosen to use the RayTrees, but not quite as intended. In the Ray Caster a ray is represented as a RayTree with 4 parts: the first part is the same 0-length base ray, the second part is the ray section before entering the voxel grid, the third part is the ray section inside the voxel grid, and the last part is the ray section after the voxel grid. In this way, we can give different properties to the ray sections to help visualize them better without having to change any data types.

4.1.2 Trace Functions

There are two main tracing functions in the ray tracer. `Trace` is used for visualizing rays and it returns a `rayTree`. `rayTrees` can be used to draw a ray and its subrays. `TraceImage` on the other hand is used for generating a high-resolution image. It returns a colour that is to be displayed on the relevant pixel. In these trace functions the main ray tracing algorithms can be found. We of course do not want to perform ray tracing, since our aim is ray casting. Therefore these functions need to be changed. `Trace` is replaced by `CastVisualizableRay`. `TraceImage` keeps the same name, but using inheritance it is overwritten.

The ray casting algorithm is performed in `CastRay`. `CastRay` returns a single `RCRay`, which is a child class of `RTRay` with some additions, such as a list of samples and compositing methods. Both `CastVisualizableRay` and `TraceImage` use `CastRay` to perform ray casting, then transform the returned `RCRay` into the appropriate data type.

4.2 Extended Classes

Because we are re-using a lot of functionality from the ray tracer, but we do not want to modify it, we decided to use inheritance. This way we can change relevant classes without changing the original ray tracing levels. There were 4 classes from the virtual ray tracer that we extended using inheritance.

- `RayManager` → `RayCasterManager`
  In this class the drawing of the rays is handled. Because we needed to draw samples as well as the rays we had to change the drawing methods.

- `RTRay` → `RCRay`
  In RTRay the information about a (sub)ray is stored. A ray casting ray has more information, like samples, that need to be stored. For this we made RCRay.

- `UnityRayTracer` → `UnityRayCaster`
  In this class the ray tracing/casting algorithms are implemented. Since our aim is to perform ray casting instead of ray tracing we overwrite the trace methods. More on this in Section 4.1.2

- `RayTracerProperties` → `RayCasterProperties`
  In `RayTracerProperties` the control panel input and output is handled. Since the Ray Caster has some overlapping controls, but also some new ones, we made a child class. We then added the new controls to the child class `RayCasterProperties`, and we deactivated any controls that we did not want to use in the unity editor.

4.3 New Classes

New classes have also been added to the application. These classes are only related to ray casting and are not tied to the ray tracing levels in any way.

- `VoxelGrid`
  To represent our voxel grid data in Unity we made a prefab. A prefab in Unity is a template with properties and settings that can be reused in multiple scenes. To this prefab, we added a transparent cube object to show in the scene. We also added the class `VoxelGrid` to represent our data.

  The `VoxelGrid` class has a variable `Grid` where the scalars of the voxel grid are stored. Loading this data from the raw files takes quite a lot of time as can be seen in table 2. In the first implementation, the application would freeze anytime the voxel grid type was changed because it was loading in the data. This was improved to only loading when a type was selected for the first time, then caching the calculation. To make the user experience feel smoother, we tried loading all the data at the start of the level, but this caused a long waiting time before the level would load without any information on whether the application was still working. In the end, we implemented a loading bar whenever the user selects a voxel grid type for the first time. This way there is no long black screen, and the user can see what they are waiting on and how long it will take.

Type	File Size	Ryzen 3700x, RTX 2070	Ryzen 5950x, RTX 3080
Bucky	0.032MB	0.30s	0.11
Bunny	92MB	6.99s	1.96
Engine	16MB	4.23s	0.69
Hazelnut	131M	17.08s	3.45

Table 2: Loading times in seconds for different voxel grids with system specifications

- `RayCalculationBreakdown`
  This class handles the Ray Calculation Window mentioned in Section 3.1.4. Its variable breakdown part is static and is simply a text box. The formulas are 4 pre-made images that are saved as sprites. When the compositing method is changed the relevant formula image is displayed.

- `Sample`, `SampleObject`, `SampleObjectPool`, and `SampleRenderer`
  These four classes together handle the samples. `Sample` itself is the data of a sample. `SampleObject` is a sample object in Unity space. `SampleRenderer` is tied to a sample object and draws it. `SampleObjectPool` is used for optimization.

There are many of these samples at a given time and to keep the application running smoothly we use the object pool. We start with an initial number of 128 samples. These samples are loaded, but set to inactive and therefore invisible. If more than 128 samples are needed they are created, but not destroyed, so they can be reused as well. The `SampleObjectPool` handles activating, deactivating and creating new samples.

5 User Study

We evaluated the performance of Virtual Ray Caster by running a user study. Participants were asked to follow all tutorial steps of Virtual Ray Tracer's ray casting level, then continue to experiment with the settings of the program until they had interacted with the product for at least a total time of 10 minutes. Following this, they were asked to fill in a simple questionnaire.

5.1 Personal Questions

Question	Answers	
Did you follow the RUG Scientific Visualization Course?	I am or was a student of the RUG Scientific Visualization course	2
	I am or was a student in a computing science-related field	1
	I have not been a student in computing science or a related field	4

With this question, we aim to divide the users into three groups: Users who are already experienced with ray casting, users who are inexperienced with ray casting and are in our target group (computing science students), and lastly users who are inexperienced with ray casting and also do not have a technical background.

5.2 Educational Questions

Question	Answers					
If you followed the RUG Scientific Visualization course, do you think the Ray Caster application would be helpful to future students of the course?	Yes	3				
	No	0				
Did the application help you understand ray casting better?	Yes	4				
	No, but I already understood ray casting well beforehand	3				
	No	0				
Do you think the application can help other people understand ray casting better?	Yes	7				
	No	0				
Which of the following things do you think were successful in helping you understand ray casting?	The text tutorial that you can follow at the bottom left	2				
	The visualization of the ray casting progress	6				
	Being able to experiment and try things out in the application	5				
	The ray calculation window with the breakdown of a single ray calculation	1				
How would you rate these aspects of the application?		Very good	Good	Neutral	Bad	Very Bad
	Visuals	1	5	1	0	0
	User Interface	3	4	0	0	0
	Tutorial	3	3	1	0	0
	Ray Calculation	3	5	1	0	0

The overall feedback on the educational questions is positive. All previous students of the Scientific Visualization course think that the ray caster will be able to help future students. Additionally, all users found they learned something new about ray casting from the application, except users who were already experienced in ray casting, who believed that the application might be helpful to other people learning about the technique. This is in line with expectations since the application only covers the basics of ray casting.

The visualization of the ray casting process and the ability to control and experiment in the application were most appreciated. Less than half of the users found the tutorial and the ray calculation window useful. This suggests that these aspects can use improvement.

5.3 Technical Questions

<table>
<tr><th>Question</th><th>Answers</th></tr>
<tr><td>How would you rate these aspects of the application?</td><td>
<table>
<tr><th></th><th>Very good</th><th>Good</th><th>Neutral</th><th>Bad</th><th>Very Bad</th></tr>
<tr><td>Ease of Use</td><td>4</td><td>2</td><td>1</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>What did you think of the complexity of the application?</td><td>
Too simple, more settings and controls would be an improvement 1
Good 6
Too complex, there are too many unnecessary settings and controls 0
</td></tr>
</table>

The general consensus here is that the application has a good ease of use. There was no particular part of the application that sprung out as having the best or worst ease of use. Most participants agreed that the application had a good level of complexity, with some users finding it too simple. No one found the application too complex.

6 Conclusion

The goal of this thesis was to add ray casting support to Virtual Ray Tracer. We added support for the data needed to perform ray casting, in the form of voxel grids. We also implemented all three major steps of the ray casting algorithm: sampling, transfer functions, and compositing methods. To visualize this and allow the user to learn how ray casting works we added visualizations for all steps. Additionally, we added extra controls to improve the learning experience of users. These controls are the distance between samples slider, alpha cutoff value slider, show opacity checkbox, and early ray termination checkbox.

From the user study, we can conclude that the application successfully helped in increasing the understanding about ray casting. The visualization and ability to control and experiment in the application were most appreciated. The ray calculation window and tutorial were least helpful, and have room for improvement.

7 Future Work

The basics for ray casting support in Virtual Ray Tracer have been added, but there is much room for improvement. In this chapter, we list some additions or changes that could be valuable additions to this project.

- **Alpha value scaling**
  Allow users to enable and disable opacity value scaling. Currently, the opacity value of the transfer function is a set value that undergoes no further altering. When changing the sample distance this can lead to problems, an example being decreasing the sample distance and using the accumulate compositing method. In this case, the image will become brighter and the early ray termination will activate earlier. Adding opacity value scaling scales the alpha value of the transfer function based on the sample distance. This way users do not have to change the opacity values in their transfer function every time they change the sample distance.

- **Show voxel structure visualization**
  Currently, the voxel grid is shown as a transparent box. This does not help in visualizing the structure of the voxel grid. Allowing a grid of the voxels to be displayed instead of the transparent box would be an improvement. However, at high resolution voxel grids this visualized grid might become too fine grained.

- **Show real time ray casted voxel grid**
  Instead of displaying the voxel grid itself, it might be more useful to show a low-resolution, 3d model of the object inside the voxel grid. This object would have to be transparent to still visualize the samples and rays inside of it.

- **Trilinear interpolation visualization**
  The trilinear interpolation that is performed to get the density of a sample from the voxel grid is currently only explained in the text during the tutorial. A visualization of this step would improve the users understanding of this concept. One could for example show the trilinear interpolation of a sample when it is selected. Or have a very low-resolution voxel grid so the individual voxels are big enough to visualize this step in.

- **Upload custom data**
  Allow users to upload their own voxel grids. This will allow them to view objects that they are interested in instead of the 4 pre-selected grids that we currently have available.

- **Better transfer function control**
  The current transfer function has a set row count of 5. At least making this dynamic would be an improvement. Even better would be to make a more intuitive design. During the Scientific visualization lecture, an example is shown where the transfer function is controlled through a graph. The x-axis of the graph represents the density, the y-axis shows the alpha value. On this graph, a dot can be placed with a certain colour. This colour will then be used for its corresponding density on the x-axis and with an alpha value equal to that of the y-axis.

- **More levels**
  Settings up more levels to go over the individual parts of ray casting with more guidance and examples would be an improvement. One could make a level for every major step: volumetric data, taking samples, transfer functions, and compositing.

8 Acknowledgements

I would like to thank both my supervisors Steffen Frey and Jiří Kosinka for enabling me to complete this thesis, and for their patience, guidance and feedback. I would also like to thank the Scientific Visualization and Computing Science students who participated in the user study. I thank my family and friends for their support, and my dog Juna for never leaving me and my computer's side.

References

[1] Willard A. Verschoore de la Houssaije, Chris S. van Wezel, Steffen Frey, and Jiri Kosinka. Virtual Ray Tracer. In Jean-Jacques Bourdin and Eric Paquette, editors, *Eurographics 2022 - Education Papers*. The Eurographics Association, 2022.

[2] W.A Verschoore de la Houssaije. A virtual ray tracer. 2021, BSc thesis, University of Groningen.

[3] Chris van Wezel. A virtual ray tracer. 2022, BSc thesis, University of Groningen.

[4] Jesper van der Zwaag. Virtual ray tracer: Distribution ray tracing. 2022, BSc thesis, University of Groningen.

[5] Bora Yilmaz. Acceleration data structures for virtual ray tracer. 2022, BSc thesis, University of Groningen.

[6] Peter Jan Blok. Gamification of virtual ray tracer. 2022, BSc thesis, University of Groningen.

[7] Roan Rosema. Adapting virtual ray tracer to a web and mobile application. 2022, BSc thesis, University of Groningen.

[8] Chris S. van Wezel, Willard A. Verschoore de la Houssaije, Steffen Frey, and Jiří Kosinka. Virtual ray tracer 2.0. *Computers & Graphics*, 111:89–102, 2023.

[9] Wikipedia contributors. Ray tracing (graphics) — Wikipedia, the free encyclopedia, 2023. [Online; accessed 5-November-2023].

[10] G. Simons, S. Herholz, V. Petitjean, T. Rapp, M. Ament, H. Lensch, C. Dachsbacher, M. Eisemann, and E. Eisemann. Applying visual analytics to physically based rendering. *Computer Graphics Forum*, 38(1):197–208, 2019.

[11] Gerard Simons, Marco Ament, Sebastian Herholz, Carsten Dachsbacher, Martin Eisemann, and Elmar Eisemann. An interactive information visualization approach to physically-based rendering. In *VMV*, 2016.

[12] B. Spencer, M. W. Jones, and I. S. Lim. A visualization tool used to develop new photon mapping techniques. *Computer Graphics Forum*, 34(1):127–140, 2015.

[13] Tobias Zirr, Marco Ament, and Carsten Dachsbacher. Visualization of coherent structures of light transport. *Computer Graphics Forum*, 34(3):491–500, 2015.

[14] Miłosław Smyk, Magdalena Szaber, and Radosław Mantiuk. *JaTrac — an exercise in designing educational raytracer*, pages 303–311. Springer US, Boston, MA, 2002.

[15] Nick Vitsas, Anastasios Gkaravelis, Andreas-Alexandros Vasilakis, Konstantinos Vardis, and Georgios Papaioannou. Rayground: An online educational tool for ray tracing. In *Eurographics (Education Papers)*, pages 1–8, 2020.

[16] Jake A Russell. An interactive web-based ray tracing visualization tool. *Undergraduate Honors Program Senior Thesis. Department of Computer Science, University of Washington*, 2, 1999.

[17] Christiaan Gribble, Jeremy Fisher, Daniel Eby, Ed Quigley, and Gideon Ludwig. Ray tracing visualization toolkit. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, page 71–78, New York, NY, USA, 2012. Association for Computing Machinery.

[18] Ashikhmin M. Marschner S. Shirley, P. *Fundamentals of Computer Graphics (3rd ed.)*. CRC Press, 2009.

[19] S. Frey. Scientific visualization lectures: Ray tracing diagram, University of Groningen.