university of groningen

faculty of science and engineering

A Virtual Ray Tracer

Bachelor's Project Computing Science

December 24, 2021

Student: Chris van Wezel

Primary supervisor: prof. dr. J. Kosinka

Secondary supervisor: dr. S.D. Frey

Abstract

Ray tracing is an integral and mathematically complex part of the field of Computer Graphics. With recent advances towards real-time ray tracing students now more than ever need help understanding the concept.

With this thesis I set out to discover what an effective way would be to visualise ray tracing within an application and how much the application would help people understand ray tracing better. To answer these question I together with a project partner designed an application that could visualise the inner workings of ray tracing and a corresponding user study to test it. Despite some setbacks while writing the application the user study turned out to be overwhelmingly positive. According to the data the application was adequate in visualising ray tracing and could aid people with learning about ray tracing. I highly recommend doing more research and improve upon this application.

Contents

1 Introduction

Since the first digital image, there has been a push towards developing realistic graphics with computers. Over the years better, faster and compacter ways have been discovered to generate nicer and realistic-looking images. However, some processes proved to be more challenging than others. Ray tracing was one of them. The first mention of the process of ray tracing, we are aware of, is in a paper of Arthur Appel [1] about computer-generated shadows. In his paper Appel described a program that could generate shadows by casting rays in a scene from a light source to an object. He does not call it ray tracing yet, but this would be the start of a world wide revolution in drawing realistic graphics.

Because of its accurate simulation of the real world and the ongoing strife to make digital images as realistic as possible, ray tracing has become an integral part of the field of Computer Graphics. With new research into real-time ray tracing the importance of accurately teaching students the concepts of ray tracing is only growing. However, this turned out to be somewhat difficult. The different implementations of ray tracing are mathematically quite complex and can therefore be difficult to teach and understand.

To address this issue, W.A. Verschoore de la Houssaije and I designed an application that could aid teachers and guide students by visualising the processes behind ray tracing. In our application the user can manipulate the environment and see what the effects are on the ray tracing process. To not overwhelm the user the application starts with simple ray tracing concepts and slowly adds complexity.

To test the effectiveness of our application we reformulated the issue into three questions and conducted a user study bases on them. The three questions are:

1. What is the best way to present ray tracing in a virtual environment?
2. What interactions should be supported and how?
3. How does such a tool influence the ray tracing learning process?

Although we worked together on the application and the user study we both wrote our own thesis and divided the research questions accordingly:

- W.A. Verschoore de la Houssaije in his thesis [4], focused on: What interactions should be supported and how? How does such a tool influence the ray tracing learning process specifically for CS (Computing science) students that followed the Computer Graphics course at the RUG (University of Groningen)?
- I, in this thesis, focus on: What is the best way to present ray tracing in a virtual environment? How does such a tool influence the ray tracing learning process specifically for CS students who have not followed the Computer Graphics course at RUG, non-CS students and people outside the university?

I included the complete division of work in Table 5 at the end of this paper.

In Chapter 2 I start with what ray tracing is, its current implementation and available ray tracing applications in the field of education, in Chapter 3 I discuss the requirements of our application, Chapter 4 explains the tools we used, the ray tracer, the application itself and how we divided the work, In Chapter 5 I go over the survey and its results, In Chapter 6 I discuss issues we encountered and flaws of the application, In Chapter 7 I draw a conclusion from the results we found, Finally in Chapter 8 I propose some ideas for future research.

2 Background Information

This Chapter explains the current ray tracing implementations and development, and shows how similar research turned out.

2.1 Ray tracing

To explain what ray tracing is we take a look at how light behaves in the real world. In the real world a light particle is cast from a light source with a given colour. For example the sun. Whenever this particle hits an object it can change direction or colour depending on the characteristics. For example if the ray hits a mirror it will only change directions or if it hits a green leave it will lose other colours and turn green. After a couple of these interaction the ray will eventually hit our eye. In our eye the information in the particle will be decoded, together with many other particles that arrived at the same time, to form an image. This image has colour where we catch a coloured ray, shadows where we did not catch that many rays and reflections where the rays solely changed direction.

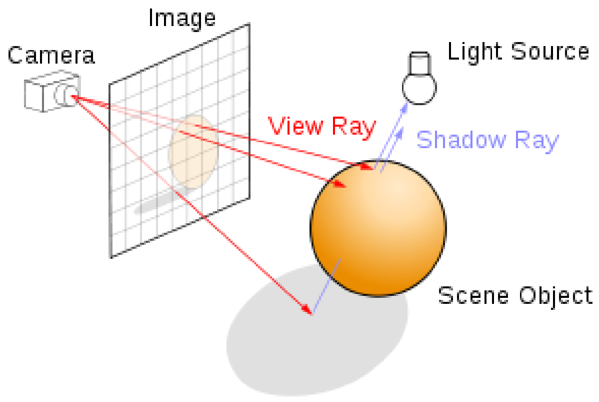


Figure 1: Example of Ray Tracing. https://commons.wikimedia.org/wiki/File:Ray_trace_diagram.svg

For ray tracing in computing science the principles are the same but there are also some differences [7]. In computing science the process is usually reversed. The light rays are cast from the camera (our eye) instead of the light source. This way you do not waste performance on calculating light rays that would have never reach the camera from the light source. To get a realistic-looking image we then divide the view of the camera in a grid. After this we cast a light ray through each pixel on this grid. When the rays hit an object new rays are cast based on the properties of the object. This process is repeated until the light rays either reach a given distance or a light source. When done we gather the final colour of the ray and put a dot on the screen, with that colour, at the corresponding pixel. A representation of this process can be seen in Figure 1. To get a sharp and accurate image we need to cast a lot of rays and allow for many interactions with objects. Add to this the different calculations we need to make for different objects and situations like; round objects, cylinders, planar quads, reflection and refraction [7]. We can conclude that ray tracing is inherently a computational costly and without dedicated hardware a time consuming process. However, this is not unexpected.

2.2 Current implementations

In general creating a pleasing image on a computer screen, (computer graphics), is a computationally costly and time consuming process without dedicated hardware. To amend this companies like Nvidia and AMD started to create just that. They developed processors that are specifically designed to solve the computations involved with computer graphics. These processors were fittingly named Graphic Processing Units (GPUs). GPUs work by focusing on specific aspects of computer graphics. Although a lot of computations are needed to generate an image, most of these computations are of the same type and do not influence each other. This means that the process could be parallelized and accelerated with many processors that excel in these specific calculation. A representation of a normal processor (CPU) vs a GPU can be seen in Figure 2. Where a CPU contains a couple of multipurpose processors a GPU contains a lot of dedicated processors that excel in a specific set of tasks. With the initial launch of GPUs a loop of continuous demand by the users of computer graphics and the development of new, faster and compacter GPUs was created.

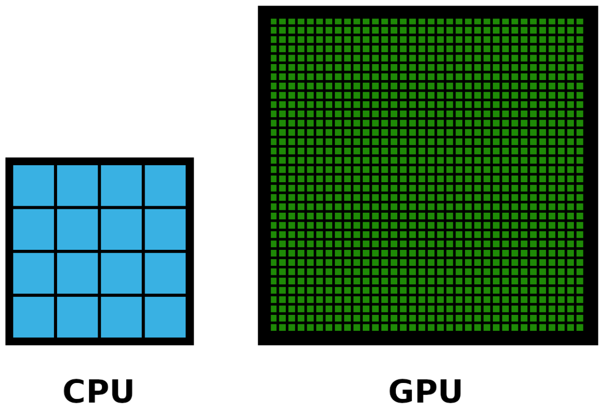
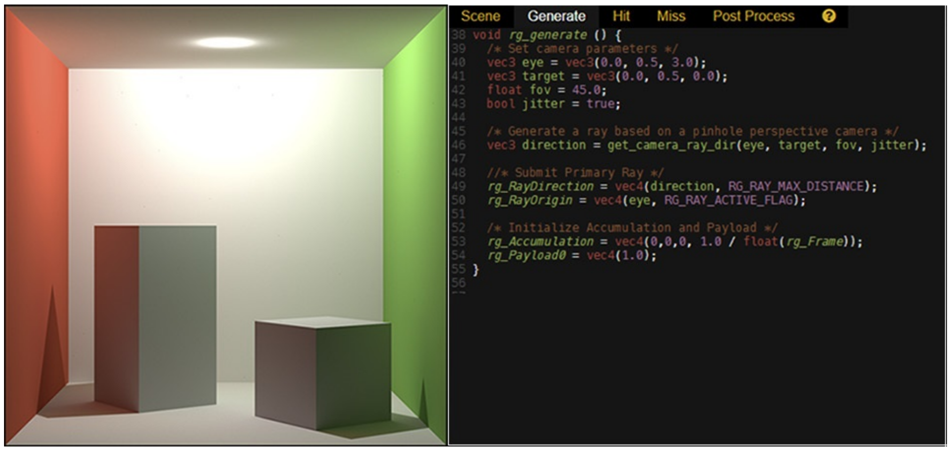


Figure 2: A representation of the amount of processors in a CPU vs GPU. Each square represent a processor.

Until the users started to demand real time ray tracing. Although ray tracing was already widely used by the computer graphics user base, for example computer generated images (CGI) in the movie industry. Real time ray tracing revealed itself to be even more computational demanding than other types of computer graphics. It took Nvidia until 2018 to announce the first graphics card that was able to preform real time ray tracing [3]. They achieved this by creating dedicated ray-tracing processing units and improving the ray-tracing calculations [2]. These processing units were called RT cores and were released with their new graphic cards series named RTX. Special ray tracing software features in DirectX called DXR enabled ray tracing for Windows.[6]. With the new real time implementation of ray tracing and the supporting tools, DXR, ray tracing could be used by the computer graphics user base and new developments soon followed. Notably in the game industry. Until now ray tracing was out of reach for the game industry because its slow implementation inherently clashed with the nature of gaming. Real time ray tracing meant that the ray tracer finally was fast enough to keep up with the render demand of the games. Developers now had a new tool to make there games even more beautiful.

2.3 Education



```
void rg_generate () {
  /* Set camera parameters */
  vec3 eye = vec3(0.0, 0.5, 3.0);
  vec3 target = vec3(0.0, 0.5, 0.0);
  float fov = 45.0;
  bool jitter = true;

  /* Generate a ray based on a pinhole perspective camera */
  vec3 direction = get_camera_ray_dir(eye, target, fov, jitter);

  //* Submit Primary Ray */
  rg_RayDirection = vec4(direction, RG_RAY_MAX_DISTANCE);
  rg_RayOrigin = vec4(eye, RG_RAY_ACTIVE_FLAG);

  /* Initialize Accumulation and Payload */
  rg_Accumulation = vec4(0,0,0, 1.0 / float(rg_Frame));
  rg_Payload0 = vec4(1.0);
}
```

Figure 3: Rayground interface. `https://rayground.com/documentation`

However, this means that the already strong position of ray tracing within the field of Computer Graphics is becoming stronger. In turn this increases the importance of students learning about ray tracing in the Computer Graphics course. Since ray-tracing, as mentioned before, is a mathematically complex process, a visual representation could aid students in learning about it. Rayground [10] found that from the students who learned ray tracing with their visual applet, 80% had the feeling it had a positive effect on their understanding of ray tracing. 73% was inspired to experiment further with ray tracing. This bodes well for the application we have developed. However, the study group only consisted of 20 students of which only 15 participated in the evaluation. Another important note is that the tool created by Rayground, as shown in Figure 3, also simulates a ray-tracing pipeline for students to practice coding [10]. In our application we only focus on the visualisation of ray tracing, not the coding aspect.

Besides Rayground there exists a tool called FUSEE [8]. FUSEE is an open-source 3D platform build by students. It is meant to fill the niche between the larger game engines and building one yourself. Its main goal is to allow students to learn about 3D programming, which includes ray tracing. The platform on its own does not help students understand ray tracing, but the existence of the application shows that there is a need for such type of application. We briefly discussed if we could use this tool to build our application on. However, a down side of the tool is that it is fairly new and is lacking in features and documentation in comparison with older well-established game engines.

3 Requirements

To establish a base for our application we determined a set of requirements. We designed the application to help students learn about ray tracing. The application can only do this if it itself knows how ray tracing works. Meaning that the first requirement is that the application should be able to perform ray tracing.

The second requirement is as important as the first. To show the user what the ray tracer has calculated we need a way to visualise the rays. Besides the rays it would benefit the user if the user is also able to see the objects, light sources and the camera in the scene.

Because, seeing the scene from only one side is not that impressive, the next requirement is that the user is able to move through the scene. To give the user even more freedom to experiment the user should be able to manipulate the scene in the game. Meaning that they can move or change objects, modify the behaviour of the ray tracer, move or change light sources, and change settings or the orientation of the camera.

The following requirement for the application is a progression system. The application should simply not overwhelm the user by giving him access to all the ray tracing concepts at once. Instead, the complexity should be slowly increase with a level system.

Not everything can be explained by showing it to the user. The application is also required to have an information system. This system can explain the user how the application works and what they see in the current level.

Lastly the application will be used for the computer graphics course at the RUG. There is a lot that can be improved or expanded up on. Therefor the final requirement of the application is that it is easily expandable.

Taking only the main parts we end up with the following list of requirements:

1. Have an internal ray tracer that outputs a list of rays depending on the objects in the scene.

2. Have a visual interface which can show the user the rays, objects, light sources and the camera.

3. Give the user a set of tools with which the user is able to move through and manipulate the environment.

4. Have a progression system. Start with 1 object and a light source and slowly introduce the user to more complex ray tracing concepts.

5. Have a set of instructions that explains the user how to use the application and what the user currently sees on the screen.

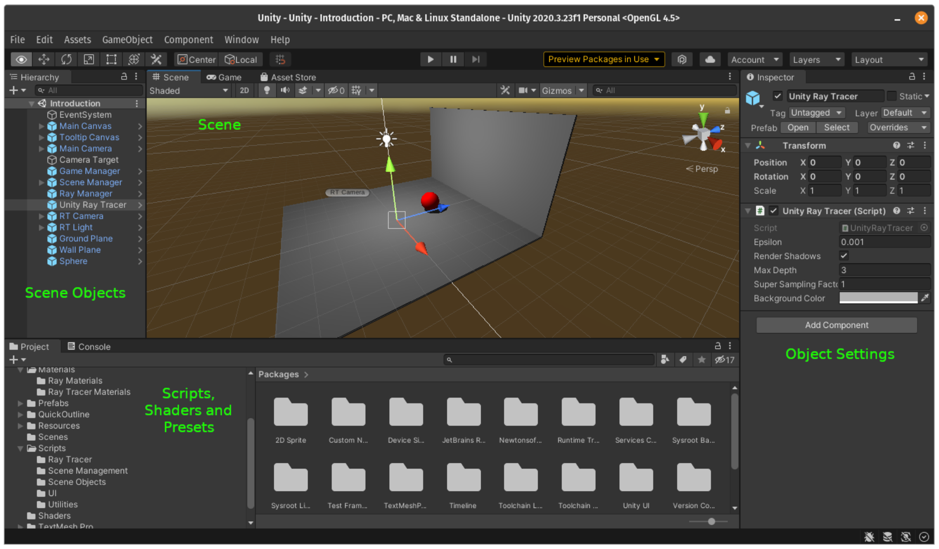
6. Easily expandable.

4 Implementation

In this Chapter I explain the tools we used to build the application, how we implemented the ray tracer and the build process of the application itself.

4.1 Tools and Languages

Our first tool is a game engine. A game engine is a software framework design to develop games. It usually contains tools, libraries and an environment in which it is easy to create a virtual world and add attributes to the objects in it. An example of such an environment can be seen in figure 4. Games are applications that are heavily based around visuals. They usually allow the user to walk around and interact with a virtual generated world. Since our application is in principle a game, we have a virtual world with which the user is allowed to interact, a game engine is a perfect fit to use as framework for our application. The game engine would handle most of the graphical aspects allowing us to spend more time coding the actual ray tracer and ray interactions.



Unity Ray Tracer (Script)

Script	UnityRayTracer
Epsilon	0.001
Render Shadows	✓
Max Depth	3
Super Sampling Factc	1
Background Color	

Figure 4: Example of the tool provided by a game engine with annotations. In this case Unity.

For this project we decided to choose between the two better known game engines Unreal and Unity. Both game engines are well documented and widely used [9][5] making it easier to search for solutions, when issues are encountered, and for other students to expand on the application in followup research. We decided to choose Unity for the following three reasons. The first reason being that the main supported programming language of Unity is C#. C# is a programming language in which you do not have to assign and free your own memory, like in C++ used for the Unreal engine. Although this generally makes the programming language slower it is more user friendly and makes it easier to use. The second reason was that one of our developers already had some experience with Unity. The last reason was that the other game engine on our list, Unreal, was simply to big to run on my current hardware.

For programming languages, we used C# and some sort of Unity shader language. Initially we also used C++ but, we dropped it which I will discuss in Section 4.3. To improve expandability we added comments to most of the code. To share the code and allow for simultaneous development we created a repository on GitHub. Finally for communication, sharing files and images between us and the supervisors we used Google Drive.

4.2 Initial Sketch

Creating the application started out with a meeting to discuss what the application should look like and what type of interaction should be supported. During the meeting we came up with the list mentioned in Chapter 3 and some suggestions of our own. Figure 5 shows a sketch of the initially proposed application.

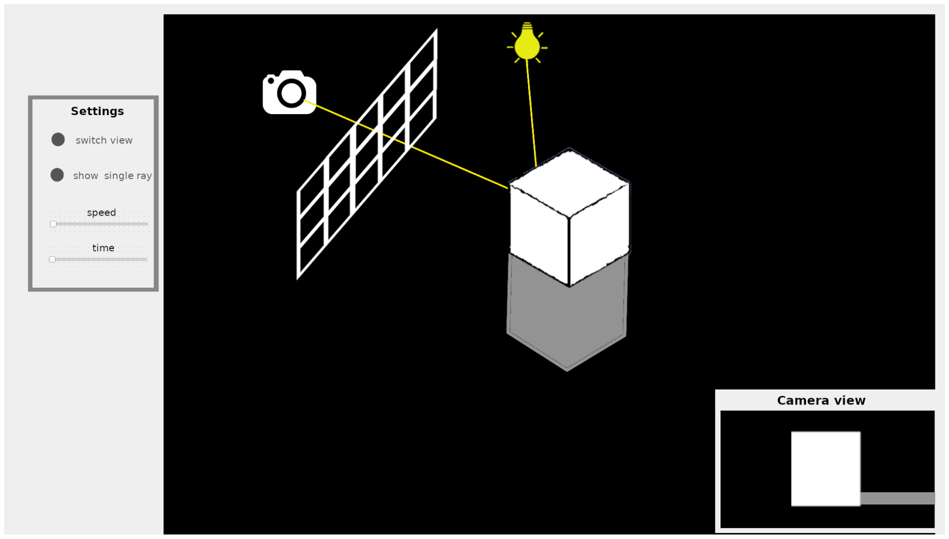


Figure 5: Sketch of our application.

The black area in the middle depicts the scene in which we show how ray tracing works. The scene contains a virtual camera, a virtual screen, a ray, an object and a light source. In the left bottom corner a small preview is shown of what the scene should look like if rays where shot from the perspective of the virtual camera through each square, pixel, of the virtual screen. The user is allowed to move freely through this area and click on objects to change their settings or move them around. When an object is selected by the user the settings are shown on the left side of the screen. Whenever the user makes a change to the scene the application will recalculate and update the rays and what is shown on the virtual screen and preview. With a clear view of what the application should look like development could start. First thing on the list, the ray tracer.

4.3 Ray tracer

The ray tracer is written in C# and is designed around an internal Unity ray cast function. The ray tracer currently supports reflective and refractive interactions and it can process a scene with point light sources and objects. The objects are defined by a material and a body of triangles. The materials have a range of properties:

- Colour: Red, Green, Blue
- Light factor: Ambient, Diffuse, Specular, Shininess
- Transparant: Breaking index

During personal testing the ray tracer preformed well and was able to update the rays on a real time basis. We decided to go for C# because it integrates nicely with Unity and issues we encountered while integrating a ray tracer written in C++.

The C++ ray tracer was initial meant to speed up the writing process by using an already available ray tracer from the Computer Graphics course at the RUG. However, integrating the C++ ray tracer in Unity proved to be difficult and time consuming. Most issues revolved around the communication of data between the two languages. For example, our first image, Figure 6, was generated by manually instructing the ray tracer what the scene looked like. After a lot of development time we only managed to implement the communication between the ray tracer and the application, not the other way around.

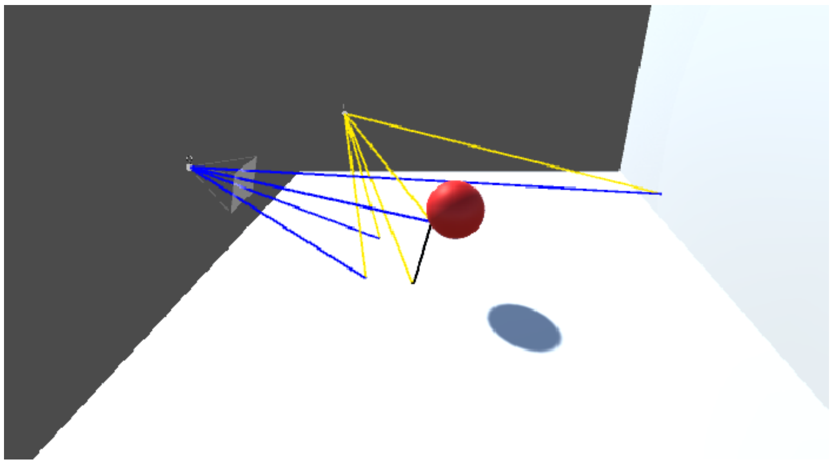


Figure 6: The first ray traced image generated by the application.

More issues came to light at the first test of the application with the ray tracer on Windows. The ray tracers performance looked promising on Linux but turned out to be sub-optimal on Windows. Only a couple of rays where enough to create a noticeable delay in the rendering of the scene. To save the project, we decided to abandon the C++ ray tracer and instead build one in C# from scratch. We where able to use the knowledge gained from the C++ ray tracer to speed up the build process of the C# ray tracer.

4.4 Application

With the ray tracer taking a lot of development time our work was split. One of us was working on the ray tracer and the other on the rest of the application. After the switch to the C# ray tracer we could finally both invest our time in developing the rest of the application. A division of work is mentioned in Table 5. Included below a list showing how we implemented each requirement and a brief description of the most prominent features for each:

1. **Have an internal ray tracer that outputs a list of rays depending on the objects in the scene:**

   The ray tracer is covered by Section 4.3. Notable features for this requirement are:

   - The ray tracer is able to handle transparent objects. Shown in Figure 7. Rays cast through a transparent object are influenced by its breaking index which can be set by the user. This feature took a while to implement. We had to create two separate shaders, transparent and opaque, for Unity to properly render transparent objects.
   - The ray tracer only updates when a user changes something in the scene. By only recalculating the rays on a user made change we minimise the burden on the processor. This allows the application to preform better overall and on older hardware.

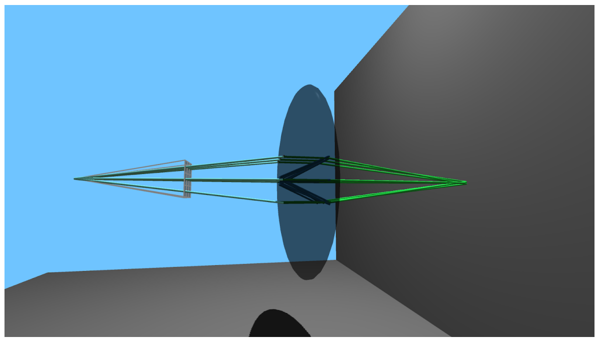


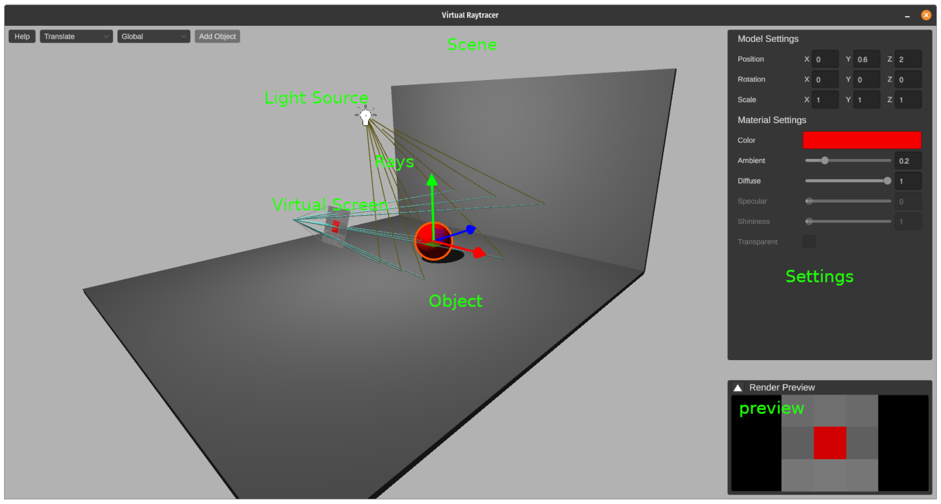
Figure 7: Ray tracing through glass with the c# ray tracer.

2. **Have a visual interface which can show the user the rays, objects, light sources and the camera:**

   The final application can be seen in Figure 8. Besides the settings window being moved from the left to the right, it looks similar to the initial sketch. All the objects and their functions seen in this image are created with the tools provided by the Unity game engine, Section 4.1. Notable features for this requirement are:

   - The scene shown to the user looks relatively similar to the scene that the ray tracer would generate. This makes it easier for the user to relate the actual scene with the ray traced image. Do note that this system is not perfect and in some cases the scene differs from the image.
   - Almost all object properties can be changed.
   - Every change the user makes is immediately shown in the scene.

- The user is able to go inside the objects. Ray tracing can also happen inside of an object. Visualising the inside of the objects gives the user more freedom to experiment and creates a more accurate visualisation of the ray tracing process.



Model Settings

Position	X	0	Y	0.6	Z	2
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Material Settings

Color	🟥
Ambient	0.2
Diffuse	1
Specular	0
Shininess	1
Transparent	☐

Figure 8: Final application.

3. **Give the user a set of tools with which the user is able to move through and manipulate the environment:**

   To allow the user to interact with the scenes from our application we added the following features to it:

   - The user has access to a button to create a high resolution ray traced image from the view of the virtual camera. The user also has access to settings to increase or decrease the resolution of this image.
   - The user is able to select a specific pixel in the preview in the bottom right. When this pixel is selected only the rays that define that pixel are shown.
   - The user can change how the rays are rendered. The application can instantly draw the rays, they can be drawn over time or they can be drawn one by one.
   - The user is able to interact freely with the scene by moving around within it, move objects or change a range of properties.

4. **Have a progression system. Start with 1 object and a light source and slowly introduce the user to more complex ray tracing concepts:**

   The application contains a basic progression system. Within the application the user has access to a menu in which the user can select different levels. Each level shows different ray tracing concepts and all levels start with a pop-up explaining the user which ray tracing concepts the user currently sees within the scene. A notable feature of this progression is:

   - The final level is a sandbox level. In this level the user is allowed to create his own scenes by adding or removing objects or light sources.

5. **Have a set of instructions that explains the user how to use the application and what the user currently sees on the screen:**

   Each level in the application starts with a text window showing the user information about the current level and how some of the tools work. The same information is also stored in a help menu in the left upper corner of the screen. The user can access this menu whenever they want.

6. **Easily expandable:** As mentioned in Section 4.1 Unity is a well documented game engine. Documentation makes it easier for new developers to understand how the application works. The Unity development tools also work on all three major operating systems, Windows, Linux and macOS, allowing the new developers to choose their own preferred platform. Furthermore, we added comments to most of the code explaining what each part of the code is supposed to do.

5 User Study

Here I explain the user study and its results. Section 5.1 is about the user study and its questions, in Section 5.2 we show the results. The questions used in the user study are added to the end of the document.

5.1 Questions

Starting of with the restrictions on the user study. From the start of the project we had a clear view of the target population of our user study. We wanted to know the influence of our application on the learning process of students who attend the Computer Graphics course at the RUG and on every one else. Further restrictions are imposed by the nature of our project. Our project needs a computer to be tested therefor the participants should have access to a computer and general knowledge of how a computer works.

For recruitment we relied on our connections with the supervisor of the project, friends and family. Via our supervisor we got access to Computer Graphics students and via friends and family we would recruit every on else. We hoped to recruit a group of 50 participants, but in an attempt to prepare for a lower amount we decided to share the user study between the two theses. By sharing we could, to some degree, both use the same data effectively increasing the efficiency of the user data. However, this imposes an extra restriction on the user study.

As mentioned in the introduction one of our research questions is split in two. One part focuses on students who attended the Computer Graphics course at the RUG and the other on every one else. Due to the nature of these two groups, the participants are either in one or the other, it is impossible to share this data to answer both of these questions. To navigate around this issue and equalise the data set for both research questions, it would be preferred that 50% of the participants are students who attended the Computer Graphics course at the RUG and the 50% are every one else.

With the participants and the restrictions handled we focused on the user study itself. The user study consisted of a practice run with our application and then a set of questions about the application. The questions are divided into 3 categories: personal, educational and technical. In the following subsections we go over each of these question and their goal. The initial intention was to conduct two user studies. First a small user study about the knowledge of ray tracing. Then, a practice run with our application. Finally, another user study about the influence of our application on their understanding of ray tracing and about how user-friendly our application is. With this setup we hoped to see a clear distinction between the knowledge before the practice run and after. However, due to time constrains we decided to drop the first user study and instead merged its questions with the second user study.

A final important note is that for our user study we required the ethical approval of the participants. Before taking the user study we asked the participant if they agree with us using their answers as data in our study.

5.1.1 Personal Questions

With the questions in the personal category we set out to achieve two goals. The first goal is to determine the relation of participants towards computing science, Question 1. An existing relation to computing science could influence the results of our user study. By knowing this variable we can rule out potential inconsistencies in the result. This data could also be used in a later more precise study.

The second goal was to determine if the participants attended the Computer Graphics course at the RUG or not, Question 2. We needed this division so we could both answer our own version of the 3rd research question. Again we do not need to know if a student failed or passed the course, however this could have an effect on their opinion of the application and could be useful in a later study.

5.1.2 Educational Questions

The goals for the questions in the educational category are to determine if the application is helpful and what parts of the application are helpful.

A yes-or-no question would have sufficed for the first goal however, in an effort to make the results a bit more in-depth we decided to divide this goal over multiple questions, specifically Questions 3, 4 and 5. With the current set of questions we gain insight in the perceived helpfulness for Computer Graphics students, potential new users and even the degree of helpfulness for the participants themselves. We are also able to mark the people that already know how ray tracing works, which is important because this could influence their responses in our user study.

The other goal became a question on its own, Question 6. We decided that three parts of our application could potentially be helpful and therefore should be evaluated with this question:

- *The Information texts:* A text block that is shown at the start of every level. The text block contains some details about the level and an explanation of the ray tracing processes shown in the current level.
- *Visualisation of ray tracing:* Ray object interactions. The way we made a distinction between different types of objects or different types of rays. How the rays are drawn.
- *Ability to experiment with various settings:* The multitude of settings available to the user to change the scene. Changing these setting directly influences the ray tracing process allowing the user to discover what the ray tracer would do under certain circumstances.

Furthermore, since it would be possible that multiple parts where helpful we allowed the participants to select multiple responses for this question.

The final question in this section, Question 7, is not directly connected to a goal but allows the participants to give feedback specifically on the helpfulness of our application. Putting this question here makes it easier to find feedback that is specifically aimed towards this section.

5.1.3 Technical Questions

The technical section contains questions focused on the complexity, usability and looks of the application.

The first question is aimed towards usability of the application, Question 8. With this question we could determine if the interface, buttons and instructions where clear enough for the users to understand and use the application.

The second question focuses on the looks of the application, Question 9. We chose four different categories for the user to chose from and allowed them to grade them from very bad to very good:

- *The visuals:* Everything the user can see on the screen and how it is represented.
- *The user interface:* The buttons, sliders, popups and menus.
- *The controls:* The way the user is allowed to move through the levels and interact with the user interface.
- *The scene and explanation of ray tracing:* Specifically how the light sources, camera, objects and ray tracing is represented on the screen.

We believed these categories covered all aspects of the application and focused enough on specific aspects to determine their influence on the looks of our application.

The last question is to determine the complexity of the application, Question 10. Although the application could be easy to use or look nice it could have to many settings or sliders for the user to use. By adding this question we get a good impression of the potential complexity of our application.

Again we added a question to this section for participants to give feedback specifically on the functionality of our application, Question 11.

5.1.4 Feedback

The final question, Question 12, was added to allow the participants to give general remarks or feedback on the application. The feedback from this question could later be used to improve the program. The positive remarks could directly be used to boost the confidence of the creators of the application.

5.2 Results

The user study had a group of 17 participants. The group of participants consists of a mix of CS students, RUG staff, and friends and family. I mainly focus on the result that apply to this thesis. Furthermore, I split the results in three subsections based on their focus. The first subsection of results focuses on the user group, the second subsection on the performance of the application and the final subsection on the usability of the application.

5.2.1 The study group

In Table 1 I combined the data gained from Question 1 and 2. In the rows the table is split between the fields the student are related to and in the columns it tells if they attended the Computer Graphics course and passed it.

Did you follow the RUG Computer Graphics course and if so did you pass?			
	No	passed	failed
I am or was a computing science student.	3	5	1
I am or was a student in a computing science related field.	0	2	0
I have not been a student in computing science or a related field.	6	0	0
Total	9	7	1

Table 1: Results from Question 1 and 2 of the user study.

When selecting the participants for the study group we aimed to have an even split between the participants that did the computer graphics course at the RUG and those who did not. From the data shown in Table 1 we can see that we succeeded in doing so. 53% participants did not follow the course and 47% did, although 1 participant failed. Furthermore, from the participants that did not attend the computer graphics course 67% had no relation to computing science and 33% where or had been a computing science student.

5.2.2 Helpfulness

Here I constructed a table, Table 2, from all the questions of the helpfulness section, questions 3 to 6. The divide in results, the last two columns, is created based on the answer of the participants on Question 2.

Questions	Allowed responses	Results	
		Yes	No
If you followed the course, do you think the application would be helpful to future students of the course?	Yes.	8	1
	No.	0	0
	Empty.	0	8
Did the application help you understand ray tracing better?	Yes.	2	8
	No.	0	1
	No, but I already understood ray tracing well beforehand.	6	0
Do you think the application can help other people understand ray tracing better?	Yes, I think the application can be moderately helpful.	4	2
	Yes, I think the application can be very helpful.	4	7
	No, I don't think the application can be helpful.	0	0
Which of the following things do you think where successful in helping you understand ray tracing?	The ability to experiment with various settings in the scenes.	5	6
	The informative text at the start of scenes.	2	6
	The visualisation of ray tracing in the scenes.	8	7

Table 2: Results from the questions in the educational section, Question 3 to 6, divided in the group that failed or passed the RUG Computer Graphics course **Yes** and those that did not attend the course **No**.

My main focus is on the people that did not attend the course, the **No** column. The first question can be skipped, because it was only meant for the people that did attend the course. For the second question 90% of the **No** group had the feeling that the application helped them understand ray tracing better. This is a good start. Looking further at the third question we see similar results. 100% of the **No** group believed the application to be helpful for other people to better understand ray tracing. Of this group 78% thought the application would be very helpful. For the final question generally the participant found at least two aspects of our application helpful. For this question we mainly hoped that the visualisation of the ray tracer would have been the most helpful. However, the scores are pretty even. Both the ability to experiment and the informative text scored 31.5% and the visualisation of the ray tracing scored 37%.

5.2.3 Usability

Table 3 shows the results of Question 7 and 9.

Questions	Allowed responses	Results
Did you find the application easy to use?	Yes, but some things were confusing or difficult.	8
	Yes, the application was very easy and intuitive.	7
	No, but it was not very confusing or difficult either.	1
	No, the application was very confusing and/or difficult to use	0
What do you think of the complexity of the application?	The application is too simple. More settings and controls would be an improvement.	0
	The complexity of the application is good.	16
	The application is too complex. There are too many unnecessary settings and controls.	1

Table 3: Results from the questions in the technical section, Question 7 and 9.

These questions tell us something about the looks of our application. From the first question we can see that 88% perceived the application easy to use. This is a great number, except 53% of these participants still noted that some aspects of the application where either confusing or difficult. Although 6% found the application not easy to use no one found it very confusing or difficult. Note worthy here is that one of our participants got lost and did not answer this question. The second question is focused on the complexity of our application. Again the result are extremely positive. Of our participants 94% said that complexity of the application was good, not to complex and not to simple. The remaining 6% considered the application to be to complex with to many unnecessary settings and controls.

The results of Question 8, are shown in Table 4. The table shows how our participants rated 4 aspects of our application from **Very Good** to **Very Bad**. In the first column the ratings are shown, in the remaining columns each aspect is shown with the score for each rating below it.

What do you think of these apsects of the application?

	The visuals	The user interface	The controls	The scenes and explanation of the ray tracing concept
Very Good	7	1	3	3
Good	7	9	10	11
Neutral	3	7	3	3
Bad	0	0	1	0
Very Bad	0	0	0	0

Table 4: Results from Question 8. Rating of each aspect of the application.

The most notable observation is the positive trend of the results. According to almost all the participants the aspects of our application are at least **Neutral** or better. There is only one **Bad** review. Looking at the rating for each individual aspect we can see that the visuals scored the best. Of our participants 41% reported the visuals to be **Very Good** and again 41% gave a rating of **Good**. In comparison only 6% of the participants gave the user interface a rating of **Very Good** and only 18% for both the controls and the explanation of the ray tracing concept. The worst score is between the user interface and the controls. Where the user interface had an overall lower score 52% **Good** and 41% **Neutral** the controls had 6% voting **Bad**. The scenes and explanation of the ray tracing concept averaged out on **Good** with 65% of the participants rating it **Good** and both **Very Good** and **Neutral** being 18%.

6 Discussion

In this Chapter I discuss the limitations of our application and the result of the user study.

6.1 Application

Although the switch to the C# ray tracer improved the integration with Unity it also imposed some limitations.

One of the limitation is the ability to cast rays through objects. In the digital scene the objects consist of meshes which in turn are a set of connected triangles. The connection between these triangles is not perfect leaving tiny gaps between them. When these gaps line up with the ray tracer it is possible to cast a ray through these gaps creating the image of Figure 9. The sphere in this image starts with one if its mesh connections directly in front of the ray tracer, making the ray miss the surface of the sphere. We where not able to fix this bug but, we where able to hide it. By rotating the mesh of the sphere a tiny bit it does not start with the mesh connection directly in front of the ray tracer. Now the user needs to make a precise rotation for the bug to appear which is almost impossible because, the controls are not precise enough.

Another limitation of the C# ray tracer is parallelization. The ray tracer is build around an internal Unity function that is hard to parallelize.

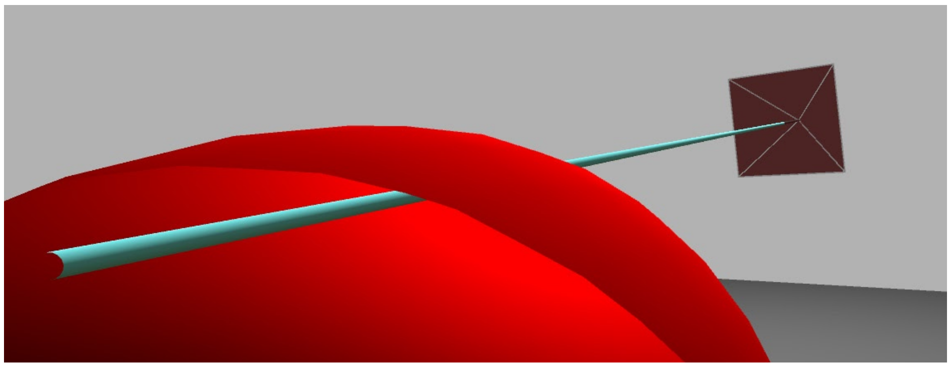


Figure 9: C# ray tracer casting a ray through a sphere.

6.2 User study

While going over the result of the user study I noticed some discrepancies within the data. Looking at the result of the personal section of the user study, Table 1. In the **No** Column three participants are (ex)CS students. Although these students have not followed the computing graphics course, their prior understanding of computing science could influence results of the user study.

In the **Yes** Column from the same Table we also see something curious. Their are two participants that are not (ex)CS students but, did pass the Computer Graphics course. I suspect that this is a precision error of our user study. These participants likely had an noteworthy relation to the Computer Graphics course but, did not have a better suiting option to select.

Furthermore the limited size of our population sample, the fact that we where not able to conduct two user studies and properly measure the influence of our application on the participants learning process could make the data unreliable. However, because of the overwhelmingly positive trend of the result I still deem the data good enough to at least draw a suggestive conclusion.

7 Conclusion

The goal of this bachelor thesis was to write an application that would give the user an environment in which they could see and experiment with the concepts of ray tracing, determine what the best way would be to present ray tracing in such an application and find out how such a tool would influence the ray tracing learning process specifically for CS students who have not followed the Computer Graphics course at RUG, non-CS students and people outside the university.

The data gathered in the user study points towards a positive influence of the ray tracing learning process. From the participants not only 90% believed that the application had a positive influence for them, 100% felt that the application would also benefit others. Overall the participants chose two or more out of the three attributes that contributed to the positive learning influence.

Looking closer at these attributes all attributes had an almost even score with only a small lead for the visualisation of the ray tracing in the scene. Combining this data with the data from the usability section of the user study and we see some mixed results. The visuals are really good 41% **Very Good** and 41% **Good**. However, The scenes and explanation of the ray tracing process scored lower. Overall it still averaged out on **Good**.

Furthermore, The application was generally easy to use 88% and not to complex 94%. Leading to the final conclusion: We successfully build an application that can present ray tracing and positively influence the ray tracing learning process specifically for CS students who have not followed the Computer Graphics course at RUG, non-CS students and people outside the university.

8 Future work

First and foremost the application has a lot of potential and it would be interesting to develop it further. In this Chapter I added a list of improvements for the application and a suggestions for a followup user study:

- **External ray tracer** The current ray tracer used by our application is build around an internal Unity ray cast function. This function forces the ray tracer to use floats for location data and makes it difficult to parallelize the ray casting process. An external ray tracer could make use of doubles instead of floats, increasing precision and would be easier to parallelize which would improve performance.

- **Everything ray traced** With the current development of real time ray tracing capable graphic cards it would be really interesting to see if it can be used for the application. By using ray tracing to render everything there would be no difference between the scene and the images generated by the camera object in the application. The rays we visualise within the application would actually be equal to the rays used to create what the user sees on his screen.

- **Virtual reality:** By adding virtual reality to the application, a mode where the user wears goggles that allows them to perceive and interact with the virtual world like it is the real world, the user could literally walk through the scene and interact with objects using special controllers in their hands. It would be interesting to research what kind of influence this would have on the learning process for ray tracing.

- **Better level system** The level system in the application has only one main track and is packed with information. The application would become a lot more accessible if the level system would be expanded. I propose a level system with groups of levels for each concept of ray tracing and goals the user could achieve. Having a group for each concept makes it easier for the user to deduce what could be learned from that section and splitting each concept into multiple levels decreases the information density. Adding goals to each level will help guide the user trough the levels. Achieving a goal will signal the user that the user is done with the level and can continue to the next. It will also clarify what each level is about.

- **User coded ray tracer** One of the users suggested to allow the user to run his own ray tracer code within the application. Although it would be really interesting to research what the influence of this concept would be on the learning process, it would be rather hard to implement it. One possibility would be a sort of module system. The modules refer to specific sections of the ray tracer allowing the backbone of the ray tracer to stay the same and making it more robust for user errors. Then the user could put his own code within the modules and see what the effect would be on the ray tracer.

- **Larger user study:** Due to time constrains our current user study had many limitations. It had a minimal amount of question, a limited group of participants and an imprecise way of measuring the influence of our application on the learning process of ray tracing. In further research it would be interesting to set up a larger user study with more questions and a larger group of participants to get a broader view of the influence of our application and the factors that play a role.

- **Split participants:** The participants of a new user study could be split beforehand effectively creating two smaller user studies. One conducted on the Computer Graphics students and one for the rest of the users. By splitting them up question could be specifically focused on each group. The Computer Graphics user study could have an exam to measure the influence of the application or specific questions for each ray tracing concept, how did the visualisation of x help and how could it be improved. The other user study could stay similar to the current one to not discourage the participants from joining it. Together with the user study even the application could be split in a more advanced version for the Computer graphics students and a simplified version for the rest group to help make it more accessible.

- **Exam:** I briefly mentioned an exam in the previous paragraph. Conducting an exam would greatly improve the data. The knowledge of the participants would preferably be measured before and after they used the application. Our current user study only asked the users if they perceived any influence from our application, to not discourage them. Measuring before and after would result in two data sets that could be compared with each other allowing for a preciser estimate of the influence of our application. It would also eliminate the vague notion of what the participant believed that the influence of the application was.

Acknowledgements

I would like to thank both my supervisors Jiri Kosinka and Steffen Frey for their support, patience, input and feedback during the project. Without their support this thesis would not have been completed. I also want to thank my project partner Willard Verschoore de la Houssaije for working together with me on this project. Finally I want to thank Ferry van Wezel, who proof read most of my thesis and added suggestions for improvements, and friends and family who supported me during this project.

References

[1] A. Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 37–45, New York, NY, USA, 1968. Association for Computing Machinery.

[2] Nvidia Corporation. Nvidia Turing GPU Architecture, 2018. `https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf`, Last accessed on 2021-03-23.

[3] Nvidia Corporation. NVIDIA Unveils Quadro RTX, World's First Ray-Tracing GPU, 2018. `https://nvidianews.nvidia.com/news/nvidia-unveils-quadro-rtx-worlds-first-ray-tracing-gp`, Last accessed on 2021-03-23.

[4] W.A Verschoore de la Houssaije. *A Virtual Ray Tracer.* Bachelor thesis, University of Groningen RUG, 2021.

[5] Epic Games. Unreal Engine 4 Documentation, 2021. `https://docs.unrealengine.com/4.27/en-US/`, Last accessed on 2021-12-24.

[6] E. Haines and T. Akenine-Möller, editors. *Ray Tracing Gems.* Apress, 2019.

[7] S. Marschner and P. Shirley. *Fundamentals of computer graphics.* CRC Press, Taylor & Francis Group, Boca Raton, FL, 2015.

[8] C. Müller and F. Gärtner. Student Project - Portable Real-Time 3D Engine. In J. Bourdin, J. Jorge, and E. Anderson, editors, *Eurographics 2014 - Education Papers.* The Eurographics Association, 2014.

[9] Unity Technologies. Unity Manual, 2021. `https://docs.unity3d.com/Manual/index.html`, Last accessed on 2021-02-23.

[10] N. Vitsas, A. Gkaravelis, A. Vasilakis, K. Vardis, and G. Papaioannou. Rayground: An online educational tool for ray tracing. In M. Romero and B. Sousa Santos, editors, *Eurographics 2020 - Education Papers.* The Eurographics Association, 2020.

Division of work

Task	Willard	Chris
Prepare GitHub and setup the project within unity	X	X
Integrate the C++ ray tracer		X
Add simple ray visualisation to the application	X	
Update ray visualisation to work with the ray tracer		X
Scene management	X	
Object selection	X	
Object move tools	X	
Adding more ray tracer integration		X
Help Menu	X	
3D rays instead of 2D		X
Switch to Unity ray tracer	X	X
Create the UI	X	
Improved 3D ray rendering	X	
Adding high resolution image functionality to ray tracer		X
Adding a shaders to better match scene and the ray tracer	X	
UI improvements	X	

Beyond this line most of the framework for the application was done and only smaller tasks remained.

Task	Willard	Chris
Ray tracer optimisation	X	
Adding transparent objects and make them behave well		X
Adding a level system and menu to access them		X
Adding a main screen		X
Adding levels	X	X
Adding a information pop-up to each level	X	
Allow objects to be added and removed for sandbox level		X
Overall bug fixes	X	X
Creating the user study format and questions	X	X
Creating and maintaining the User study		X
Building the application for Windows, Linux and Mac-OS	X	X
Manage the distribution of the user study and application		X
Create the presentation	X	X
Create a cycling intro screen used for the presentation		X

Table 5: The division of work.

User study questions

Q1: What is your relation towards computing science?
R: CS (ex)student, CS related (ex)student, No relation to CS.

Q2: Have you followed the RUG Computer Graphics course?
R: Passed, Failed, No.

Q3: If you followed the course, do you think the application would be helpful to future students of the course?
R: Yes, No.

Q4: Did the application help you understand ray tracing better?
R: Yes, No but I already understood it well beforehand, No.

Q5: Do you think the application can help other people understand ray tracing better?
R: Yes very helpful, Yes moderately helpful, No.

For this question multiple responses where allowed.
Q6: Which of the following things do you think where successful in helping you understand ray tracing?
R: Information texts, Visualisation of ray tracing, Ability to experiment with various settings.

Open question.
Q7: If you have any comments about the educational part of the application you can leave them here.

Q8: Did you find the application easy to use?
R: Yes, Yes but some things were confusing or difficult, No but it was not very confusing or difficult either, No.

this question allowed a rating for each sub topic: S.
Q9: What do think of these aspects of the application?
S: The visuals, The user interface, The controls, The scenes and explanation of raytracing.
R: Very good, Good, Neutral, Bad, Very Bad.

Q10: What do you think of the complexity of the application?
R: To simple, Perfect, To complex.

Open question.

Q11: If you have any comments about the technical aspects of the application you can leave them here.

Open question.

Q12: This field is for overall feedback, bugs, things you wanted to do in the application but where unable to, or just a comment for the creators.