Ray Tracing Explorer in VR

Lídia Gómez Custódio

Thesis to obtain the Master of Science Degree in

Engenharia Informática e de Computadores

Supervisors: Prof. Alfredo Manuel Dos Santos Ferreira Júnior
            Prof. João António Madeiras Pereira

Examination Committee

Chairperson: Prof. Name of the Chairperson
Supervisor: Prof. Alfredo Manuel Dos Santos Ferreira Júnior
Members of the Committee: Prof. Name of First Committee Member
                           Dr. Name of Second Committee Member
                           Eng. Name of Third Committee Member

November 2022
This work was created using \LaTeX\ typesetting language in the Overleaf environment (www.overleaf.com).
Acknowledgments

I want to thank my advisors, Alfredo Ferreira, Nuno Trindade, and João Madeiras Pereira, for their guidance and continuous support throughout the development of this work.

Thank you to the students who volunteered to test RayTracerVR, for, without them, we could not have done a proper user evaluation of this work. I also want to acknowledge the authors of the papers of previous work analyzed, for they inspired and influenced the development of this work.

Finally, I would like to express my gratitude to my family and friends for their encouragement and moral support that made this work possible.
Abstract

When learning new concepts and topics, having a visual medium to aid in the explanation can positively impact the overall grasp of the given information. Algorithms are processes that can easily be represented visually, allowing the receiver to consolidate its specificities. Virtual reality applications elevate this visualization by immersing users in a virtual environment, which has been increasingly used for educational purposes.

This work addresses the problem of learning ray tracing, an algorithm that occurs in a 3-dimensional space but is usually explained using 2-dimensional mediums. We present a virtual reality prototype created for visualizing ray tracing, RayTracerVR. It allows the users to visualize several aspects of the algorithm, namely its code accompanied by a textual description and the rays visually created in the scene. We evaluated its effectiveness by comparing users' acquired knowledge with our virtual reality application by answering a theoretical test before and after their interaction. We noted an increase in the acquired knowledge and a positive usability experience.

Keywords

Ray tracing; Virtual reality; Computer science education; Educational simulations
Resumo

Ao aprender novos conceitos e tópicos, ter um meio visual para auxiliar na explicação pode impactar positivamente na compreensão geral das informações fornecidas. Algoritmos são processos que podem ser facilmente representados visualmente, permitindo ao recetor consolidar suas especificidades. As aplicações de realidade virtual elevam essa visualização por meio da imersão dos utilizadores em um ambiente virtual, que vem sendo cada vez mais utilizado para fins educacionais.

Este trabalho aborda o problema de aprendizagem de ray tracing, um algoritmo que ocorre num espaço tridimensional, mas geralmente é explicado usando meios bidimensionais. Apresentamos um protótipo de realidade virtual criado para visualização de ray tracing, RayTracerVR. Este permite aos utilizadores visualizar vários aspetos do algoritmo, nomeadamente o seu código acompanhado por uma descrição textual e os raios criados visualmente na cena. Avaliamos a sua eficácia comparando o conhecimento adquirido dos usuários com a nossa aplicação de realidade virtual, respondendo a um teste teórico antes e depois de sua interação. Notamos um aumento no conhecimento adquirido e uma experiência de usabilidade positiva.

Palavras Chave

Ray tracing; Realidade Virtual; Educação na Ciência da Computação; Simulações Educacionais
Contents

1 Introduction 1
   1.1 Motivation ......................................................... 3
   1.2 Objectives ......................................................... 3
   1.3 Thesis structure ................................................... 4
2 Ray tracing algorithm 5
   2.1 Ray generation ..................................................... 7
   2.2 Ray-object intersection .......................................... 7
   2.3 Shading ............................................................. 7
   2.4 Code ............................................................... 8
   2.5 Learning ray tracing .............................................. 8
3 Related Work 15
   3.1 Interactive learning of algorithms ............................... 17
   3.2 Virtual reality in educational applications ..................... 19
   3.3 Discussion ........................................................ 23
4 RayTracerVR 27
   4.1 List of requirements .............................................. 29
   4.2 Architecture ....................................................... 29
   4.3 Algorithm Coding .................................................. 31
   4.4 User interface ...................................................... 33
       4.4.1 Mockup .......................................................... 33
       4.4.2 World environment ........................................... 34
       4.4.3 Scene .......................................................... 34
       4.4.4 Panels .......................................................... 37
5 Evaluation 43
   5.1 Presentation and Preliminary Feedback .......................... 45
   5.2 Evaluation ........................................................ 46
       5.2.1 Methodology .................................................. 46
List of Figures

2.1 Diagram from “An Introduction to Ray Tracing” [1] showing the creation of reflected, refracted, and shadow rays from a single primary ray ................................. 10
2.2 Image from “Ray Tracing from the Ground Up” [2] of rays shooting from pixels intersecting the objects in the scene .................................................... 11
2.3 Sequence of frames from the first video in the series by NVIDIA showing the creation of reflected, refracted, and shadow rays from a primary ray ......................... 12
2.4 Frames from “Disney’s Practical Guide to Path Tracing” where we can see the path of a single ray (left) and the full illuminated scene (right) .............................. 13
2.5 Interactive exercise from Khan Academy’s “Pixar in a Box - Rendering”, where the user can choose which ray is seen in the scene ........................................... 13

3.1 VisuAlgo [5] final step of an insert operation into a binary search tree ...................... 17
3.2 Algorithm Visualizer [6] final step of an insert operation into a binary search tree .... 18
3.3 Algorithms Animated [7] insert operation into a binary search tree ......................... 19
3.4 OOPVR [8], where an instance is created by placing the parameters of the constructor on the window sill ................................................................. 20
3.5 User picking up the boxes to order them according to bubble sort in VirtSort [9] ........ 21
3.6 User interface of VR application [10] ................................................................. 21
3.7 Two VR medical tools ...................................................................................... 22
3.8 Two VR tools for education ............................................................................. 22

4.1 RayTracerVR system architecture ................................................................. 31
4.2 Algorithm information structure ..................................................................... 32
4.3 Algorithm flowchart ....................................................................................... 33
4.4 Mockup of the application’s 3D environment .................................................. 34
4.5 Virtual environment ....................................................................................... 35
List of Tables

3.1 Comparation between applications that visualize algorithms ........................................... 24
3.2 Comparation between applications that use immersive technologies .......................... 25
4.1 Playback control requirements .......................................................................................... 29
4.2 Code visualization requirements ..................................................................................... 30
4.3 Ray tracing visualization requirements ............................................................................ 30
5.1 Problems and solutions identified for RayTracerVR, along with a severity scale (1 - 4), cost in person/month, and solved status ................................................................. 53
Introduction

Contents

1.1 Motivation ............................................. 3
1.2 Objectives ............................................ 3
1.3 Thesis structure ....................................... 4
Ray tracing is a technique for generating realistic images by simulating light transportation or, as A. S. Glassner (1989) [1] describes, “a technique for image synthesis: creating a 2-D picture of a 3-D world”. This concept has been around for some time and may not be simple to understand on a first approach. It is usually explained using a 2-dimensional medium, such as paper or a blackboard.

In recent years, there has been an increase in educational tools that use immersive technologies such as virtual reality (VR) or augmented reality (AR). These technologies allow the users to immerse themselves in a simulated environment and interact directly with materials and concepts. This hands-on experience can facilitate the learning process.

1.1 Motivation

An algorithm is a set of instructions that will produce a specific result. In ray tracing’s case, this result is an image. We could easily explain how it works by reciting each step. However, this is not the most used approach since it can be more challenging and confusing to understand. Most teachers use visual diagrams to explain each ray tracing step. It gives the students a clearer spatial sense of each operation and can facilitate its comprehension complementing the algorithm’s abstract description (or pseudo-code).

The purpose of ray tracing is similar to that of a digital camera. It takes a 3-dimensional scene and creates a 2-dimensional representation. The image is only the final result, as the execution of ray tracing occurs in a 3D scene. Nonetheless, when we try to explain this algorithm, we can rely on 2D mediums, like hand-drawn diagrams on a piece of paper or even animated diagrams in a video. With these methods, we lose a third dimension essential to comprehend the algorithm fully.

1.2 Objectives

By employing virtual reality, we can better explain the ray tracing algorithm. Since there is a gap in this particular field, we proposed this research question for this project:

**Does virtual reality facilitate learning the inner workings of ray tracing?**

We confirm that learning ray tracing through VR better enhances its understanding than using traditional methods. To validate this hypothesis, we set the following objectives:

**Create a virtual reality application.** We created a VR application, RayTracerVR, aimed at visualizing and explaining ray tracing. With this application, we expected that the people who interact with it could comprehend and understand how the algorithm works and be able to answer theoretical questions regarding the topic.
**Visualize ray tracing.** RayTracerVR simulates the algorithm’s execution according to user inputs. We presumed that this ability to change the inputs could facilitate understanding the algorithm in different conditions than other traditional teaching methods. Moreover, the application illustrates the algorithm step-by-step, allowing the user to control the playback.

**Make the application available online.** We intend for our prototype application is available online for download, allowing users to install and run the application locally. Also, this project is open-source, and as such, the project’s source code is available online to download.

**Validate the research question.** After developing the prototype, we tested our hypothesis that people will learn ray tracing better using virtual reality instead of traditional methods. We tested 16 users who had learned ray tracing via traditional methods to interact with our application. They answered a theoretical test before and after interacting with RayTracerVR to assess their understanding of the algorithm. We concluded that the users showed an increased knowledge after interacting with the application. We also registered the problems and issues users had while interacting and delineated possible solutions. Some of these were already addressed for this work.

### 1.3 Thesis structure

The following chapter will present an abbreviated explanation of the ray tracing algorithm and traditional methods used in learning it. Next, we discuss related work in Chapter 3. In Chapter 4, we characterize our proposed solution with RayTracerVR, and in Chapter 5, we discuss how we evaluated it. Finally, we present our conclusions in Chapter 6.
Ray tracing algorithm

Contents

2.1 Ray generation ................................................... 7
2.2 Ray-object intersection ..................................... 7
2.3 Shading .......................................................... 7
2.4 Code ............................................................ 8
2.5 Learning ray tracing ......................................... 8
Ray tracing is an algorithm for rendering photo-realistic images by simulating light transportation. The algorithm creates light rays representing the paths of the light particles moving through space, reflecting and refracting through the objects in the scene until they reach our eyes or the camera. The light that reaches the camera at a certain point will be translated into color for that image’s pixel. The final image will, thus, be rendered by determining the color at each pixel.

However, as we will soon see, the rays created by ray tracing do not simulate the path of the light from the light source to the camera, called forward ray tracing. It would be expensive to compute, considering that most rays from a light source will not hit the camera. Ray tracing computes the rays in the opposite way, from the camera until they hit the light source, which is why it is also called a backward ray tracer.

All in all, ray tracing starts with a 3D scene with several objects with different textures and creates a 2D image showing the scene from a particular position and perspective, depending on the camera. This image is created by calculating the final color of each pixel. We can subdivide the whole process into three parts: ray generation, ray-object intersection, and shading.

2.1 Ray generation

In the first part, we create a ray for each pixel in the image. A ray is an infinite line mathematically defined by an origin and a direction. These first rays created are called primary rays: the origin is at the center of their corresponding pixel and has a direction defined by the vector that goes from the position of the camera (often called eye) to the center of the pixel.

2.2 Ray-object intersection

For each primary ray created, we need to check with which objects in the scene the ray intersects and save the closest one. We can mathematically determine the intersection points by using the analytical expressions of the objects, for example, spheres, planes, and triangles. The saved object will be the one in direct view of the pixel, and for which we will determine the color at the intersection point. If the ray does not hit any object, the final color at that point will be the background color.

2.3 Shading

Before determining the color at the intersection point, we need to check if this point is in direct view of any light sources; otherwise, the point will be in shadow. To accomplish this, we create a ray from the intersection point to the light source’s position for all the light sources in the scene. These rays are called shadow rays. Much like the primary rays, we test the intersections with all the objects in the
scene. However, for shadow rays, it is not necessary to know which is the closest object, only if there were any intersections, since it is required to know if the intersection point has a direct view to a light source.

If the intersection point is illuminated, we can determine the color by combining the local color, the color that comes from the reflection direction, and the refraction direction. The local color is determined using a shading model. One of the most simple and commonly used is Blinn-Phong shading since it determines the diffuse color of the surface along with the specular reflections. Depending on the object’s material properties, there can be a reflective and refractive component. For any of these components, we determine the reflected or refracted rays (secondary rays) and proceed to determine their colors as if they were primary rays. This is the recursive part of the ray tracing algorithm since these secondary rays can consequently create other secondary rays infinitely until it reaches the maximum recursion depth defined.

2.4 Code

We will now present a simplified version of the algorithm usually used to explain the fundamental aspects of ray tracing: a recursive algorithm that will determine the color of each pixel in the image. The procedure that computes the pixel’s color is depicted by the function trace in Algorithm 2.1. This function first detects if the ray intersects any object in the scene to determine its color; otherwise, the color of that pixel will be the background color. If the ray intersects an object, we must first check if the object is illuminated by the light sources, i.e., if a ray from the object to the light source has no obstacles in its path. Granted that the object is illuminated, the object’s color at the intersection point can be determined using a chosen shading model. Afterward, we check for the stopping condition of the recursion if the current depth of the ray is greater than the maximum defined. In that case, the function returns the color determined. Supposing that the maximum depth was not reached, we can determine the colors resulting from the reflected and refracted rays, which compute using the trace function. Combining these colors with the color previously determined will allow for reflections and refractions of the light, which will allow for an approximation of the realistic color at that point. This whole procedure described will happen for each pixel in order to build the entire rendered image.

2.5 Learning ray tracing

The most straightforward way to learn the ray tracing algorithm would be by reading a book on the matter. There are multiple computer graphics studies solely dedicated to the intricacies of the ray tracing algorithm, considering its relevance in the rendering of computer-generated images.
Algorithm 2.1: Ray tracing

Function \texttt{trace(scene, ray, depth)}:
\begin{algorithmic}
\State \textbf{for each object in the scene do}
\State \hspace{1em} compute ray-object intersections and save the closest one
\State \textbf{if no intersection then}
\State \hspace{1em} \textbf{return} background color
\State \textbf{else}
\State \hspace{1em} color $\leftarrow$ black
\State \textbf{for each light in the scene do}
\State \hspace{1em} create shadow ray for the light \textbf{for each object in the scene do}
\State \hspace{2em} compute ray-object intersections
\State \hspace{2em} \textbf{if no intersection then}
\State \hspace{3em} color $\leftarrow$ shadingModel(ray)
\State \textbf{if depth $\geq$ maxDepth then}
\State \hspace{1em} \textbf{return} color
\State \textbf{if object is reflective then}
\State \hspace{1em} ray$_r \leftarrow$ reflectedRay(ray)
\State \hspace{1em} color$_r \leftarrow$ trace(scene, ray$_r$, depth + 1)
\State \hspace{1em} color $\leftarrow$ combine(color, color$_r$)
\State \textbf{if object is transmissive then}
\State \hspace{1em} ray$_t \leftarrow$ refractedRay(ray)
\State \hspace{1em} color$_t \leftarrow$ trace(scene, ray$_t$, depth + 1)
\State \hspace{1em} color $\leftarrow$ combine(color, color$_t$)
\State \textbf{return} color
\end{algorithmic}

Function \texttt{main()}:
\begin{algorithmic}
\State \textbf{for each pixel do}
\State \hspace{1em} ray $\leftarrow$ primaryRay(pixel.x, pixel.y)
\State \hspace{1em} pixelColor $\leftarrow$ trace(scene, ray, 1)
\end{algorithmic}
The book “An Introduction to Ray Tracing” [1] is a classic example of the first approach to ray tracing. It starts by explaining how the algorithm can create images, clarifying the basic concepts of the camera model, how rays are traced, and the different types of rays, which includes explaining the recursive nature of the algorithm when it comes to transporting light in the scene. Regarding visualization, the book contains several diagrams and images throughout that can aid the understanding of the algorithm. Most of the time, these diagrams are of rays in a scene, showing intersections, reflections, or refractions, like in Figure 2.1. These can give the reader a better idea of the vectorial mathematics necessary to calculate the rays in the scene. The book’s last chapter contains code samples for structures and functions required to write a fully functional ray tracing algorithm.

“Ray Tracing from the Ground Up” [2] is focused on the programming aspect of writing a ray tracing application. Every chapter contains a detailed description of the topic, accompanied by diagrams and images (like the one in Figure 2.2) and comprehensive code samples on a possible implementation of the topic being explained in the C++ programming language. Seeing that the book is intended for students or professionals in the computer-graphics field that are accustomed to programming and especially C++, the provision of code samples next to the description of each part of the algorithm certainly enhances the comprehension of the concepts.

Descriptive resources like the ones mentioned are suitable for a better understanding of ray tracing when read in order. However, they can be overwhelming when learning the subject and fundamental concepts for the first time. An oral explanation, accompanied by other media types such as images and videos, can be another alternative for a more engaging and entertaining medium.

The seven-part video series hosted by NVIDIA’s Eric Haines1 presents some of the basic notions of

---

1 https://www.youtube.com/playlist?list=PL5B692fm6–sgm8Ulava01lvUo@FOCSR
ray tracing in their first part and go on to explain other essential aspects like ray tracing effects and the rendering equation. Having someone with good presentation skills explaining a topic is better to engage the audience and improve their understanding. Additionally, the videos are accompanied by animated diagrams that show how the algorithm works, enhancing our understanding of it. In the first video of the series, this is especially important to understand the concept of ray casting; when rays are cast from the eye source, they see which elements in the scene they intersect and which new rays are created with reflections and refractions. Seeing the moving path of a ray can promote the assimilation of the concept. Figure 2.3 shows several frames from the video that showcases how a single ray’s path is visualized.

Regarding animated visualizations, the video “Disney's Practical Guide to Path Tracing”\(^2\) provides a straightforward and entertaining explanation of path tracing, how the light interacts with the surfaces and how the algorithm can be more efficient using ray sorting. It demonstrates how a scene can be lit by showing the process for a single ray (Figure 2.4): the scene starts pitch-black, and when a ray from the light source (sun, in this case) hits the rock on the ground, a small area of its texture is visible; then the ray is reflected in the tree, and we get to see a little more of the tree, and so on, until the whole scene is illuminated. Additionally, it shows how tracing every ray emitted from the light source will be computationally expensive because only a few sets of rays will hit the camera. The animation reminiscent of classical Disney shorts contributes to an appealing presentation and engages the audience in the theme. Besides, the language and images used do not restrict the viewers to only computer science students and professionals, allowing middle- or high-schoolers to grasp the basic notions of ray tracing and path tracing.

There are other types of platforms dedicated to teaching that contain video or textual explanations

---

2https://www.youtube.com/watch?v=frLwRLS_ZR0
of the concepts, accompanied by exercises where the users can test and consolidate their knowledge. An example of this platform is Khan Academy. One particular module, the Rendering module\textsuperscript{3}, focuses on explaining the basics of ray tracing. It is organized into two main parts, one more directed to school students, so the concepts are explained more simply; the second is more complex, which explains the mathematics behind ray tracing. Each part contains several small videos with explanations, animated diagrams, and practical exercises where the user can consolidate what they learned in the video. One particular exercise has a small interactive scene (Figure 2.5) where the user can see a house, an image plane grid, and a ray that goes through a specific pixel in the plane to the house. The user can choose which ray is seen through the grid on the left, which represents the final image plane, and they can also see the final color of that pixel. This exercise provides a simple visualization of the calculation of the final image through ray tracing: rays that go through each pixel intersect the objects in the scene (the house in this case) and calculate the final color of that pixel accordingly. Even though this visualization does not show the secondary rays created, it presents a straightforward visualization of ray tracing.

Lastly, regarding an online platform, we have the project Rayground \cite{Rayground}, which is a web-based application for ray tracing. The application’s interface is divided into two main sections; on the left, we can see the final result of the rendering scene, and on the right, we can alter the ray tracing code that generates the scene (Figure 2.6). It allows changing the code of different stages of ray tracing, such

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figures/sequence.png}
\caption{Sequence of frames from the first video in the series by NVIDIA showing the creation of reflected, refracted, and shadow rays from a primary ray}
\end{figure}

\textsuperscript{3}https://www.khanacademy.org/computing/pixar/rendering
as the scene description and the primary rays’ generation. The playground allows the user to see the changes in code in real time, encouraging the acquaintance with the algorithm and the importance of certain variables. The project was built using WebRays [11], a WebGL-based ray intersection library for the web. The availability of the application online in a simple browser, with no need to install any plug-in or software, is a great asset when it comes to online teaching. The authors of the Rayground project described in the paper “Remote Teaching Advanced Rendering Topics Using the Rayground Platform” [12] how this platform can be used for online teaching. The authors describe how an advanced computer graphics course can be adapted for online coursework and how the student projects can be adapted to use Rayground as the central platform to learn ray tracing.
Rayground is a framework for rapid prototyping of algorithms based on the ray tracing paradigm. This project shows how an unidirectional path tracer with a point light and diffuse materials can be implemented using the Rayground API.

**Figure 2.6:** Rayground [3] interface with sample project of the Cornell Box [4] scene
3 Related Work

Contents

3.1 Interactive learning of algorithms ........................................... 17
3.2 Virtual reality in educational applications .............................. 19
3.3 Discussion ........................................................................... 23
Until now, we have seen examples that teach ray tracing by relying more on the presentation of the subject rather than directly interacting with the running algorithm. In the coming sections, we present a diverse set of tools that range from interactive algorithm visualization to VR applications in the educational field. Subsequently, we discuss these tools and compare their characteristics.

3.1 Interactive learning of algorithms

Many platforms allow the visualization of algorithms by also granting the user to interact with the visualization, be it to change the input values or to stop and play the animation of the algorithm at their own pace.

VisuAlgo [5] is a website that shows visualizations of several algorithms and data structures. It is a growing website where new algorithms are added over time. At the moment of writing, it contains twenty-four different algorithms shown on the website’s front page, where each module shows a diagram of the algorithm/data structure. When choosing an option, we are presented with an extensive description of the data structure, which includes motivation and explanations of every operation available to that data structure. We can choose which operation we wish to simulate within the visualization itself. Afterward, we can see a step-by-step animation of the operation running, including the pseudo-code, which has the current step highlighted, alongside a text explanation of that step. It also includes playback buttons (play, stop, go back, go forth) that let the user change the play of the animation as they choose. Some modules allow the user to change the input values, though they can be limited in other modules. All these elements can be seen in Figure 3.1. Additionally, some algorithms/data structures provide a training module that consists of a small quiz regarding that algorithm/data structure with which the user can test their knowledge and consolidate the intricacies of the algorithm.

![Figure 3.1: VisuAlgo [5] final step of an insert operation into a binary search tree](image-url)

17
Algorithm Visualizer [6] is another example of a website showing algorithms’ visualizations. When choosing an algorithm, the screen gets divided into three zones: one with the visualization and animation of the algorithm, the console output, and the complete code in Java or C++. Every algorithm has a page with a description of the algorithm. After initializing the program, we can see the animation of the algorithm and the program’s output in the console, as shown in Figure 3.2. It also contains playback buttons that allow the user to play, stop and change the speed of the animation. Having the whole code is helpful for computer science students when they need to implement the algorithm. Additionally, the steps of the algorithm are highlighted while the animation is running; however, it focuses more on the code that allows the animation and not the current step that the animation is playing.

Algorithms Animated [7] is a blog-style website that explains and exemplifies several different algorithms and data structures. Each module explains the algorithm/data structure and various animations for each operation in that data structure. The interaction revolves mainly around the playback of the animation, and sometimes the user can choose where the algorithm will start, like choosing the node to remove on a tree; however, they cannot change the input values. As shown in Figure 3.3, the interaction with the animation is minimal, the user cannot choose the nodes inserted into the binary search tree, and no code is provided. However, a short description gives context to the operation being executed. Overall, this website can be an initial approach to these algorithms and data structures, but it will not be enough to learn them thoroughly.

These algorithm visualization applications can possibly grant a level of engagement in the users that is not achieved by traditional education methods, considering that interacting with the algorithm can help the user gain a better level of understanding. Moreover, this interaction can be enriched with VR, seeing that the user can gain a degree of immersion that is impossible in a 2D application. The following section will present how VR can be used in educational applications.
3.2 Virtual reality in educational applications

Nowadays, VR has been increasingly used in educational applications due to its easier accessibility and the advantages of immersing users in a virtual world. When physically engaging with the subject matter, the user can improve their perception of the concepts and better consolidate the topic. We can find projects that use VR in several fields; still, our primary focus is computer science.

Pirker et al. [13] did a great job of summarizing the main advantages and limitations of using VR technologies in the education of computer science concepts by analyzing previous papers. In total, they reviewed 12 papers that fit their criteria. They concluded that the reduced number of relevant papers in this area shows that there is still room for further studying and research. As they described, the main advantages pointed out by several authors were interaction and immersion, visualization and metaphors, playful design, and social experiences [13]. One of the issues pointed out refers to the user interface (UI) design. Traditional UI elements, such as buttons and menus, can break the VR environment’s immersion. In that case, the interaction with the environment should be designed so that the immersion is not broken and the user can intuitively interact with the environment.

“OOPVR” [8] relied on analogies and visualization to explain basic object-oriented programming (OOP) concepts with VR. In their analogy, a blueprint represented a class, and a house represented an instance of the class, where each room was an instance method that received the arguments as boxes with values inside. In Figure 3.4, we can see the creation of a new instance of the class Person by placing the constructor’s parameters on the window sill. This analogy allowed users to move around the...
Figure 3.4: OOPVR [8], where an instance is created by placing the parameters of the constructor on the window sill.

Kong’s work [9], “VirtSort”, focuses on teaching computer science algorithms, specifically sorting algorithms. The user is set in a virtual world where they can choose the algorithm to be visualized. After choosing the algorithm, a set of boxes is displayed in front of them where they will interactively run the algorithm. Also, if they choose, they can be presented with an audio and text explanation of the algorithm before executing it. Each box has a number written on it, placed on top of colored triggers, where the trigger’s color can indicate the algorithm’s step. For example, blue indicates that the current box is being compared with the others, yellow the already sorted boxes, and green when all the boxes are correctly placed. At each step, the user can pick up the boxes, compare them, and place them in the correct position according to the selected algorithm, as shown in Figure 3.5.

Still on the topic of sorting algorithms, the authors of “The Potential of Virtual Reality for Computer Science Education - Engaging Students through Immersive Visualizations” [10] made both VR and Web-based applications for visualizing sorting algorithms. In total, they implemented nine sorting algorithms. The user can spawn several elements that will be sorted with the chosen sorting algorithm. The user has complete control of the visualization, being able to play, pause, view previous and next steps, and change the step speed. They can also visualize more than one algorithm at once to compare them easily. The VR interface of this project is shown in Figure 3.6. From the results of their study, they concluded that users showed more engagement using the VR application.
As previously mentioned, there is a myriad of projects that use VR in other educational fields. We will present some projects that exemplify different approaches to teaching various topics.

In the medical field, the project “Virtual Reality For Anatomical Vocabulary Learning” [14] allowed the visualization of the human body to learn anatomical vocabulary. Using VR, they could have a whole human body available for studying. The application allowed them to focus and select different body parts to have a closer and more interactive look at them. Figure 3.7, on the left, shows the heart selected with a textual description. The practice model allowed the users to take an assessment test where a part of the body was shown without any vocabulary. The goal was for the user to input the correct nomenclature through a virtual keyboard.

Additionally, “AirwayVR” [15] focuses on learning and training a specific medical procedure. To provide a more meaningful learning experience for novice learners, the procedure was separated into different modules so that the user could learn and practice them at a time (Figure 3.7, right, shows the practice environment of an operating room). In the case of professionals already familiar with the procedure, they could use a module designed to practice before an actual complex procedure.
In the mathematical field, more directed to middle- and high-school students, the main focus of the project “Teaching platonic polyhedrons through augmented reality and virtual reality” [16] was to teach platonic polyhedrons through the means of augmented and virtual reality. The students could interact with each polyhedron, deconstructing them through their vertices, edges, or faces. The application also allowed students to see a planification view of the polyhedron, allowing them to better understand the structure of these objects. Figure 3.8, on the left, shows the application’s environment in VR of an icosahedron with its faces separated.

Outside of the science field, “Hand-by-Hand mentor” [17] is an application that helps beginner piano players follow a piece of music using augmented reality (AR). With the headgear on, the users can still see the piano keyboard, but now the keys that should be played will be displayed in a highlighted color (Figure 3.8, right). This project is an excellent example of how augmented reality helped beginners learn a particular practical process: playing a piano piece with specific fingerings and hand motions while reducing the cognitive overload of reading a piece of sheet music.

The list of projects presented is not extensive; countless more applications use VR for educational
purposes. Nonetheless, the intention was not to make an exhaustive analysis of all the different solutions found in this particular field. We believe we have enough to identify the strengths of creating an educational application using VR.

3.3 Discussion

The diverse teaching methods analyzed in the previous sections offer specific advantages and disadvantages according to the medium used and the theme addressed. First, we saw applications that allow the user to control the visualization of the algorithm, sometimes possibly changing the algorithm’s input. Following that, we saw projects that used VR to teach concepts from different areas of knowledge, where its main advantage is allowing the user to interact physically with the materials in a virtual world.

Nevertheless, we want to compare some characteristics that not every project provides since not all allow interaction from the user’s side. We will separate this comparison into two parts: first, the projects that are related to the teaching of algorithms (Table 3.1) and then the applications that use immersive technologies (Table 3.2). The projects considered are referenced in each row, and the comparative characteristics are in each column.

When comparing applications that visualize algorithms, we considered five main characteristics: code, code step-highlight, change input, learning module, and training module.

- **Code** Indicates which type of code the application uses to explain the algorithm if any. Half of the projects used pseudo-code, whereas the other half did not use code descriptions. This can be due to the last two projects being more directed to visualizing and practicing the algorithms and not intensely studying them.

- **Code step-highlight** Denotes that the application highlights each step of the algorithm while running on top of the code it shows. Only VisuAlgo provides good feedback on this since Algorithm Visualizer highlights lines of code that are more specific to the creation and animation of the diagrams shown and not precisely the algorithm being executed.

- **Change input** Indicates whether the application allows the user to change the input values of the algorithm that will be executed. This change of input can be limited to some applications.

- **Learning module** Describes if the application provides any descriptive explanation and context of the algorithm that will be executed. Considering that the applications aim to visualize and teach algorithms, all provide some sort of learning module.

- **Practice module** Indicates whether the user can put their knowledge of the algorithm into practice. As seen in Table 3.1, not all of them support practical exercises.

Regarding the applications that use immersive technologies, we compared them considering four characteristics: interaction, move in space, textual description, and hardware.
Table 3.1: Comparison between applications that visualize algorithms

<table>
<thead>
<tr>
<th>Application</th>
<th>Code</th>
<th>Code step-highlight</th>
<th>Change input</th>
<th>Learning module</th>
<th>Practice module</th>
</tr>
</thead>
<tbody>
<tr>
<td>VisuAlgo</td>
<td>Pseudo-code</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Algorithm Visualizer</td>
<td>Pseudo- and</td>
<td>Not always</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source-code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithms Animated</td>
<td>N/A</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>VirtSort</td>
<td>N/A</td>
<td>Array size, not</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting algorithms VR</td>
<td>Source-code</td>
<td>Array size, not</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application</td>
<td></td>
<td>values</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Interaction** Describes how the user can interact with the environment. In most cases, the primary interaction is pointing and clicking with the handheld controllers, and some projects even allow grabbing objects in the scene. The only exception is Hand-by-Hand mentor since it is an augmented reality application where the only interaction involves playing the piano keyboard.

**Move in space** Indicates whether the user can displace themselves in the virtual space depicted. This can provide more freedom to the user if the space is vast or to approach a specific area. The main disadvantage this can bring if it is not well designed is inducing motion sickness in the user during the movement.

**Textual description** Indicates if the application provides some explanation of the topic that is being addressed. Indeed, most use this to contextualize the application’s environment or provide learning materials.

**Hardware** Indicates for which type of equipment the application was developed. Some projects do not specify the headset and controllers used, but most were developed with Oculus Rift or HTC Vive.

We believe that using an interactive application through VR could be very beneficial to understanding the ray tracing algorithm. VR can provide the 3D scene inherent to the setting of ray tracing. Through animations, we can enhance the diagrams usually used in theoretical approaches to ray tracing. Like the interactive applications of algorithms, we can show the algorithm’s pseudo-code with each step highlighted so that the user can keep track of the execution; and the user can interactively change the input of the visualization, like choosing which ray to see executed. Lastly, a training module could help users consolidate their understanding of the ray tracing algorithm.
Table 3.2: Comparison between applications that use immersive technologies

<table>
<thead>
<tr>
<th>Application</th>
<th>Interaction</th>
<th>Movement in space</th>
<th>Textual description</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOPVR</td>
<td>Point and click</td>
<td>✓</td>
<td>✓, tablet with code</td>
<td>Not specified</td>
</tr>
<tr>
<td>VirtSort</td>
<td>Grab boxes, point, and click</td>
<td>✓</td>
<td>✓, algorithm explanation</td>
<td>Oculus Rift and HTC Vive</td>
</tr>
<tr>
<td>Sorting algorithms VR application</td>
<td>Grab, point, and click</td>
<td>✓</td>
<td>✓, algorithm code, and counter for swaps and operations</td>
<td>HTC Vive</td>
</tr>
<tr>
<td>VR for Anatomical Vocabulary Learning</td>
<td>Grab and move parts of the anatomy, virtual keyboard</td>
<td></td>
<td>✓, description of anatomy</td>
<td>Oculus Rift</td>
</tr>
<tr>
<td>AirwayVR</td>
<td>Grab equipment, move patient, point and click</td>
<td>Not specified</td>
<td>✓, in learning modules</td>
<td>Oculus Rift and HTC Vive</td>
</tr>
<tr>
<td>Polyhedrons in VR</td>
<td>Point and click</td>
<td></td>
<td>Minimal (some didactic resources)</td>
<td>HTC Vive</td>
</tr>
<tr>
<td>Hand-by-Hand mentor</td>
<td>Playing keys on a real piano</td>
<td></td>
<td></td>
<td>Not specified</td>
</tr>
</tbody>
</table>
RayTracerVR

Contents

4.1 List of requirements ................................................. 29
4.2 Architecture ......................................................... 29
4.3 Algorithm Coding ..................................................... 31
4.4 User interface ......................................................... 33
Our proposed solution is named RayTracerVR: an open-sourced VR application. This application has the primary goal of exploring and visualizing the ray tracing algorithm as a learning platform as well as validating our research question. The user needs a VR headset and controllers to interact with the application. The user can change the primary ray shown in the scene and interact with the algorithm’s playback: starting, stopping, going forwards, and backward. The virtual environment also contains information panels regarding the executing algorithm, such as the pseudo-code or textual description of the step. These panels give the user more context to the executing algorithm.

4.1 List of requirements

Informal meetings with stakeholders were held to delineate the possible requirements for this application. The more relevant requirements were condensed into three lists, grouped by topic: playback control in Table 4.1, code visualization in Table 4.2, and ray tracing visualization in Table 4.3. The functionalities not implemented in this work will be left for future work.

<table>
<thead>
<tr>
<th>Playback control</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play/continue algorithm execution</td>
<td>✓</td>
</tr>
<tr>
<td>Pause algorithm execution</td>
<td>✓</td>
</tr>
<tr>
<td>Stop algorithm execution</td>
<td>✓</td>
</tr>
<tr>
<td>Restart algorithm</td>
<td>✓</td>
</tr>
<tr>
<td>Go to next step of the algorithm</td>
<td>✓</td>
</tr>
<tr>
<td>Go to previous step of the algorithm</td>
<td>✓</td>
</tr>
<tr>
<td>Step into current function</td>
<td>✓</td>
</tr>
<tr>
<td>Step out of current function/iteration</td>
<td></td>
</tr>
<tr>
<td>Skip code line</td>
<td></td>
</tr>
<tr>
<td>Add breakpoints and conditional breakpoints</td>
<td></td>
</tr>
<tr>
<td>View list of breakpoints</td>
<td></td>
</tr>
<tr>
<td>Activate/deactivate breakpoints</td>
<td></td>
</tr>
<tr>
<td>Run to breakpoint</td>
<td></td>
</tr>
<tr>
<td>Change speed of algorithm visualization</td>
<td>✓</td>
</tr>
<tr>
<td>Change panels size and distance from user</td>
<td>✓</td>
</tr>
</tbody>
</table>

4.2 Architecture

RayTracerVR was developed with Unity. This game engine was chosen due to the easier learning curve since previous work with it had been carried out and for its ease of creating VR applications. The
Table 4.2: Code visualization requirements

<table>
<thead>
<tr>
<th>Code visualization</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show complete algorithm code</td>
<td>✓</td>
</tr>
<tr>
<td>Highlight current code line</td>
<td>✓</td>
</tr>
<tr>
<td>Highlight breakpoints</td>
<td></td>
</tr>
<tr>
<td>Show values of current variables</td>
<td>✓</td>
</tr>
<tr>
<td>Change values of variables</td>
<td></td>
</tr>
<tr>
<td>Syntax highlighting of the code</td>
<td>✓</td>
</tr>
<tr>
<td>Collapse and expand code lines</td>
<td></td>
</tr>
<tr>
<td>Small explanation for each code line</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.3: Ray tracing visualization requirements

<table>
<thead>
<tr>
<th>Ray tracing visualization</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show/hide viewport plane</td>
<td>✓</td>
</tr>
<tr>
<td>Choose the current pixel from the viewport</td>
<td>✓</td>
</tr>
<tr>
<td>For each step of the algorithm, visualize the step in the scene</td>
<td>✓</td>
</tr>
<tr>
<td>Change objects’ material properties</td>
<td></td>
</tr>
<tr>
<td>Navigate through the visualization</td>
<td>✓</td>
</tr>
<tr>
<td>Focus on the visualization, following the ray</td>
<td></td>
</tr>
<tr>
<td>Different starting scenes</td>
<td></td>
</tr>
<tr>
<td>Add/remove objects to the scene</td>
<td>✓</td>
</tr>
<tr>
<td>Change lights’ color</td>
<td></td>
</tr>
<tr>
<td>Read scenes from files</td>
<td></td>
</tr>
<tr>
<td>Save the current state of the application</td>
<td></td>
</tr>
</tbody>
</table>

The engine manages the input and output from the VR headset and the controllers, facilitating the application development. The architecture of RayTracerVR is depicted in Figure 4.1. The user interface is composed of the modules **Panels** and **Scene**.

The **Panels** module manages the several panels available to the user. The content of these panels is further described in more detail in Section 4.4. The code, help, and variables panels only display information, they do not alter algorithm-specific input, so they receive the step data information from the manager to represent it accordingly for each panel. Additionally, the object context panel only shows information relative to the scene object the user is pointing to with their right controller. Contrarily, the playback panel can control the algorithm’s execution, and the settings panel can change other scene and algorithm properties; thus, they send this input to the manager.

The **Scene** module handles the display of the scene where the ray tracing algorithm will be executed. This scene is composed by the viewport grid and by the target scene. For the visualization of the algorithm, we will only run and show the visualization for one pixel at a time. Then the user needs to
choose the pixel in the viewport grid so that the algorithm can be calculated by the manager for that specific pixel. The target scene only needs to receive the step visualizations to represent in the scene.

Finally, the Manager is responsible for executing the ray tracing algorithm and managing the execution of the algorithm. With the necessary inputs, it creates “steps” that will compose the whole algorithm. Each step will contain the necessary information to be provided to the other modules to visualize the algorithm.

4.3 Algorithm Coding

The algorithm data displayed to the user is stored in a static structure in the project code. For each step in the ray tracing’s algorithm, this structure contains an identifiable name, the actual code step string, a textual description of that step in English, and a list of steps inside its context (Figure 4.2). For example, the first step function rayTracing(scene, ray, depth) will have all the following steps inside this function as children. With this structure, we can quickly get the list of pseudo-code steps with getCode, the list of descriptions for each step with GetHelp, and get the index of a step with its RTStepName with the function GetStepIndex.

The global manager will be responsible for managing the application’s state, including creating the list of objects representing each step in the application. When the user starts the algorithm, the global manager will call the function RayTracing, which will execute the ray tracing algorithm and, simultaneously, create the step objects that are saved on the list of steps. These objects will have the necessary data to update the whole visualization: the necessary step to be highlighted in the code panel, the names and values of the variables in the context of the step for the variables panel, the description for the current
step for the help panel and other necessary visualization updates for the target scene, such as drawing rays or highlighting objects, among others.

The application flow can be better visualized in Figure 4.3. Before starting the algorithm’s execution, the user must select a pixel in the viewport, which will be initialized with black color. Then, the manager will initialize the current step index. Now, the user needs to decide if they will want to run the algorithm in step-by-step mode or not. The flowchart divides the two flows, running in step-by-step mode or not, but in reality, the user can change modes during the execution. This division was done to simplify the chart.

For the continuous mode, the opposite of the step-by-step mode, the user needs the click on the “Play” button available in the playback panel, which will cause the current step index to increment. If we have not reached the final step, we can run the step. Running a step implies updating the panels and target scene with the necessary information:

- highlighting the current step in the code panel
- showing the description of the current step in the help panel
- showing or updating the variables of the current step in the variables panel
- and updating the target scene with the visualization of the current step

When the duration of this step finishes, the current step index will be automatically increased, and the next step will be executed. Suppose at any time the user wishes to pause the execution. In that case, they need to click on the “Pause” button, and the current time elapsed of the step will be paused until the user wants to continue the algorithm’s execution, clicking again on the “Play” button. The algorithm execution will be finished once it has reached the last step.

The flow for the step-by-step mode is very similar, where the big difference is that the duration of the step will be dismissed and will only change steps if user input is provided. To execute the next step, the user needs to click on the “Next” button on the playback panel for the current step index to be increased. If, on the other hand, the user wishes to execute the previous step again, they need to click on the “Back”
button, which will decrease the current step index. The execution of a step is in every way the same as in continuous mode, where it needs to update the other visualization components.

4.4 User interface

In a VR project, the application's user interface is one of the most relevant parts of developing a good user experience. Without it, the application's functionality would not be useful if the user does not know how to interact with it. The following section discusses the initially planned user interface and describes the end prototype's look and functionality.

4.4.1 Mockup

At the start of this work, we created a mockup (Figure 4.4) of how the immersive environment and the interface would be, which functioned as a guideline for developing the prototype. It was not tested with
end-users but was discussed among the development team.

On the left side is featured a panel with playback buttons and some other possible buttons. On the right, a panel is represented with the code and description of each step. As we will see in the following section, we maintained the concept of panels but subdivided them according to their content.

In the center of the mockup, the target scene is shown where the algorithm’s animation will take place, along with the viewport (the grid-like object) and the rays created during the animation. The final interface stayed very similar to this.

### 4.4.2 World environment

The user is located in a futuristic room where they can interact with the ray tracing visualization scene and the panels, which display information and allow user input. The user can dislocate around the scene with teleportation - with a controller, they point to the floor and clicks on the grab button to move to that point. The user can only teleport in the available floor area, so they cannot go beyond the bounds of the room.

### 4.4.3 Scene

As described in the architecture section, the scene is composed of the viewport grid and the target scene, which will be described in the following sections.
Viewport grid

The viewport grid represents the pixels that compose the image plane (Figure 4.6). When the user hovers each pixel with the controller, the background color will change to highlight it and show a small label with the coordinates of that pixel.

Before starting the algorithm's execution, the user can choose a pixel by pointing the controller to it and clicking on the trigger button. The ray tracing algorithm will create the primary ray for this given pixel. The background color of the pixel will update at each step that this color is determined so that at the end of the execution, the pixel will contain the final color determined by the algorithm. During the algorithm's execution, the user can no longer change the pixel, only after the state is reset.

Target scene

The target scene is where the ray tracing algorithm representation will occur. Figure 4.7 shows that the default scene is a Cornel Box with spheres inside. These objects have different materials and colors to offer some varied ray tracing executions depending on the objects the primary ray would intersect.

This scene executes different visualization types at each code step change to represent the step. Steps that are simple changes of variables do not affect the target scene. The types of visualization steps are as follows:

- **draw ray**: the principal visualization in this application is seeing the rays created in the algorithm represented visually by a line that draws from its origin to a defined final point. If there is still no knowledge of the ray’s end in the current step, it will draw a finite, although long, length. If the endpoint of the ray is already determined, when we have already determined the intersection with the object in the scene, the ray will only be drawn from the origin to the specified endpoint.
• **change ray color:** the current ray that is being used in the algorithm will have a lighter color than the other rays. When we finish using a ray for the algorithm’s primary function, we will change its color to be darker to avoid giving as much importance.

• **hide ray:** especially for the representation of vector normals. When we change the context of the primary ray used, we hide the vector normals before the following recursion to simultaneously limit the number of visual objects in the scene and simplify the subsequent visualization of the recursion.

• **highlight object:** when iterating through the objects in the scene, the object currently being iterated through will have a yellow outline color. If the object has an intersection with the ray, the outline will change to blue, and if the object contains the closest intersection saved at the moment, then the outline will be green.

• **hit point sphere:** at each determined hit point, we draw a small blue sphere in the location of the hit point to better highlight in the scene.

• **change object opacity:** specifically for objects with a dielectric material, when the refracted ray traverses the inside of the object, it is essential that the user can see the ray, so the opacity of the object is decreased to allow seeing its inside.
4.4.4 Panels

A couple of panels are available for the user that shows essential information or allows the user to change specific scene parameters. The primary panels with information, the code, help, and variables panels their basic behavior are as follows:

• on controller hover, a small border is shown to indicate to the user that they can perform some command using the controller’s buttons

• on clicking the grab button of the controller, the user can position the panel in the space around them. While clicking the button:

  – moving the controller will move the panel always at the same distance from the user and rotate automatically so that it always points to the user
  – moving the thumbstick vertically will change the distance of the panel to the user. Moving up will increase this distance, and moving down will make the panel closer to the user
  – moving the thumbstick horizontally will change the scale of the panel. Moving to the right will increase the panel’s size, while moving to the left will decrease its size

However, the playback panel is an exception. We considered that this panel should have different behavior than the others since the user will need to interact with it more frequently. This panel will always stay in front of the user at a specific height from the ground. If the user changes their position or rotates their head, the panel will always stay in the same position, at approximately the height of their torso. This way, the user will readily have the playback buttons to press with the controller, with their arm pointing downwards. This will be a more comfortable position for the user, considering they can interact with the application for long periods.
Regarding the implementation of these panels, the displayed data comes from the application's global manager, which is responsible for updating every panel at each step change.

**Code Panel**

This panel shows the full ray tracing pseudo code (Figure 4.8). The code has a simple syntax highlight and is correctly indented. During the algorithm's execution, the algorithm's current step is highlighted. Since the complete code can be extended, the screen can be scrolled up and down with the controllers. Nevertheless, during the algorithm's execution, the panel will automatically scroll to show the current line of the code.

At the start of the application, this panel receives the code text from the static algorithm data and formats it to be displayed correctly. It adds the necessary indentation taking into account the hierarchy of the steps. Also, simple syntax highlighting is applied.

The syntax highlight is done using regular expressions, in other words, rules that capture certain patterns. Some specific keywords were captured in different colors, such as `function`, `if`, `for`... Some variables and function calls were also captured to be in a different color.

At each code step change, the global manager will give this panel the current code step index, updating its visual accordingly. With the previous index stored, it will remove the highlight from that step and add the highlight for the current one. The scroll position will also change to ensure that the current step is in view to the user.

![Figure 4.8: Code panel](image)
Variables Panel

While the algorithm is executing, the variables used in the code will be displayed in this panel (Figure 4.9). The values of the variables will be updated automatically at each step. If the user goes to the previous steps, the values of the variables will show the value they had in previous lines of code. Hence, the value is relative to the current context of the algorithm’s execution. The variables that have changed their value in the current step will be highlighted with a brighter color.

The global manager is responsible for providing the variables’ names and values to this panel. When this global manager first creates the algorithm’s steps data for the whole application, it updates a structure that contains the variables’ names and values. Each step has associated a copy of this structure at the time of this step; in other words, each step has associated the structure of the variables in the context of the step so that when the user changes the step (to the next or the previous one), this variables panel will always show the values at the time of the step.

Help Panel

This panel shows a short textual description, in English, of each step in the algorithm, as shown in Figure 4.10. Its purpose is to give more context to the line of code to help the user better comprehend the algorithm.

Each code step has associated with a specific description. At the start of the application, this panel stores the descriptions of all the steps in a list. This way, at each code step change, the global manager gives this panel the index of the current step and changes the text displayed to the user to the one from the new index.
As previously mentioned, this panel will remain static relative to the user to facilitate its access. This panel will contain the buttons relevant to the algorithm’s execution (play, pause, next, and back), a slider that allows the user to change the speed of the execution, and a reset button, to reinitialize the algorithm’s state as seen in Figure 4.11. The play and pause buttons will respectively run and pause the algorithm’s execution with a specific duration for each step. The next and back buttons will allow for step-by-step execution of the algorithm. In the first step, the back button will be disabled, and in the last step, the next button will be disabled. The user can interact with these buttons by pointing to the buttons with the controller and pressing the trigger button.
Settings panel

The settings panel contains some buttons and sliders that change some parameters of the scene. It is also connected to the user’s left controller for easy access (Figure 4.12).

The left and right buttons allow the user to rotate the scene in front of them. While the user is clicking on the button, the scene continuously rotates. The other three sliders change some grid properties that represent the view plane and pixels. The grid opacity slider will change the grid's opacity to make it more transparent or opaque. The grid resolution sliders change the number of cells representing the grid's pixels in the x and y axes. After the user starts the algorithm's execution, they will be unable to change the resolution sliders so they will be disabled. The last shorter slider toggles the visibility of the camera model that is visible in the scene.

Object context panel

The object context panel is only visible when the user points to an object in the target scene with the right controller. When visible, the panel appears attached to the top of the right controller (Figure 4.13).

The information on this panel relates to the specific object the user is pointing to. It shows the object's name, the type of material (diffuse, metallic, or dielectric), and the material's color. For dielectric materials, it also shows the index of refraction.
Figure 4.13: Object context panel

- Name: Sphere 1
- Color: Orange
- Type: Metal
# Evaluation

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Presentation and Preliminary Feedback</td>
<td>45</td>
</tr>
<tr>
<td>5.2 Evaluation</td>
<td>46</td>
</tr>
<tr>
<td>5.3 Discussion</td>
<td>52</td>
</tr>
</tbody>
</table>
During the development of RayTracerVR, we presented the prototype's current state to target users, students who had a basic knowledge of ray tracing, to receive initial feedback and adjust certain decisions that would affect the final product of the application. This presentation and feedback are further described in Section 5.1.

With the final prototype, we wanted to evaluate the effectiveness of RayTracerVR in teaching ray tracing. The evaluation methodology and results collected from these testing sessions are detailed in Section 5.2.

5.1 Presentation and Preliminary Feedback

In order to get initial feedback to RayTracerVR, a presentation of the current state of the prototype was made to an audience of Computer Engineering Master students enrolled in “3D Programming”. One of this subject’s objectives is ray tracing.

This presentation gave context to this work’s objectives and prototype with the help of slides and a video recording of the prototype’s environment and interaction. After the presentation, the students had the opportunity to ask some questions regarding the project and answer a questionnaire to provide some feedback on the prototype (Appendix A). With this questionnaire, we wanted to ascertain certain aspects regarding the interface and environment of the project. Since the students could not interact with the application itself, we could only request feedback on the project’s visuals. We got 13 answers in total.

One of the questions concerned the chosen environment for the application. We wanted to verify if the virtual environment developed, a sci-fi-looking room, was adequate to the context and activities in the application or if a simpler “darkroom” would be more appropriate. According to the answers gathered, 61.5% of the students preferred the sci-fi room.

In regards to the panels, they were all classified as being useful or very useful, in which the variables panel received more strongly positive answers to its usefulness. Additionally, in terms of the features listed, users classified all of them as positive usefulness, with an almost unanimous strongly positive answer to choosing the pixel in the viewport. Finally, there were no negative answers (“Strongly Disagree” or “Disagree”) to the intuitiveness of the algorithm visualization shown in the video.

We did this initial presentation to provide helpful feedback on the project’s current state and adjust our route in the following weeks to develop the final prototype. It was also a way to inquire about possible users for future testing sessions, although there was not much enrollment.
5.2 Evaluation

In this section, we detail the formal evaluation performed for this work. A group of students answered a theoretical test before and after interacting with the application. With the results from these testing sessions, we aim to prove our hypothesis that using RayTracerVR can facilitate the understanding of ray tracing. Thus, we tested the following null hypothesis:

\[ H_0: \text{using RayTracerVR is equal to or worse than using traditional methods when it comes to learning the ray tracing algorithm} \]

Considering this, we performed the testing sessions with the methodology described in Section 5.2.1, and the analysis of the results gathered are detailed in Section 5.2.2.

5.2.1 Methodology

The user tests were performed in INESC-ID\(^1\) in a room reserved exclusively for executing this project's tests from July 4th to July 8th. During the performance of each test, only the user and the observer were present in the room, resulting in a quiet environment.

During the testing session, the users sat at a desk with a computer where they filled in the questionnaires. For the interaction part of the session, the users had a space of about one meter by one and a half meters where they could stand and move their arms around to interact with the VR application. Both moments are depicted in Figure 5.1. Figure 5.2 shows the user’s view of RayTracerVR using the VR headset.

\[\text{Figure 5.1: Users during the testing session: answering the questionnaire (left) and interacting with the VR application (right)}\]

\(^1\text{https://www.inesc-id.pt/}\)
The computer used to perform the tests had an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz processor, with 16GB of RAM and an NVIDIA GeForce GTX 1060 3GB graphics card. A monitor, keyboard, and mouse were also present. The VR equipment used was an Oculus Rift Headset with two controllers and two sensors, which detect the position of the headset and controllers.

The user testing script and the consent form in Portuguese are included in Appendix B. All the users performed the same testing session. First, they were welcomed to the room and given some initial context to what RayTracerVR was about and how the testing session would proceed. They were then offered to sign a consent form for the use of information, videos, and photos that would be gathered during the session. Afterward, they could start answering the user testing questionnaire (Appendix C), divided into sections. First, the user replied to the sections relative to user characterization, virtual reality, and the theoretical test regarding ray tracing. After finishing these sections, the user was ready to interact with RayTracerVR. With the VR equipment adjusted, each user was asked to perform the same set of tasks inside RayTracerVR that would provide both feedback for the usability of the application and require the users to analyze and comprehend the ray tracing algorithm. When finishing the last task, the users could resume the questionnaire, answering the ray tracing theoretical test again and the last two sections - System Usability Survey (SUS) and specific questions regarding the usability of RayTracerVR.

To measure the effectiveness of our project, we examined the test score of the theoretical test before the interaction with the application and after, which could give an idea if the user learned the algorithm better after interacting with the application.
5.2.2 Results

The tests were performed by 16 users following the methodology described in the previous section. Next, we will analyze these results.

5.2.2.A User characterization

Considering that our topic of learning ray tracing is directed at people studying Computer Science and Computer Graphics subjects, we chose students at Instituto Superior Técnico that were enrolled in the Computer Science course, be it at a Bachelor’s or a Master’s level. For this reason, all the users were in the age group of 18 to 24, which is the usual university student age.

The objective of the evaluation is to assess if the users gained additional knowledge of the ray tracing algorithm after interacting with our application, so we asked the users to answer the affirmation “You are familiar with the ray tracing algorithm” on a scale of 1 to 5, whereas 1 means “Strongly Disagree” and 5 “Strongly Agree”.

5.2.2.B Theoretical Test

Initially, we had planned to perform an evaluation with control groups. However, since the user tests were performed right at the end of the semester, it was harder to find available students to perform the tests, so we thought that an intergroup evaluation study would not be possible with many users. For this reason, we performed the same test plan with all the users and chose students with at least an

![Figure 5.3: User’s familiarity with the ray tracing algorithm](chart)
initial approach to ray tracing. Then, we asked the users to answer the theoretical test before and after interacting with RayTracerVR and learning more about ray tracing.

The questions gathered for the test were based on actual questions from exams on the subject of 3D Programming. Most of the questions were multiple choice, a couple of ones had more than one correct answer, and only one question was of short answer. Each question was scored with the number of correct options, i.e., if a question had two correct answers, it had a score of 2. We chose to score the questions this way instead of weighting each one the same to favor answers users had correctly, even if the question was not entirely correct. The test had ten questions with a total of 13 points.

In Figure 5.4, we can see the user test scores on the first and second times they took the test. Each line represents a user, and the colors represent the comparison between the two scores. The area below the horizontal dashed line represents a failing grade.

As we can see in the chart, most users increased their scores the second time they answered the test. Only one user decreased their score, and the other three maintained the same score. The average user score also increased from 6.44 to 8.81 out of 13, which is equivalent, on a scale of 100, from 49.52 to 67.79. This increase shows that the average test score went from a failing to a passing score. It is important to note that even though most users increased their scores the second time, some still maintained a failing grade, and no user had a full score. This can be explained by the difficulty of some of the questions in the theoretical test.

The bar chart in Figure 5.5 shows the percentage of correctly answered questions for each question.
Figure 5.5: Comparison of correctly answered questions on each theoretical test

in the test on the first and second time the users answered. We can see that each question had an increase of users that answered correctly, and we can note that question 5 was the most difficult since, even on the second try, the percentage of correct answers was minimal compared to the other questions.

The distribution of test results can be compared in Figure 5.4. As shown, on the second try, almost 75% of the users achieved a passing score, while on the first try, only 50% achieved that. It is also of note that the range of results did not increase on the second try, but the minimum and maximum scores did, and there are no outliers.

Since we performed the same theoretical test on users twice, we can apply the Paired Sample T-Test to analyze the statistical difference between the two tests’ averaged scores and evaluate our null hypotheses. The Paired Sample T-Test requires that the sample has a normal distribution; however, with the limited results collected (16), it is hard to prove the normality, so we will assume it for this test. We found a statistical difference \( p = 0.0005 \) between the two groups, which means that the test passed for \( p < 0.05 \).

With this result, we can reject the null hypothesis proposed that "using RayTracerVR is equal to or worse than using traditional methods when it comes to learning the ray tracing algorithm", and say the opposite - using RayTracerVR can benefit users in learning ray tracing more than traditional ways of learning.
5.2.2.C Usability

At the interaction moment of the testing session, users were asked to perform a set of tasks with Ray-TracerVR to evaluate the application’s usability and allow the users to learn the algorithm.

The users performed a preliminary task with the intent of them to get familiar with the application before the actual tasks. The first task was planned to compel the users to interact with the panels to change their position and size. As expected, the users only spent around a minute modifying the panels, yet there are a couple of outliers. Regarding the usability of the controls, users responded that they were reasonably easy to manage. The second task was created so that the users could start the algorithm and get a general view of the visual representations, not supposed to view each step attentively. Afterward, the third task required the users to reset the current state of the algorithm, which they found rather quickly, taking less than a minute to do so. Finally, the last task was designed so that the users could spend some time reading and analyzing each step of the algorithm to be able to answer the theoretical test afterward. Here we could note some variations among the users: some used all the panels (code, help, and variables) in accompaniment with the central scene, some never used the variables panel, and some only looked at the code and help panels, occasionally glancing to the central scene.

To evaluate the usability of RayTracerVR, we asked the users to answer the standard questions from the System Usability Survey (SUS) after all the tasks. Calculating the SUS score for each user’s answer, we get an average of 78.8 with a standard deviation of 12.8. Considering the score grading for SUS scores, our average score fits in the interval of B+ (77.2 - 78.8). Also, looking at the boxplot in Figure

![Boxplot and scatter chart](image)

**Figure 5.6:** User’s SUS score: boxplot (left) and scatter chart with correlation to time interacting with the application (right)
5.6, more than 75% of the results are above the SUS average of 68. Additionally, there is a positive correlation between the SUS score users attributed to the application and the time they spent interacting with it. We can then assume that with the increased time interacting with RayTracerVR, they perceive a better usability experience.

5.2.2.D Problems and features

In the test questionnaire, users were asked to write what they liked the most and the least in the application and suggestions for future features. Likewise, some problems users faced during their interaction that they did not point out in the questionnaire were identified. These problems were consolidated along with possible solutions in Table 5.1. The problems were classified in severity on a scale of 1 - 4, where one would be a minor issue and four a grave usability issue. The solutions were classified in terms of cost per person/month - how many months would a one-person team take to implement the solution.

Some of these solutions have a small cost, so they were implemented in the project to fix these usability issues. Some solutions would bring significant value to the project, such as skipping loops and functions; however, the cost of implementing these would be greater than the time we had left to conclude the application. Nonetheless, these possible solutions are documented here for possible implementation in future work of this project.

5.3 Discussion

We got preliminary feedback from the target users and polled possible candidates for user testing. Even though the number of users did not allow for intergroup evaluation, we noted an increase in acquired knowledge in the users after interacting with RayTracerVR, as shown by the results in the theoretical test.

The final pool of tested users could also have been larger for more accurate results, yet we believe we had enough users to apply the Paired Sample T-Test. With the results from this test, we could reject our null hypothesis and affirm that RayTracerVR can facilitate users in understanding the ray tracing algorithm.

Additionally, we tested the usability of the application. Some usability issues were pointed out, yet the consensus was that RayTracerVR provided a positive usability experience, getting an average SUS score of 78.8. From the users’ interaction with RayTracerVR, we noticed that some panels or more used than others, emphasizing the variables panel, which was hardly ever used, only by users interested in examining the actual values of the variables.

We collected the issues users faced during the interaction, the problems they mentioned in the questionnaire, and suggestions they gave for future features. The solutions for these were outlined; some
Table 5.1: Problems and solutions identified for RayTracerVR, along with a severity scale (1 - 4), cost in person-/month, and solved status

<table>
<thead>
<tr>
<th>Problems and Solutions</th>
<th>Severity</th>
<th>Cost</th>
<th>Solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiresome to go through each repeated step and functions</td>
<td>3</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>- Skip loops (and repeated loops)</td>
<td></td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>- Skip functions (and repeated functions)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>There is no way to identify the rays after they are created</td>
<td>2</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>- Show ray information on hover, like for the objects in the scene</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controls to scroll are not very intuitive</td>
<td>2</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>- Update the controls’ instructions for scrolling</td>
<td></td>
<td>0.2</td>
<td>✓</td>
</tr>
<tr>
<td>The initial scene has too many objects</td>
<td>4</td>
<td>0.1</td>
<td>✓</td>
</tr>
<tr>
<td>- Remove some objects from the starting scene</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The target scene does not allow customization</td>
<td>3</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>- Allow moving/scaling/rotating objects in the scene</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unable to add own notes inside the application</td>
<td>1</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>- Enable note-taking associated with the current state of the algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous movement would be preferred instead of teleporting</td>
<td>2</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>- Allow changing mode of movement (teleport and continuous)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unable to rotate the scene with the camera fixed</td>
<td>1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>- Allow rotating the scene with the camera fixed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unable to choose a ray that will hit a specific object</td>
<td>2</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>- Preview primary rays before selecting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unable to change the algorithm’s variables</td>
<td>3</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>- Allow changing algorithm variables</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The algorithm shown does not have many ray tracing effects</td>
<td>3</td>
<td>6.0</td>
<td></td>
</tr>
<tr>
<td>- Extend the algorithm with other effects (e.g., Distributed Ray Tracing)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The current environment is not very relaxing</td>
<td>1</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>- Use a more relaxing environment (e.g., green landscape, beach)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The camera object gets in the way when looking at the scene</td>
<td>2</td>
<td>0.5</td>
<td>✓</td>
</tr>
<tr>
<td>- Allow toggling camera visibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The playback panel moving around the user is confusing</td>
<td>2</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>- Allow toggling playback panel fixed relative to user or not</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The viewport grid opacity too strong during algorithm execution</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- After choosing the first pixel, change the viewport opacity automatically</td>
<td></td>
<td>0.2</td>
<td>✓</td>
</tr>
<tr>
<td>The reflected and refracted rays’ colors are too similar</td>
<td>2</td>
<td>0.1</td>
<td>✓</td>
</tr>
<tr>
<td>- Change the reflected and refracted rays’ colors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The playback and settings panels are too big</td>
<td>1</td>
<td>0.2</td>
<td>✓</td>
</tr>
<tr>
<td>- Decrease the size of the playback and settings panels</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
were addressed in this work, and others will be left for future work.

All in all, the results gathered from the evaluation of this work were positive. We discovered some issues with the application; nevertheless, the users showed a good learning experience.
Conclusion
Work related to educational simulations with VR has been increasing in recent years. Immersing the user in a virtual space and allowing the user to interact directly with the subject can aid in learning those concepts better.

We analyzed previous work related to VR in educational applications and saw a lack of work directed at teaching algorithms. Additionally, we summarized a set of applications specific to teaching algorithms. For this work, we focused on a single, although complex, algorithm: ray tracing. In the planning stages, we discussed a set of requirements that would be relevant for this project. Of these requirements, we implemented a group of those dimmed integral to the essential use of the application.

To make sure that the project is publicly available, a simple website was created to hold a homepage for RayTracerVR. It shows a brief description of the application accompanied by a general image of the environment. More importantly, from this page, it is possible to download the application, execute it locally, and also download the source code.

![RayTracerVR website](https://web.tecnico.ulisboa.pt/lidiagcustodio/)

This website is available at [https://web.tecnico.ulisboa.pt/lidiagcustodio/](https://web.tecnico.ulisboa.pt/lidiagcustodio/).

We tested our project with a group of 16 users, students who have been presented with this algorithm before with more traditional learning methods. We had the users interact with the VR application and answer a theoretical test before and after to assess their gained knowledge after the interaction. We applied a statistical analysis to these results, and they were positive, showing that the grade in the theoretical test after interacting with the application was increased.

Moreover, the users answered a standard usability questionnaire, SUS, and the results showed that RayTracerVR has an average usability score of 78.8, above the SUS’s average result of 68. We also noted the issues users faced with the application and gathered the suggestions they gave for possible improvements and features. These were compiled in a list classified by the severity of usability and cost.
of implementing the solutions. Due to time constraints, only the lowest-cost solutions were implemented.

As previously stated, not all of the requirements set at the beginning of the project were implemented, as well as improvements suggested and noted after the user testing sessions. Due to time constraints, these can be addressed in future work of this project. The requirements not implemented can be reviewed in Tables 4.1, 4.2 and 4.3 and the improvements suggested after the evaluation are summarized in Table 5.1. Nevertheless, the following items summarize possible suggestions for future works:

**Enhance playback control functionality.** The playback control of the application can be improved to support other features commonly used in Integrated Development Environments (IDE), such as stepping into and out of functions/iterations and breakpoint functionality - adding/removing breakpoints, viewing a list of breakpoints, run to breakpoint, among others.

**Change algorithm variables.** At the start of the algorithm’s execution, there are a couple of constant variables whose values could be changed by the user, as well as variables created during the algorithm’s runtime.

**Target scene customization.** The target scene could be further improved if it allowed customization from the user, for example, adding/removing objects from the scene, changing position, size or rotation, and changing the object’s material properties.

**Extend ray tracing algorithm.** The ray tracing algorithm in the application could be further extended to add other effects, using, for example, distributed ray tracing.

In conclusion, we consider that RayTracerVR, in its current state, would provide a good asset in teaching ray tracing. The improvements suggested and further work to this prototype would increase its value. Furthermore, the evaluation conducted confirmed our initial hypothesis - virtual reality can improve the understanding of ray tracing.
Bibliography


Preliminary Questionnaire
Hi!

My name is Lídia Custódio and I'm working on my Master Thesis with the subject "Ray Tracing Explorer in VR".

As you've seen from the presentation, I would like to receive some feedback to the project that I have developed. It will take no more than 5 minutes and I would greatly appreciate it.

If you would like to participate in user testing you can leave your contact email in the last section of the form.

Thank you!

**Required**

**Environment & UI**

1. Which room did you prefer? *
   - Sci-fi room
   - Dark room

2. You found each panel to be very useful. *
   - Code Panel
   - Variables Panel
   - Help Panel
   - Playback Panel
   - Object Context Panel
   - Settings Panel

<table>
<thead>
<tr>
<th></th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables Panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Help Panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Playback Panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Context Panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Settings Panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. You found these features to be very useful

<table>
<thead>
<tr>
<th>Feature</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neutral</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changing the resolution of the viewport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Changing the opacity of the viewport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choosing the pixel in the viewport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moving the panels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Changing the size of the panels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rotating the scene with buttons</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Any comments/suggestions you would like to add about the environment and UI

_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________

Ray Tracing Visualization

5. You found the visualization of the algorithm intuitive

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

6. Any comments/suggestions you would like to add about the ray tracing visualization

_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________

User Testing

7. Would you like to be contacted for final user testing of the project?

- Yes
- No

8. If you answered "yes", please leave your email

_____________________________________________________________________________________________

Thank you for your time!
User Testing Script
Guião Testes com Utilizadores

Material

- Material VR
  - Oculus Rift HMD
  - Oculus Rift Controllers
  - Oculus Rift Sensors
- Carregador de pilhas (para os controllers)
- Computador (para questionário e interação)
- Ecrã
- Rato
- Teclado
- Álcool e papel (para limpeza do material)

Procedimento (32 MIN)

Preparação do Teste (3 MIN)
Antes do utilizador entrar na sala:

1. Limpar material VR com álcool e papel (1 min)
2. Abrir questionário no browser para o utilizador (1 min)
3. RayTracerVR aberto no Unity para permitir alterar valores durante a execução e repor valores iniciais (1 min)

Introdução (3 MIN)
O utilizador é recebido na sala com todo o material necessário na mesa. O utilizador senta-se e eu digo:

1. “Bom dia, o meu nome é Lídia Custódio e estou a trabalhar na tese de mestrado com o tema Ray Tracing Explorer in VR”
2. “No âmbito deste projeto, desenvolvemos uma aplicação em VR – RayTracer VR – cujo objetivo é representar o algoritmo de ray tracing de modo a o utilizador ter uma melhor aprendizagem sobre o mesmo”
3. “Antes de interagir com a aplicação VR, vou pedir para responder a um breve questionário com algumas perguntas teóricas sobre ray tracing”
4. “De seguida, poderá interagir livremente com a aplicação e depois irei pedir para fazer algumas tarefas”
5. “No fim da interação com a aplicação vou pedir para continuar a responder ao questionário”
6. “O questionário será anónimo, e os seus dados serão apenas utilizados no âmbito deste projeto”
7. “Imagens e vídeos poderão ser recolhidos, mas serão apenas usados no contexto deste trabalho”
8. “Aqui tem o formulário de consentimento. Depois de ler e assinar poderemos continuar com o teste”

Entregar formulário de consentimento e esperar que o utilizador o leia e assine (1 MIN)

Questionário Pré-utilização (2 MIN)
O utilizador responde às secções:

1. Caracterização do utilizador
2. Virtual reality
3. Teórica pré-teste
Preparação da utilização (3 MIN)
1. Mostrar vídeo (1 min)
2. Explicar nomes dos botões dos comandos (1 min)
3. Ajustar HMD ao utilizador, dar controllers e utilizador coloca-se de pé (1 min)

tarefas (15 MIN)
0. Interação livre com a aplicação (3 min)
1. Mover e posicionar painéis no modo mais confortável para o utilizador (2 min)
   a. Métricas:
      i. Posição final dos painéis
      ii. Necessidade de alteração (utilizador moveu os painéis, ou não)
2. Seleccionar o pixel (7, 5), mudar tempo de cada step para 0.5s e iniciar algoritmo (3 min)
   a. Pixel (7, 5) para um objeto com material DIFFUSE
   b. Métricas:
      i. Tempo
      ii. Erros
      iii. Vezes que o utilizador se teleporta
      iv. Vezes que o utilizador pausa e reinicia a execução
3. Reiniciar o estado do algoritmo (1 min)
   a. Métricas:
      i. Tempo
      ii. Erros
4. Seleccionar o pixel (9, 6), mudar tempo de cada step para 3s e iniciar visualização do algoritmo no modo step-by-step, observando a representação do algoritmo, juntamente com o código e lendo a descrição textual de cada passo em voz alta (exceto os repetidos) (7 min)
   a. Pixel (9, 6) para um objeto com material DIELECTRIC
   b. Métricas:
      i. Tempo
      ii. Erros
      iii. Vezes que o utilizador anda para o step anterior
      iv. Vezes que o utilizador usa os botões do painel vs botões do comando
      v. Vezes que o utilizador se teleporta

Pós Utilização (1 MIN)
1. Retirar equipamento do utilizador, e poderá sentar-se novamente (1 min)

Questionário (5 MIN)
O utilizador responde às secções: (5 min)
   1. Teórica pós-teste
   2. Usabilidade
RayTracerVR – Testes com Utilizadores: 
Termo de Consentimento e Acordo de 
Registos Áudio e Vídeo

TERMO DE CONSENTIMENTO

Eu, (por favor escreva o seu nome) consinto em participar de livre e espontânea vontade neste teste de usabilidade sob a direção do Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa (INESC-ID).

Compreendendo, que a minha participação é totalmente voluntária e em qualquer momento da sessão poderei retirar o meu consentimento, suspendendo a minha participação, sem qualquer tipo de penalização ou prejuízo para mim.

Foi-me dado o direito de colocar perguntas sobre os procedimentos utilizados neste estudo de forma que esses procedimentos sejam explicados de uma forma satisfatória para mim.

ACORDO DE REGISTOS ÁUDIO E VÍDEO

As gravação áudio e vídeo realizadas durante este estudo, serão utilizadas para investigação e desenvolvimento desta aplicação de realidade virtual. Fui informado de que os investigadores e estudantes do INESC-ID assistirão às minhas gravações e que algumas imagens poderão ser usadas em publicações académicas para documentar estes testes.

Concedo o meu consentimento, ao INESC-ID para usarem as gravações de áudio e vídeo para:

☐ Efeitos de estudo
☐ Publicações académicas

Assinatura: _________________________________

Data: _________________________________
User Testing Questionnaire
RayTracerVR User Test

Hi!

My name is Lídia Custódio and I’m working on my master’s thesis with the subject "Ray Tracing Explorer in VR". In the context of this subject, I developed an educational VR application called "RayTracerVR" and now I need your help to test it.

I will ask you to answer this questionnaire as well as interact with the VR application. To participate, you only need a base knowledge of the ray tracing algorithm.

The whole testing session will take 35 - 45 minutes. All data collected (information, images or videos) will be anonymous and only used in the context of this project.

Thank you for your participation!

Required *

1. User ID: ______________________________________

User Characterization

2. How old are you? *
   - 18 - 24
   - 25 - 30
   - 31 - 40
   - More than 40

3. What is your gender? *
   - Male
   - Female
   - Prefer not to say
   - Outra: ______________________________________

4. You are familiar with the Ray Tracing algorithm. *

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

5. Select the LEIC/MEIC courses you took that mentioned ray tracing topics: *
   - Computer Graphics (Computação Gráfica, LEIC)
   - 3D Programming (Programação 3D, MEIC)
   - None of the above
   - Outra: ______________________________________

Virtual Reality

6. How many times have you used Virtual Reality (VR) equipment? *
   - 0
   - 1 - 10
   - 11 - 50
   - 50+

7. Have you ever thought about purchasing VR equipment? *
   - Yes, I own VR equipment
   - Yes, but haven’t bought it
   - No, I have not

8. Do you think VR could help you in learning computer graphic topics? *

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

(adapted from Google Forms)
Theoretical Test (before interaction)

All the questions are relative to the algorithm shown in the application, which is Whitted Ray Tracer.

9. Which type(s) of material(s) can generate secondary shadow rays? *
   - Diffuse
   - Metal
   - Dielectric

10. Which type(s) of material(s) can generate secondary reflected rays? *
    - Diffuse
    - Metal
    - Dielectric

11. Which type(s) of material(s) can generate secondary refracted rays? *
    - Diffuse
    - Metal
    - Dielectric

12. Consider a scene with 1 million diffuse objects and 3 point lights. At maximum, how many secondary rays will be generated with the Whitted Ray Tracer algorithm for rendering a 800x600 image resolution? *
    - 800 * 600
    - 3 * 800 * 600
    - 1 000 000 * 800 * 600
    - 3 * 1 000 000 * 800 * 600

13. How many shadow rays are missing from the diagram? *
    
    Your answer: ____________________________

14. What type of material is sphere B? *
    - Diffuse
    - Metal
    - Dielectric

15. What type of material is sphere C? *
    - Diffuse
    - Metal
    - Dielectric
16. What is the relation between the angles created by rays 4 (α) and 5 (β), relative to the normal at the intersection point? *

- $α = 2β$
- $2α = β$
- $α = β$
- $α = \frac{1}{β}$

17. What is the relation between the angles created by rays 4 (α) and 6 (β), relative to the normal at the intersection point? *

- $n_1 = \sin α$
- $n_2 = \sin β$
- $n_1 = \sin β$
- $n_2 = \sin α$
- $n_1 = \cos α$
- $n_2 = \cos β$
- $n_1 = \cos β$
- $n_2 = \cos α$

18. What is the name of the relation you used previously? *
- Schlick's Approximation
- Snell's Law
- Fresnel Equations

Now interact with the application!

Theoretical Test (after interaction)

All the questions are relative to the algorithm shown in the application, which is Whitted Ray Tracer.

19. Which type(s) of material(s) can generate secondary shadow rays? *
- Diffuse
- Metal
- Dieletric

20. Which type(s) of material(s) can generate secondary reflected rays? *
- Diffuse
- Metal
- Dieletric

21. Which type(s) of material(s) can generate secondary refracted rays? *
- Diffuse
- Metal
- Dieletric

22. Consider a scene with 1 million diffuse objects and 3 point lights. At maximum, how many secondary rays will be generated with the Whitted Ray Tracer algorithm for rendering a 800x600 image resolution? *
- $800 \times 600$
- $3 \times 800 \times 600$
- $1\ 000\ 000 \times 800 \times 600$
- $3 \times 1\ 000\ 000 \times 800 \times 600$

23. How many shadow rays are missing from the diagram? *
24. What type of material is sphere B? *
- Diffuse
- Metal
- Dielectric

25. What type of material is sphere C? *
- Diffuse
- Metal
- Dielectric

26. What is the relation between the angles created by rays 4 (α) and 5 (β), relative to the normal at the intersection point? *
- $\alpha = 2\beta$
- $2\alpha = \beta$
- $\alpha = \beta$
- $\alpha = \frac{1}{\beta}$

27. What is the relation between the angles created by rays 4 (α) and 6 (β), relative to the normal at the intersection point? *
- $n_1 \sin \alpha = n_2 \sin \beta$
- $n_1 \sin \beta = n_2 \sin \alpha$
- $n_1 \cos \alpha = n_2 \cos \beta$
- $n_1 \cos \beta = n_2 \cos \alpha$

28. What is the name of the relation you used previously? *
- Schlick’s Approximation
- Snell’s Law
- Fresnel Equations
### System Usability Survey

<table>
<thead>
<tr>
<th>Question</th>
<th>Rating Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>29. You would like to use this application more often. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>30. You think the application is more complicated than it should be. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>31. You think the application is simple and easy to use. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>32. You need help to use this application. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>33. You think the application's features are well integrated. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>34. You think there was too much inconsistency in this application. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>35. You think that people would learn to use this application very quickly. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>36. You found the application very cumbersome to use. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>37. You feel confident using this application. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>38. You think there are a lot of things to learn before you can start using this application. *</td>
<td>Strongly Disagree</td>
</tr>
</tbody>
</table>

### RayTracerVR

<table>
<thead>
<tr>
<th>Question</th>
<th>Rating Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>39. How would you rate your physical discomfort after interacting with the application? (motion sickness, head discomfort...) *</td>
<td>Not at all</td>
</tr>
<tr>
<td>40. The controls to interact with the panels were easy to manage. *</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>41. The UI for the playback panel was clear. *</td>
<td></td>
</tr>
</tbody>
</table>
42. The help panel showed relevant information.

43. The code panel showed the algorithm in a clear manner.

44. The variables panel was very useful.

45. The different colored rays were distinguishable between each other.

46. What did you like the most in the application?
47. What did you like the least in the application?

_____________________________________________________________________________________________
___________________________________________________________________________________________
_____________________________________________________________________________________________

48. What features do you suggest should be implemented in the future?

_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________

49. Any general comments regarding the application.

_____________________________________________________________________________________________
_____________________________________________________________________________________________
_____________________________________________________________________________________________

Thank you for your participation!