Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Rikard Storheil Eriksen

Visualizing the Principles of Ray Tracing Algorithms in Virtual Reality

Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2022

NTNU
Norwegian University of
Science and Technology

Rikard Storheil Eriksen

Visualizing the Principles of Ray Tracing Algorithms in Virtual Reality

Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Since the advent of modern virtual reality headsets in 2016, VR has seen success and opportunities within teaching and visualizing, and continues to increase in popularity with standalone VR devices like the Meta (formerly Oculus) Quest 2. Like virtual reality the idea of ray tracing is not a novel one, but it only recently became feasible to use it in practice, and it is widely used in the movie industry for producing high quality realistic renders. Ray tracing is also seeing real-time applications today with technology like the Nvidia RTX graphics cards that feature hardware support for important ray tracing processes.

Concepts within ray tracing can be difficult to learn from a book. This thesis explored how ray tracing could be taught by utilizing VR. The PathVis application was developed for the SteamVR platform using the Unity game engine, and lets the user trace rays and paths interactively in VR by using a ray gun. The application is structured as a zig-zagging hallway that introduces concepts one by one until reaching a gallery room where the user has access to a virtual camera that can render images with a simple path tracing rendering algorithm. The rendering algorithm supports diffuse, reflective, refractive and emissive materials, but the algorithm is simplistic and does not perform branching or shoot shadow rays. The user is required to perform simple tasks to proceed through the hallway, starting with a tutorial that teaches the user how to navigate and interact with the application. Concepts are introduced via "booths" along the walls and the learning material is presented as text as well as narrated audio.

User testing was conducted on two versions of the application to evaluate the usability and whether users enjoyed this type of learning. Users felt that ray tracing was a good use case for a VR application and enjoyed using the interactive ray gun. The PathVis application can act as an introduction to ray tracing, and be used as a supplement to other learning methods.

Two playthroughs of the application were recorded for demonstration purposes, without narration (6:46): `https://youtu.be/ho7HpW9-PHs` and with narration (15:20): `https://youtu.be/a7GscTKB0hw`

Sammendrag

I årene etter at moderne virtual reality headsets kom på markedet i 2016 så har VR blitt brukt til læringsspill og visualisering med gode resultater, og populariteten til VR fortsetter å vokse med enkeltstående enheter som Meta (tidligere Oculus) Quest 2. Raytracing er på samme måte som VR ikke et nytt konsept, men det tok en stund før raytracing kunne kjøres raskt nok til å kunne brukes i praksis. I dag brukes raytracing mye i filmindustrien for å produsere realistiske bilder med 3D-grafikk. Ny teknologi har også gjort det mulig å raytrace i sanntid, som f.eks. ved bruk av grafikkort innen Nvidias RTX-serie som har maskinvare for å kjøre viktige deler av prosessen innen raytracing.

Konsepter innen raytracing kan være vanskelig å lære seg fra en bok. Denne masteroppgaven utforsket hvordan raytracing kan bli undervist ved å ta i bruk VR. Applikasjonen PathVis ble utviklet ved bruk av Unity og kjører på SteamVR-plattformen, og lar en bruker eksperimentere med raytracing og path tracing interaktivt ved å skyte individuelle rays og paths fra en pistol (ray gun). Applikasjonen er strukturert som en korridor som går i sikk-sakk, og konsepter innen raytracing blir introdusert gradvis til brukeren ender opp i et større galleri-rom, hvor brukeren har tilgang til et virtuelt kamera som kan rendre bilder ved bruk av en enkel path tracing-algoritme. Algoritmen har støtte for diffuse (matte), reflektive, refraktive og lysende materialer. Brukeren må utføre en rekke enkle oppgaver for å kunne gå videre i korridoren, og blir først lært hvordan å bevege seg rundt i VR og hvordan objekter kan plukkes opp. Konseptene bak raytracing blir introdusert i "båser" langs veggene, og læringsmateriellet blir presentert gjennom både tekst og tale.

Brukertesting ble gjennomført på to versjoner av PathVis for å evaluere brukbarheten og om brukere liker denne formen for læring. Brukerne føte at raytracing var et godt bruksområde for VR, og likte interaktiviteten som pistolen (ray gun) gav dem. PathVis-applikasjonen kan fungere som en introduksjon til raytracing generelt, og kan brukes som et tillegg til andre læringsformer.

To videoer ble laget for å demonstrere applikasjonen, både uten tale (6:46): `https://youtu.be/ho7HpW9-PHs` og med engelsk tale (15:20): `https://youtu.be/a7GscTKB0hw`

Contents

Figures

Acronyms

3DoF Three Degrees of Freedom. 6

6DoF Six Degrees of Freedom. 6, 16, 64

BVH Bounding Volume Hierarchy. 66

GPU Graphics Processing Unit. 69

HMD Head-Mounted Display. 5, 6, 13, 15, 25–27, 51, 52, 61, 64, 69, 72

IOR Index of Refraction. 12, 33, 35, 67

SDK Software Development Kit. 15

SUS System Usability Scale. 26

Chapter 1

Introduction

Since the release of the Virtual Reality headsets Oculus Rift and HTC Vive in 2016, VR has seen a steady growth in adoption and software applications over the years in addition to hardware improvements. Virtual reality lets the user experience a 3D environment from within instead of looking at a flat 2D projection on a screen. This fact makes VR a great place to visualize things that are three-dimensional in nature, with ray tracing being an example.

1.1 Motivation

Ray tracing became a household term after the introduction of Nvidias RTX line of graphics cards, with many current and upcoming games utilizing ray tracing for some of their graphical effects[1]. To simulate light behavior by tracing rays into a three-dimensional scene is not a novel concept, and ray tracing as we know it today can be traced back to the 1980s[2][3][4]. It took a while before ray tracing caught on because of its heavy computational needs, but as hardware became more performant and ray tracers more efficient and sophisticated, ray tracing started seeing use in the movie industry for rendering animated movies and visual effects[5]. Today ray tracing is also being used in real-time applications like video games for tasks that traditional rasterization techniques are inferior at, like reflections and shadows[6][7].

Because ray tracing took so long to become performant, the TDT4230 - Graphics and Visualization course left ray tracing out of the curriculum for some time, but has since been added back due to the comeback of ray tracing. Learning the inner workings of ray tracing by reading from a textbook can be difficult, so the goal of this thesis is to implement an application that can present to a user how ray tracing works by putting users with no prior knowledge of the topic in a virtual 3D environment where they can inspect ray tracing at their own pace and gain an intuition for how it works and how it could be implemented in practice.

1.2 Goal and Research Questions

The objective of this master thesis is to implement a 3D application that lets a user learn about ray tracing and path tracing by interactively tracing individual rays as well as rendering ray traced images, all within an immersive virtual reality environment with full positional tracking. Three research question were defined for the thesis:

- **RQ1:** How can virtual reality be used for teaching ray tracing?
- **RQ2:** Can a ray tracing rendering algorithm be visualized in virtual reality?
- **RQ3:** What do users think about learning in virtual reality compared to traditional learning methods?

1.3 Contributions

There are a few examples of VR educational applications aimed at teaching computer science. At the Norwegian University of Science and Technology there have been previous master thesis projects in which VR educational application was developed and tested. One such application explored how VR could be used to teach sorting algorithms in an algorithm and data structures course [8]. Another example is an application that teaches concepts within artificial intelligence [9]. Both projects were met with interest from the test users, but that the use of VR did not always give a clear benefit in terms of learning.

Computer graphics topics tend to be visual in nature, but some concepts can be hard to grasp when presented via text or static images. In this project, an application was developed that allows for interactive ray tracing and path tracing in a virtual environment. The application can introduce and visualize ray and path tracing to users in VR. The application does not require extensive knowledge of VR, and was tested by users who had a varying amount of existing experience with VR. The application runs on any SteamVR-compatible VR headset such as the HTC Vive, Meta (formerly Oculus) Quest 2, Valve Index and Windows Mixed Reality. The Unity project for the application is available in a public repository on Github [10]. Links to demonstration videos and a download of the final build can be found in Section 5.1.

1.4 Thesis Structure

This report is structured as follows: Chapter Chapter 2 gives an overview of VR and its use in teaching, followed by background theory for ray tracing and a description of the technology used. Chapter Chapter 3 describes the methods used for developing the PathVis application. The design and implementation of the PathVis application is described in detail in Chapter 4. Chapter 5 presents the results of

the user tests and Chapter 6 discusses the results and potential improvements that can be made.

Chapter 2

Background and Related Work

2.1 Overview

This section introduces and gives a brief history of virtual reality and instructional games, followed by a description of the fundamentals of ray tracing. At the end is a section on the technology used to implement the application, and related work regarding the use of virtual reality for teaching.

2.2 Theory

2.2.1 Virtual Reality

The idea of entering a virtual reality powered by machines dates back to the 1960s, with the Sensorama being the first implementation of such a system [11][12]. The Sensorama showed a stereoscopic film accompanied by stereo sound, scent, wind and vibration to engage as many senses of the viewer as possible, but it was not an interactive system. The first Head-Mounted Display (HMD) was constructed by Ivan Sutherland around the same time, which tracked the position and orientation of the user's head and could draw primitive wireframe graphics [11][12]. HMDs became the most viable implementation of virtual reality, though there were other approaches like CAVE that projected stereoscopic images on the walls of a room [11]. Virtual reality saw a lot of excitement in the 1980s and 1990s but the limitations of computer hardware at the time lead to disappointment, and the popularity of virtual reality quickly faded.

VR saw a comeback in 2012 with the Kickstarter project for the Oculus Rift, as computer graphics had become powerful enough to deliver a much better experience, and the HMD could be made very affordable [13]. The first development version of the Oculus Rift (DK1) provided rotational tracking and a stereoscopic view. The second development version, Oculus Rift DK2 added positional tracking utilizing an external camera sensor and infrared diodes on the outside of the HMD[14]. Positional tracking adds another level of immersion, as the movement

of the user in physical space corresponds to movement in the virtual space. HMDs with only rotational tracking are referred to as having Three Degrees of Freedom (3DoF), whereas positional and rotational tracking combined is referred to as Six Degrees of Freedom (6DoF).

In 2015, the HTC Vive headset was revealed, featuring a different approach for positional tracking that utilizes special sensors on the HMD that keep track of laser pulses emitted by external base stations. This tracking system opened for a much larger tracked area and introduced the notion of room-scale VR [15]. As gamepad controllers are not feasible for walking around in VR, the Vive has wireless motion controllers for each hand that are also tracked in VR space.

Both the Oculus Rift and the HTC Vive were released to the consumer market in 2016. The Oculus rift got its own motion controllers shortly after release. In the years that followed, many new PC VR headsets have appeared on the market or been announced, like the Windows Mixed Reality line of headsets, Pimax, Oculus Rift S, HTC Vive Pro and Valve Index [16].

In recent years, standalone VR has become a popular option for VR, mainly because of accessibility and affordability of the Meta Quest 2 (formerly Oculus Quest 2), which offers positional tracking through the use of onboard cameras, removing the need for external sensors or base stations as well as the need for a powerful PC. The fidelity of VR applications running on standalone VR systems is lower than those running from a PC, as the hardware has to be small and light enough to fit within the headset itself. For better fidelity, the Quest 2 can be used as a PC VR headset by connecting it to a PC with a cable or wireless link.

As the virtual reality HMDs primarily display graphics, they can be categorized by what system powers the graphics:

- Personal Computers (PCVR): These VR headsets are connected to a computer that processes the tracking of the VR hardware and renders the images to be shown in the headset. HTC's Vive line of headsets and the Valve Index are examples of these. 6DoF tracking can be achieved either through external sensors or onboard cameras. PCVR headsets are typically tethered to a PC with a cable that provides the video signal and power, but some headsets like the Vive can also use a wireless adapter. The adapter is then responsible for transmitting the video stream wirelessly as well as powering the headset from a battery.
- Play-Station VR (PSVR): Instead of a PC, the PSVR connects to Play-Station gaming consoles.
- Phone VR: Mobile phones are inserted into these headsets to act as both the display and processing system. Examples of Phone VR are Google Cardboard and Samsung Gear VR. These typically only offer 3DoF tracking.

- Standalone VR: These headsets have the processing system built into the headset itself. Meta Quest 2 is an example of this, though it can also connect to a PC to act as a PC VR headset. Connecting to a PC can be done either through a cable or wirelessly.
- Cloud based: Similar to wirelessly streaming from a PC, it is also possible to stream from a remote server in the cloud and to an untethered or tethered thin client. Nvidia's CloudXR system is an example of this, which currently supports both tethered PCVR headsets and standalone Android-based headsets.

2.2.2 Instructional Computer Games

As computer technology became more affordable, it seemed inevitable that computers would become a staple in homes as well as in schools. In 1980 Malone [17] postulated that computers could be utilized in educational settings through the use of instructional games. Examples of instructional games will be presented in Section 2.4. For instructional games to be a success they need to be enjoyable, and so Malone emphasized three characteristics of good computer games: *challenge*, *fantasy* and *curiosity*.

The challenge in a computer game can facilitate learning, by requiring the player to learn a skill in order to reach a goal. Goals should be clear and compelling, and can be assisted by adding an element of fantasy like a story that the player can progress through as they learn new skills. Feedback is also important, as players must know when they are getting closer to the goal, and when they are not. The element of curiosity motivates learning independently from the goals or the fantasy, and curiosity can be distinguished into two categories: sensory and cognitive. Sensory curiosity stimulates the senses of the player, which can be both auditory and visual. Sound and visuals can be used to enhance the fantasy element, or to provide a reward for reaching a goal. Cognitive curiosity arises from the need of having knowledge be consistent and complete, and as such the cognitive curiosity of a player can be engaged by presenting information that is new to the player, or that somehow clashes with their existing knowledge.

2.2.3 Ray Tracing

$$\mathbf{P}(t) = \mathbf{A} + t\mathbf{b} \tag{2.1}$$

Ray tracing is a method of rendering 2D images from 3D scenes by tracing "rays" for each pixel of the image until they hit surfaces in the scene (Figure 2.1). Additional rays can then be traced towards lights and other surfaces, enabling effects like reflection, refraction and shadows to be simulated [2].

A ray can be modeled as a line in 3D space with an origin $\mathbf{A}$ and a direction vector $\mathbf{b}$. Equation 2.1 then defines a point along the ray using the parameter t.

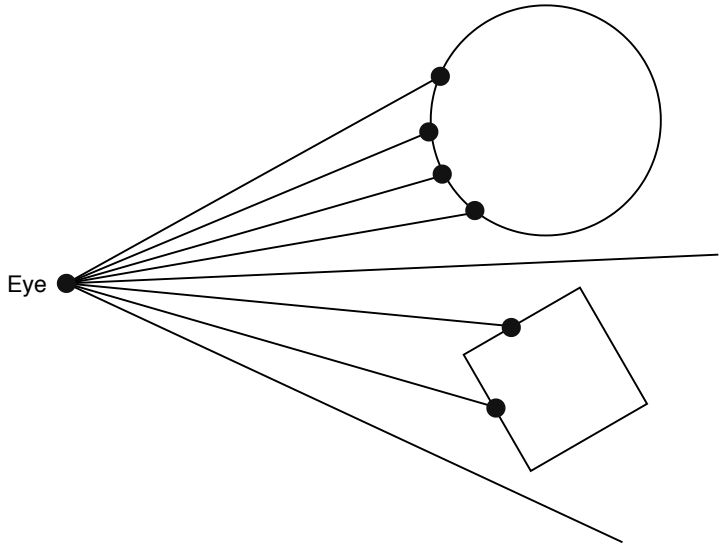


Figure 2.1: Tracing rays from the eye into a scene

A ray intersects a surface if its surface equation and the ray equation 2.1 share a point. Ray tracing can thus render parametric surfaces like spheres or planes without needing to first make a mesh consisting of triangle primitives[18]. In practice however, triangular meshes are preferred as they can be used to model any 3D surface and the ray-triangle intersection routine can be made very efficient[19].

Distributed Ray Tracing

Early ray tracing was limited to rendering sharp shadows, sharp reflections and sharp refraction, as lights were modeled as singular points and rays intersecting a point on a reflective surface were always reflected in the same direction. To get over this hurdle, Cook et al. [3] introduced distributed ray tracing in 1984, which distributes rays in various dimensions to capture fuzzy phenomena like motion blur, glossy reflections, blurred transparency, depth of field and penumbras (soft shadows).

The need for distributing rays arises from the complexity of the shading function (Equation (2.2)), as described in [3]:

> The intensity I of the reflected light at a point on a surface is an integral over the hemisphere above the surface of an illumination function L and a reflection function R[1].
>
> $$I(\phi_r, \theta_r) = \int_{\phi_i} \int_{\theta_i} L(\phi_i, \theta_i) R(\phi_i, \theta_i, \phi_r, \theta_r) d\phi_i d\theta_i \qquad (2.2)$$
>
> where
> (ϕ_i, θ_i) is the angle of incidence, and

(ϕ_r, θ_r) is the angle of reflection.

Early ray tracing made assumptions that simplified the reflectance and illumination functions, which in turn made it only possible to produce sharp reflections, refractions and shadows. Distributed ray tracing solves this by shooting rays in directions according to the distribution of the illumination, reflection and refraction functions.

2.2.4 Path Tracing

Since all rendering algorithms try to simulate how light scatters between surfaces, a generalized rendering equation (Equation (2.3)) was introduced in 1986 by Kaiyja and James [4]:

The rendering equation is

$$I(x,x') = g(x,x')[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''] \quad (2.3)$$

where:
$I(x,x')$ is related to the intensity of light
passing from point x' to point x'
$g(x,x')$ is a "geometry" term
$\epsilon(x,x')$ is related to the intensity of emitted light
from x' to x $\rho(x,x',x'')$ is related to the intensity of light scattered
from x'' to x by a patch of surface at x'

This equation does not attempt to model all optical phenomena but is instead a geometrical optics approximation, modeling the emission, transport and scattering of light. Path tracing is a method that gives numerical solutions to this rendering equation. As a monte carlo method, path tracing converges towards an approximate result by tracing paths of rays from the eye and into the scene. Many samples are taken by tracing multiple paths for each pixel, and the results for all the paths are averaged together into one value for each pixel. Higher number of samples taken give less noisy images.

Motion Blur

Motion blur is the phenomenon of fast moving objects appearing blurred or smeared in an image. In photography this happens when objects move during the time at which the camera shutter is open. Distributed ray tracing makes motion blur possible by introducing the concept of time, and randomly distributing rays across a period of time. Each ray checks for intersections with the scene as it appears in a single instant of time. For this to be possible, the representation of the scene needs to be defined across a range of time, so rays know where all the objects in

the scene are at the time of their occurrence [3]. As results are averaged together, areas in an image that had moving objects will show a blend between the objects and the background.

Surface Normals

The surface normal is the vector that is normal (perpendicular) to a surface. It is used extensively within ray tracing, and defines the direction the surface faces. Unless stated otherwise, the normal vector is treated as a unit (normalized) vector pointing to the outside of the surface.

Texture Coordinates

To be able to map textures onto 3D geometry, a 3D surface needs to have texture coordinates defined across its surface. In triangular meshes, texture coordinates are defined at each vertex and interpolated across each triangle. Texture coordinates are commonly wrapped to the $[0, 1]$ range, and referred to as UVs or UV coordinates, as the XYZ coordinates are used for the position of a vertex.

2.2.5 Materials

Surfaces in the real world have many different types of appearances; bright surfaces reflect more light than dark ones, glass will refract light, lamps will emit light. When rendering surfaces of different types, it is useful to use the concept of a material. The amount and color of the light that is reflected from a surface depends on the properties of its material and the incident light ray.

Diffuse Materials

Diffuse materials model matte surfaces that scatter incident light randomly, independent on the direction of the incident light ray. A perfectly diffuse material is said to have Lambertian reflectance, and reflects incident light in all directions equally [20]. One way to simulate Lambertian reflection is to randomly choose the reflected light direction by picking a random point on the unit sphere offset by the surface normal [18] as shown in Figure 2.2.

Reflective Materials

Reflective materials scatter light more prominently in some directions than others. Perfectly reflective surfaces like mirrors always scatter the light in the direction of the incident light ray reflected across the surface normal vector, as seen in Figure 2.3. Fuzzy reflections can be achieved by adding a small random component to the reflected vector [18].

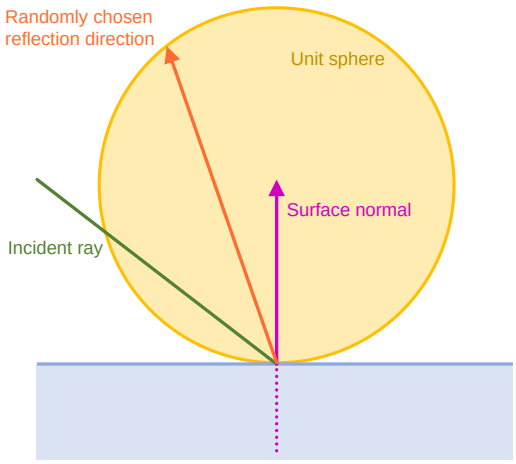


Figure 2.2: Choosing a direction to send the reflected ray.

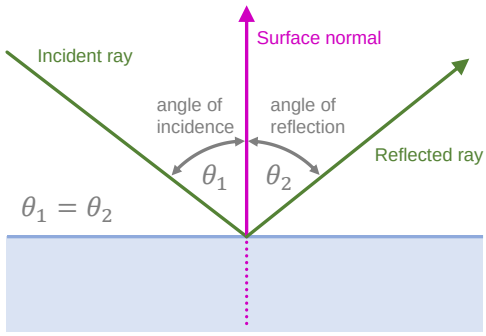


Figure 2.3: Reflection across the surface normal.

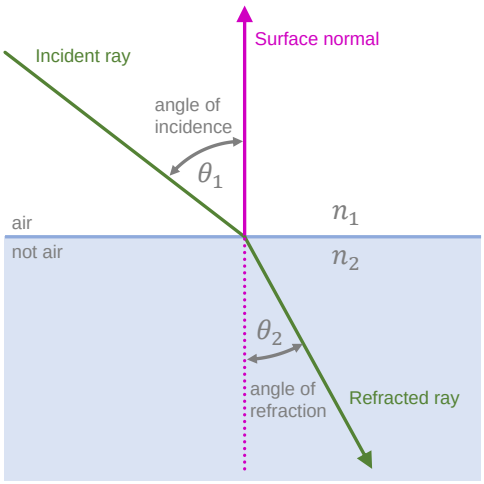


Figure 2.4: Refraction of a light ray entering a material.

Refractive Materials

Refraction is when a ray of light changes direction when passing into a different material, as shown in Figure 2.4. The incident ray can enter the material, depending on the angle of incidence - the angle between the incident ray and the surface normal. In reality these materials both refract and reflect, e.g. glass and water, with more light being reflected as the angle of incidence increases. To simulate refraction, each material is given an Index of Refraction (IOR) n (Equation 2.4) which specifies the speed of light c in vacuum divided by the speed of light in the material v.

In practice empty air is treated as having an IOR of 1.0, with common materials like water having an IOR of 1.333 [21]. The direction of refracted light can then be calculated using Snell's Law (Equation 2.5) [18].

$$n = \frac{c}{v} \tag{2.4}$$

$$\frac{\sin\theta_2}{\sin\theta_1} = \frac{n_1}{n_2} \tag{2.5}$$

Bounding Volume Hierarchies

Every traced ray needs to test for intersections with the scene. When scenes contain many primitives, it is too expensive for each ray to test if it intersects with each primitive. One solution is to first test if rays intersect with a bounding volume before testing for ray-primitive intersections. If the ray does not intersect with the bounding volume, it doesn't intersect with any of the primitives contained within,

and all the expensive ray-primitive tests are avoided. A common approach is to use a hierarchy of axis-aligned bounding boxes [22], as the more accurately the bounding volume represents the group of primitives, the fewer ray-primitive tests are wasted.

2.2.6 Ray and Path tracing vs Rasterization

The power of ray and path tracing lies in its simplicity and elegance compared to the traditional computer graphics approach. The traditional approach is to feed a 3D scene through a pipeline that performs transformations, clipping, projection, hidden surface elimination and finally shading and texturing [22]. This rasterization based technique for rendering has significant limitations, notably reflections, indirect lighting and shadows [7]. In comparison, ray and path tracing has no issues dealing with reflections, indirect light and shadows, and the simplest path tracer needs only a well defined 3D scene with methods to test for intersections to be able to produce realistic images [18].

The downside of ray and path tracing is the computational burden - testing for intersections and gathering enough samples to reduce noise is expensive. Path tracing has become the method of choice in the movie industry for its accuracy and flexibility, and where render time is less important due to massive render farms [5]. Recent advancements in hardware is also making real-time ray and path tracing possible, like the Nvidia Turing architecture which features hardware acceleration of bounding box and ray-triangle intersection tests [6][7].

2.3 Technology

2.3.1 Game Engine Chosen

To develop an application that can be viewed in a VR HMD, a game engine is typically used. Game engines have built-in systems that handle all the core functionality that makes a game run; graphics, input handling, audio playback, asset loading and more. With a game engine a developer is able to focus on creating the functionality that is specific to their own game or application. For VR development, there are two common choices of game engine: Unity and Unreal Engine. Unity is easy to pick up and works well for independent developers, whereas Unreal Engine is more suitable for larger teams and projects. As Unity was chosen for this project, there will not be a description of Unreal Engine.

Unity is a game engine that supports development of 2D, 3D, VR and AR software on many different platforms. It is popular among independent developers as the engine is free to use to develop and publish a game with, if the revenue from the game is below a certain threshold. Learning resources are numerous thanks

to its popularity, with many tutorials and courses available for many topics within game development.

In Unity, development takes place in the editor (Figure 2.5), which consists of panels that serve different purposes. The *Scene Panel* acts as a viewport and shows the currently opened *scene*, with *scenes* being the context in which objects exist. Objects can have *Components* attached to them depending on their role, and the components appear in the *Inspector Panel* when an object is selected. The *Hierarchy Panel* gives an outline of the scene, showing the names of all the objects in a hierarchy depending on their parent-child relationships. The *Project Panel* is a file browser that gives access to the assets available within the project, and assets can be dragged into the scene from here.

The role of an object depends on the components attached to it: for an object to be visible it needs a *Renderer* component, which has a pointer to a 3D mesh. Physics is supported by having objects with *Collider* components. Unity comes with many components that can be added to objects, but it is also possible to create your own components via scripting. When a script component is attached to an object, the developer can write code in the high-level programming language C# which interfaces with Unity's Scripting API.

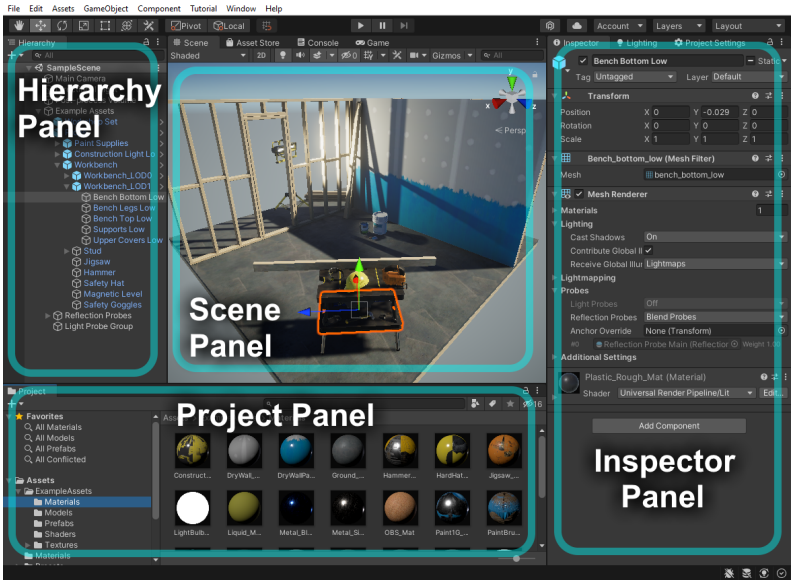


Figure 2.5: Screenshot of the Unity Editor

2.3.2 VR Software

To make an application able to interface with VR devices, a software layer referred to as a runtime is used. Tracking data from the hardware is relayed to the applica-

tion to update the positions of the virtual camera and controller models. Frames are rendered by the application and displayed in the VR HMD. The runtime is also responsible for providing a basic environment outside of the VR application, including the ability to display a virtual boundary at the edge of the defined play space. Software Development Kits (SDKs) are used to develop applications for specific VR runtimes.

When the HTC Vive and Oculus Rift launched in 2016, there were two main choices for VR developers. The Oculus SDK could be used to develop applications for the Oculus platform, and the OpenVR SDK integrates with the SteamVR platform. The Oculus platform only officially supported Oculus devices like the Rift and later Rift S, whereas the SteamVR platform could be used with most PCVR devices including the HTC Vive, Oculus Rift and later also the Windows Mixed Reality devices. The SteamVR platform was chosen for this project because of its ability to run with any PCVR device.

SteamVR Unity Plugin

The SteamVR Unity Plugin from Valve is used to let Unity applications smoothly interface with SteamVR [23]. The plugin is responsible for loading 3D models for the correct VR controllers and handling the input from them. An interaction system is included to enable developers to quickly get their application usable within VR. To make a scene usable with SteamVR, a camera rig or *Player* prefab is placed within a scene. Many other example prefabs and scenes are also included to demonstrate how the interaction system can be used in practice. When developing an application with this plugin, the result is a portable executable file that can be run on any PC that has the SteamVR runtime installed.

XR Interaction Toolkit

The XR Interaction Toolkit is Unity's own framework for developing VR and AR applications [24]. It consists of components that support many different interaction tasks crucial for VR, including controller input handling, object grabbing, canvas UI interaction and a VR camera rig for room-scale VR.

Many things have changed in the VR software landscape since the project of this thesis began. When development of the prototype started, the XR interaction toolkit was not compatible with SteamVR, and so the SteamVR plugin was chosen for developing interactions for the application.

2.4 Related Work

Virtual Reality in Education

Today, there are many existing educational VR applications available on popular VR platforms such as the Oculus Store and Steam. A market analysis [25] of such VR educational applications from 2019-2021 provides an overview of the most popular areas for VR applications, finding that nature, space, medicine, art and history to be the most prominently featured topics.

The content within educational VR applications is typically presented either as 360° video or 3D models. Using 3D models and environments makes it possible to give applications a higher degree of interactivity compared to 360° video, as well as increased immersion by the user being able to move around in the virtual environment with 6DoF. The immersion and presence offered by VR opens up new possibilities for teaching complex topics, especially those that pertain to shapes and spatial ability [26]. Complex datasets can also be visualized in a more manageable way with VR, like tracing neurons in microscope scans of the brain [27] or displaying geophysical data [28]. All of these applications benefit directly from the extra dimension that virtual reality gives, but VR can also help with data that isn't necessarily three-dimensional in nature, like teaching sorting algorithms [29][8].

Many studies have explored the use of VR for learning in comparison to less immersive technology. A review of 29 such studies found that in most cases the use immersive virtual reality offered a benefit for the learning outcome, particularly for teaching highly complex or conceptual problems that require spatial understanding and visualization[30]. In some studies, the immersive VR did not offer any significant benefits over more traditional technology like a 2D video. The novelty of VR within educational contexts mean that learning outcomes may be hindered by users' unfamiliarity with the technology, and needs to be considered when comparing outcomes with other learning methods.

VirtSort

One example of a VR application that teaches a concept within computer science is the VirtSort application by Kong and Kruke [8], which puts students into a virtual environment where they can learn three different sorting algorithms by interacting with boxes on a table. The application was tested with students and showed that VR could be used as an effective tool for learning, though some users new to VR had trouble interacting with the application in the intended way.

Virtual Reality for AI Education

Another application that utilizes virtual reality for teaching is the one developed by Sølve [9], which teaches concepts within artificial intelligence. The learning material is presented within an escape room environment, and the user has to progress through the rooms by solving 3D-puzzles, doing calculations and quizzes. The testing of this application indicated that users appreciated 3D visualizations of difficult concepts, as well as "learn-by-doing".

Chapter 3

Methodology

3.1 Overview

This chapter describes the research and development methods used to develop interactive ray tracing in Unity, and how the user tests were carried out.

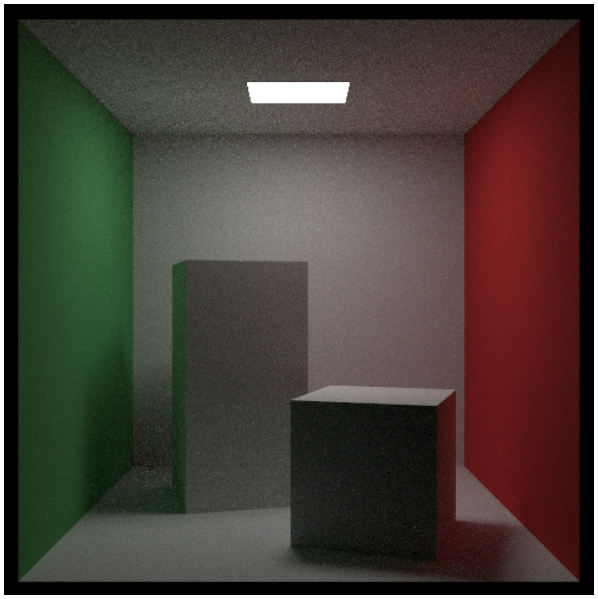
3.2 Research Methodology

To teach concepts of ray tracing to potential users, it was first decided that a firm understanding of ray tracing would be needed, and the chosen method to achieve this was to implement a standalone ray tracing application, and then evaluate which of the concepts could be visualized in VR. The VR application was then to be developed using an iterative development process, with user testing being used to gain feedback and make improvements.

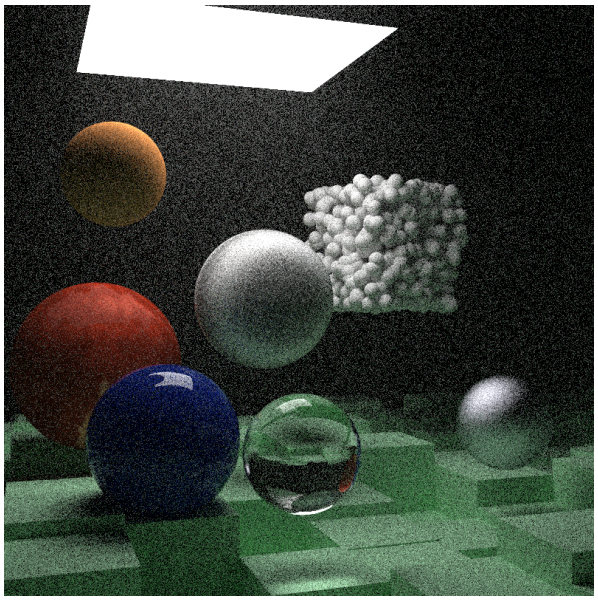
3.2.1 Standalone Ray Tracer

A standalone ray tracing application was implemented in C++ based on the material from the three books Ray Tracing in One Weekend[18], Ray Tracing: The Next Week[31] and Ray Tracing: The Rest of Your Life[32].

The ray tracer features diffuse, metallic, glass, emissive and volumetric materials, as well as motion blur, texture mapping and depth of field. The renderer supports rendering spheres, planes and boxes by using ray-sphere and ray-plane intersection equations. To improve performance, the OpenMP API was used to make the rendering routine multithreaded. Figure 3.1 shows two example renderings produced by the standalone ray tracer.



(a) The Cornell Box



(b) Refractive, reflective, diffuse, emissive and volumetric materials

Figure 3.1: Example renders from the standalone ray tracer

The standalone ray tracer implements path tracing in a simple brute force fashion. For each pixel in an image, rays are traced from the camera and into the scene. Upon hitting a surface, new rays are traced in the direction of the reflection as dictated by the material of the surface that was hit. This continues until the max number of reflections per path is reached. As path tracing is a numeric solution to the shading equation, rendering continues to trace paths for every pixel over and over, and averages the results to give the final image. Because diffuse reflections are essentially random, small light sources will have a low probability of being hit, and the result is a lot of noise at low sample counts. An easy solution to this is to make light sources large.

Due to the simplicity and compelling results given by the path tracing algorithm in the standalone ray tracer, it was decided that the VR application developed for this thesis should be themed around path tracing. Features from the standalone ray tracer were implemented as C# scripts in Unity, with interactive ray tracing functionality made possible by the use of Raycasts offered by Unity's physics subsystem, as presented in Section 4.4.2.

3.2.2 Iterative Development

Being a solo project, there was less need for a strict development process, but the overall development would follow the Waterfall Model:

1. Requirements: a number of requirements were defined as to what the application should let the user do.
2. Design: The elements of the application were designed and planned out.
3. Implementation: The features were implemented in the application and tested on their own.
4. Testing: The overall application was to be tested by users.

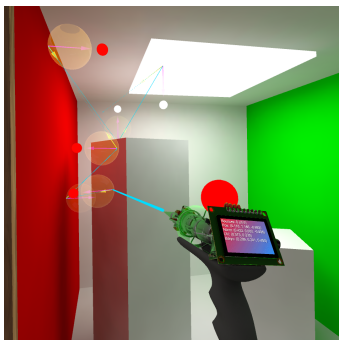
The chosen game engine made rapid prototyping possible: 3D scenes were assembled and tested in VR to confirm that basic functionality was in place. Additional features were developed and tested one by one, until a final design of the environment and the functionality was reached. This final functionality would then go on to be tested by users.

3.3 Stages of Development

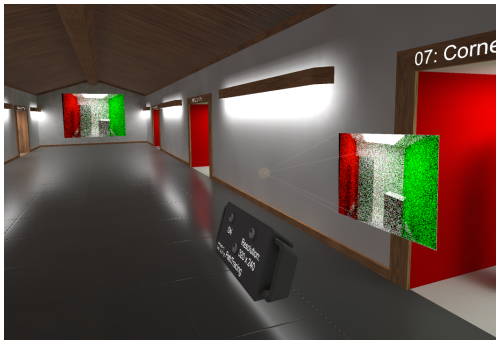
The development of the application was split into two major stages, with the second stage further being divided into two smaller parts for two user tests.

3.3.1 Prototype

The first stage of the project was to develop a prototype of the application that would let users see the path tracing algorithm visualized in VR. The prototype ap-



(a) The prototype ray gun visualizing a traced path



(b) The ray tracing camera and booths of the prototype

Figure 3.2: The prototype application

plication featured a ray gun (Figure 3.2a) and a ray tracing camera (Figure 3.2b) that could interactively trace rays and paths into "booths" and visualize the result.

This prototype was then to be tested by users to determine what features of the application should be developed further. The prototype was completed in late 2020, at a time where the widespread COVID-19 pandemic made in-person user testing impossible. Because of this, no user testing was performed on the prototype version of the application.

3.3.2 PathVis

Development of the application continued in 2022, and with pandemic restrictions being lifted, in-person user testing was now possible. The environment of the application was redesigned from a passive exploratory experience to a linear progression based one, with new features in addition to improvements to features already present in the prototype like the ray gun and ray tracing camera. A tutorial was also added to make the application accessible to users who had never used VR before, and would introduce the controls and forms of interaction.

Since the application intends to visualize and teach path tracing, the name PathVis was chosen, and the remainder of this thesis will refer to the application by its name. In this development stage, two user tests were conducted to evaluate PathVis. The first part of this stage focused on the new linear progression and tutorial that was tested in the first user tests. The second part used the feedback gathered from the first user test to improve and polish the tutorial, level design and functionality to prepare PathVis for the second user test.

3.4 Technology

This section describes the software and hardware that was chosen for developing PathVis. Figure 3.3 illustrates the overall development process for the project and the relations between the chosen software.

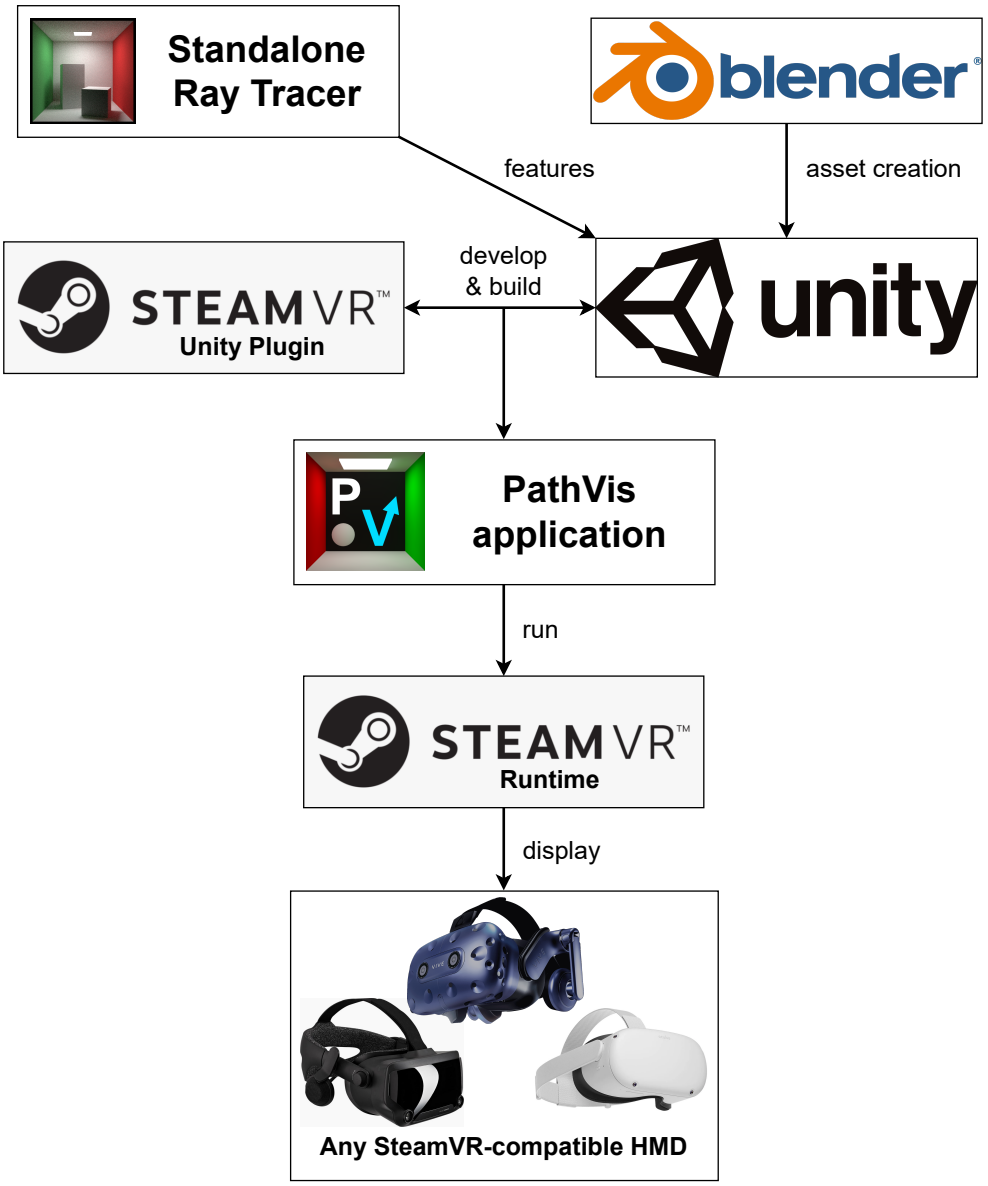


Figure 3.3: An illustration of the overall development process for the project.

Game Engine

Unity was chosen as the game engine to use for the project, as the author has previous experience with the engine, and it is a popular choice for newcomers to game development. Unity allows for quick prototyping and testing of functionality, even for VR applications.

Asset Creation

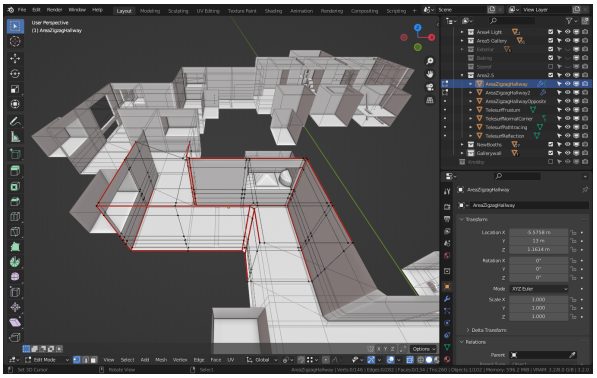


Figure 3.4: Creating the level for PathVis in Blender

While Unity has an asset store with free assets of varying quality, the free 3D software Blender was used for creating the models and environments for PathVis (Figure 3.4), as the author had experience with the software. Creating assets from scratch meant that assets with consistent quality could be tailor-made for PathVis to ensure they would perform well in VR. Blender also has support for viewing scenes in VR, which is useful for testing rough designs of 3D environments compared to viewing them on a flat 2D monitor. This feature helps identify problems early on in the design phase, such as the height of a table being too low or too high.

VR Platform

When development started, the author already had access to an HTC Vive system, which features motion controllers and utilizes external base stations for positional tracking in space. To make the application usable with as many VR HMDs as possible, it was determined that PathVis would use the SteamVR runtime, as SteamVR supports all PC-based HMDs as well as the standalone Quest 2 via PC tethering.

The SteamVR framework with the OpenVR SDK provides an abstracted layer for VR development, which means developers do not have to worry about differences between VR headsets and controllers or VR runtimes. The SteamVR Plugin for Unity features many prefabs and scripts useful for getting started quickly, as well as example scenes that can be tested in VR.

The choice of PC VR means that in-person testing requires more setup, depending on the headset it may necessitate setting up a PC with cables and base stations. The advantage of PC VR is the extra power available, which would prove to be useful for the interactive ray tracing features within PathVis.

3.5 Evaluation

This section describes the method that PathVis was tested by users. User testing was initially planned for the prototype application as well, but the COVID-19 pandemic restrictions made this unfeasible. Two user tests were carried out during the second stage of development.

3.5.1 User Testing

User testing was conducted to gain feedback on the usability and functionality of PathVis. The target audience was initially students of computer graphics, but throughout development it became clear that PathVis could be used as a general introduction to ray tracing, as there are no theory-heavy goals to complete within the application. The supervisor handled acquisition of testers, most of which had some background within computer science. Users were observed during the test and asked to fill out an online survey when they had completed the test.

Test stations

As PathVis requires a PC with SteamVR to run and enough space for room-scale VR, the tests were conducted in a PC room large enough for two test stations. For the first test station, SteamVR was installed onto one of the PCs in the room and an HTC Vive Pro was set up with 3 base stations to provide positional tracking. The second test station consisted of a powerful laptop connected to an Acer OJO 500. This HMD is one of the Windows Mixed Reality headsets and provides positional tracking by using cameras built into the HMD itself.

A virtual boundary was defined for both test stations, for the desktop station this was done by running Room Setup in the SteamVR application and then drawing the boundary by defining points in the room with one of the VR controllers. For the laptop station, the boundary was defined in the Windows Mixed Reality Portal application by moving the headset itself through space to draw the boundary.

To make sure users would not wander too far, chairs were set up between the test stations, acting as a physical boundary in case the virtual one was ignored. Figure 3.5 shows the test stations as they were set up for the first user test, with the laptop test station with the Acer OJO visible on the left, and the desktop with the Vive Pro in the back.

Figure 3.5: The two user testing stations

Carrying out the test

When users showed up for the test, they were first asked if they had previous experience with VR, and then given a brief introduction on how to equip and adjust the HMD at the test station. They were then given the VR controllers and the PathVis application was launched on the test station. The overall introduction was short, with users finding themselves inside the application within 1 minute after receiving the HMD.

Users were then observed both in the physical world as well as what they were doing in the application, via the displays on the test station PCs. Upon reaching the end of PathVis, the VR equipment was retrieved from the user, and they were instructed to fill out the survey accessible via a QR-code link posted on the door.

3.5.2 Data Generation

To generate data for evaluation of the application, the two main methods were surveys and observation. Observation would provide qualitative data, and the survey would provide both qualitative and quantitative data. The same data generation methods were used for both user tests.

Survey

The survey was used to gather feedback from testers in an organized way. A System Usability Scale (SUS) form was used to evaluate the usability of the PathVis application. This form consists of 10 questions with five options for each. The benefits of SUS is the ease of use, and it can give reliable results even with small sample sizes [33].

In addition to the SUS form, the survey has additional questions relating to experience and interest with VR and computer graphics, as well as VR learning potential. These also use a five option response type. To provide qualitative data, the survey also has a number of questions that let the user answer with text. These questions mainly asked about the users' experience with specific features within PathVis. The surveys for the first and second user tests were created with Google Forms, which provides an easy way to create and conduct surveys online. The full set of questions and statements used for the surveys are included in Appendix A.

Observation

In both user tests, users were observed during their playthrough of the application. Both the physical behavior of the user in VR as well as their behavior inside the PathVis application was observed. The displays on each test station would show what the users were seeing inside the HMDs.

Chapter 4

Implementation

4.1 Overview

This chapter goes into detail on the development of the prototype application and the two versions that were user tested.

4.2 Concept

The act of tracing a ray lies within the very name of ray tracing, so it was immediately clear that the application needed to visualize a ray in 3D space being traced or drawn from one point to another. With the depth perception and positional tracking that VR gives, drawing rays as lines works well. The application also needed to introduce a number of different concepts within computer graphics to the user, and the form chosen for this was to introduce different concepts at different locations within the virtual application. The user would then be able to learn and play with a concept at one location before moving to the next one.

It was decided that the user would be the one to perform ray tracing, by giving them a ray gun that could shoot visible rays that would then interact with objects and surfaces in the application. Most VR controllers are shaped somewhat like a gun with a trigger button, so a virtual ray gun would make for a good virtual depiction of the held controller.

Ray tracing and path tracing are typically used to render images of 3D scenes with realistic shadows and light. Because of this, the application would also need to give the user a way to render images, in the form of a virtual camera that could be picked up and moved around. The camera would perform the same type of ray or path tracing as the ray gun, but produce an image of a part of the virtual scene.

To bridge the gap between individual rays shot by the ray gun and a full image rendered by a camera, the application would also need a way to visualize rays being traced through a grid of pixels that would eventually lead to an image.

4.3 Requirements

As the goal of the application was to teach ray and path tracing by visualizing it in VR, a number of functional requirements were defined before development of the application started. In addition, a number of non-functional requirements were defined for the development and performance of the application.

In the second stage of development it became apparent that a tutorial was needed to teach users how to move and interact with the virtual objects. A first iteration of the tutorial was implemented before the first user test, and the feedback gathered during the test was then used to define a number of functional requirements for the tutorial that would be tested in the second user test.

Non-Functional Requirements

- **NFR1:** The application should run at a high framerate in VR to avoid discomfort.
- **NFR2:** The appearance of the virtual environment should be immersive and comfortable.
- **NFR3:** Unity Prefabs should be used where possible to speed up creation and modification of objects.
- **NFR4:** Script components should be generalized and reusable for many objects.

Functional Requirements

- **FR1:** The application should let the user trace rays interactively.
- **FR2:** The application should be able to visualize a traced path.
- **FR3:** The application should let the user render images with path tracing.
- **FR4:** The application should be usable with most consumer VR HMDs.

Tutorial Functional Requirements

- **T-FR1:** The user should be taught how to teleport themselves through virtual space.
- **T-FR2:** The user should be taught how to pick up objects.
- **T-FR3:** The user should be taught how to teleport with objects.
- **T-FR4:** The user should be taught how to release objects that stick to the hand.
- **T-FR5:** The user should be taught how to interact with buttons.

4.4 Design and Implementation

This section details the development of the PathVis application. The core features needed for interactive ray and path tracing in VR were implemented during the prototype stage and further improved upon for the later stage of development. Some features were changed or improved between the two user tests as well.

4.4.1 VR Functionality

When starting the project, the author already had prior experience with setting up scenes in Unity, but needed to learn how to build an application so that it would run with SteamVR. As mentioned in Section 2.3.2, the SteamVR plugin for Unity comes with examples and prefabs that simplifies the process of getting an empty scene to run in VR. A "Player" prefab from the plugin was used, which provides all needed functionality for the application to run in SteamVR and display realistic hand or controller models.

Locomotion

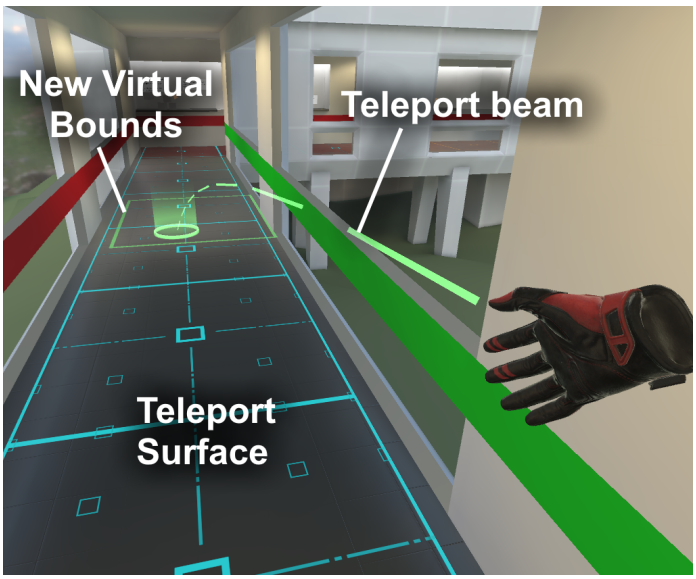


Figure 4.1: The teleportation beam

With room-scale VR, movement in physical space translates to movement in virtual space, letting the user explore the 3D environment naturally. However, virtual environments are typically larger than the defined physical bounds of a VR play space. As such, another method is required for moving greater distances. VR applications and games typically use one of two methods for moving (locomotion): continuous movement or teleporting.

Continuous movement is how a player character is usually moved in a non-VR game or application, with input making the character smoothly move through space. This is also possible in VR by having the virtual space move in relation to the user, but can introduce motion sickness for users as it appears the floor is moving when they are standing still.

Teleporting on the other hand moves the user through virtual space instantly, typically with a short fade to make the transition more comfortable. This eliminates the motion sickness that continuous movement causes, as the virtual space remains still at all times except during the faded transition. To make PathVis accessible to users who are new to VR, it was decided that teleportation would be the type of locomotion used.

The SteamVR plugin includes a prefab for teleporting, which handles controller bindings and lets surfaces be defined as areas that a user can teleport onto. Teleporting is performed by pressing a button on the VR controller to initiate a teleportation beam that can be aimed at a point on the floor the user wishes to teleport as shown in Figure 4.1. The beam also shows an outline of where the new virtual bounds of the play space will be after teleporting. When releasing the button, the user is teleported to the new space with a short fade transition. For a surface to be eligible for teleporting, it requires a *Teleport Area* component from the SteamVR plugin. These components can be locked and unlocked at run-time, making it easy to control the areas that a user is allowed to go to.

Interaction

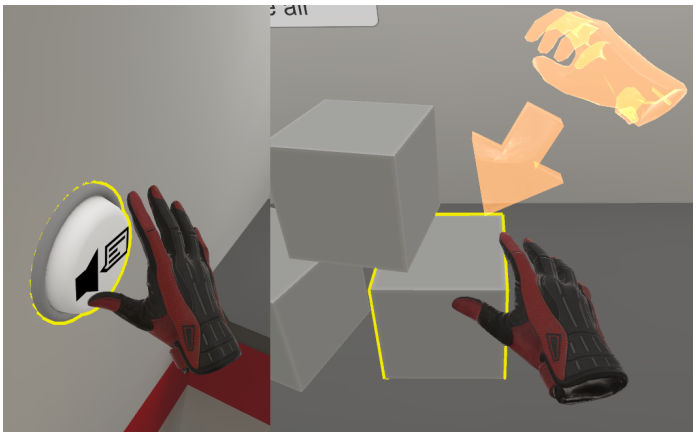


Figure 4.2: Interacting with virtual buttons and objects

Interacting with objects or user interfaces in VR can be done in a number of ways. One method is to implement a 2D user interface on a wall and then make it interactable with virtual laser pointers on the VR controllers. Another method is

to model physical 3D buttons that the user can press by touching the button with the virtual hand model.

The examples included with the SteamVR plugin typically implemented buttons as objects with an *Interactable* and a *Throwable* component. The *Throwable* component gives the object the ability to be picked up, dropped and thrown, but can also call functions when the object is picked up. Buttons in PathVis were made with this component by having the object not attach to the hand when it is "picked up". To operate such a button, the user presses a button on the VR controller when the virtual hand is close enough to the virtual button for a highlight to appear, as shown in Figure 4.2.

With the *Throwable* component, carrying an object requires the user to keep holding a button on the VR controller, which can be straining for objects that should be carried for longer times. To solve this, a new *ToggleThrowable* component was made that extends the existing functionality of the *Throwable* component, by letting the "held" state be a toggle. When the user releases the button that was used to pick up the object, the object remains attached to the hand, and a different button can be pressed to release the object again. This would be used for the ray gun (Section 4.4.7) as this object was to be carried for longer periods of time by the user.

4.4.2 Interactive Ray Tracing

With interactive ray tracing being a goal for PathVis, the application needed a way to perform ray tracing in real time. One way to trace rays in Unity is to use a *Raycast*. Raycasts shoot rays from a point and check for intersections with physics *Colliders* present in the scene [34]. Information about successful ray intersections are stored in a *RaycastHit* data structure. Of particular interest for image rendering are the *normal* (Vec3), *point* (Vec3) and *textureCoord* (Vec2) values of the *RaycastHit*, as these are essential for the path tracing algorithm that is implemented in PathVis.

To visualize path tracing intuitively, it was decided that the ray traced 3D scene would also be displayed in VR and lit with precomputed (baked) lighting, so that the ray traced image mostly matches the scene in the VR environment the user is already exploring. To avoid having the path tracing algorithm render everything in the VR scene, the path tracing only checks for intersections on objects that have a custom *RaytracingParticipator* component. This component also defines the material properties used for the interactive ray tracing, including material type (Lambertian, Mirror, Refractive, Lamp) and material-specific values like roughness for mirrors and IOR for refractive materials.

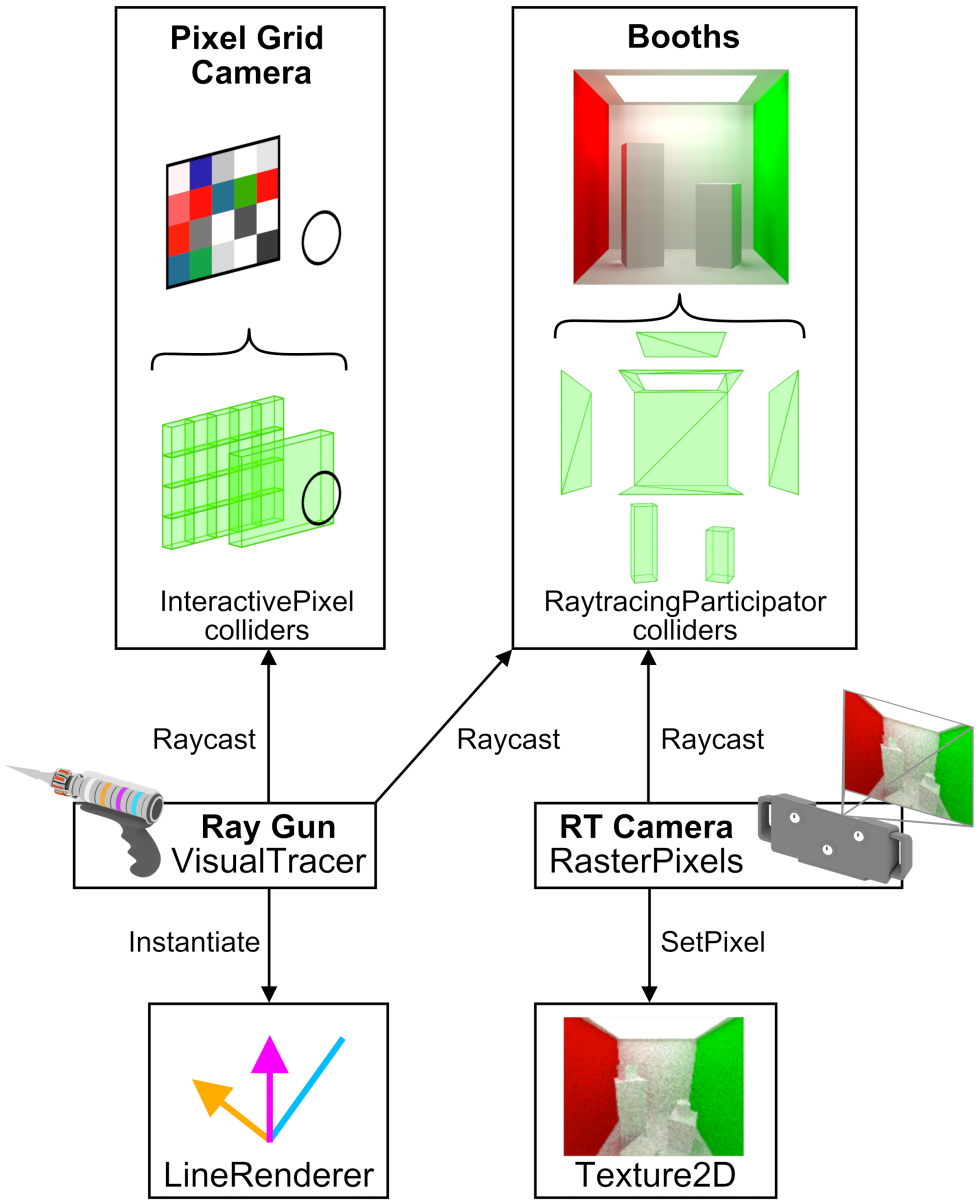


Figure 4.3: An overview of the interactive ray tracing system used in PathVis

Interactive ray tracing was implemented in two script components: *VisualTracer* and *RasterPixels*. The *VisualTracer* component is used for tracing individual paths of rays and visualizing them as lines, whereas the *RasterPixels* component is used for tracing many paths for image rendering. These scripts use a simplified version of the path tracing algorithm from the standalone ray tracer. The scripts use raycasts to check for intersections with objects and read the values of *RaytracingParticipator* components to determine where to shoot the next ray in a path. The max number of bounces per path was made configurable, but set to 6 by default. To avoid booths interfering with each other, the max ray length was set to 5 meters, and an invisible barrier was added to prevent rays from exiting booths. Figure 4.3 gives an overview of the overall components used to implement interactive ray tracing.

As *Raycast*s only detect intersections with the outside of a physics collider, refraction is made more difficult to implement. The refraction when a ray enters an object is detected correctly, but there is no refraction when a ray exits an object. To solve this problem, refractive objects in PathVis were given an additional inside collider, with the surface normals pointing in the opposite direction. For correct refraction, the inside collider uses a *RaytracingParticipator* with an IOR value that is inverse of the IOR in the outside collider.

LineRenderer components are used to make rays visible in 3D space. These components draw a line between two or more points, with the color and width being configurable both from within the Unity editor and via custom scripts. In the prototype application, the visualized paths were drawn instantly as soon as the path tracing algorithm finished. To make the rays and paths be drawn gradually from the origin, the drawing was put into a coroutine that updates positions of the endpoints in *LineRenderer* components every frame. The path is still traced instantly however, which is not a problem with scenes that are mostly static.

4.4.3 Prototype Level Design

To demonstrate different concepts within computer graphics and how they are used in ray and path tracing, the environment in the prototype version of the application was designed as a museum-like hallway with "booths" cut into the walls. Each booth would introduce a concept or graphical effect and let the user experiment by interacting with the ray gun and ray tracing camera.

The prototype environment consisted of eight booths along the walls of a big hall, with each booth featuring a different scene. The big hall was made to appear somewhat realistic in its visuals, with the intention that users would feel immersed and experience a form of sensory curiosity when exploring each exhibit.

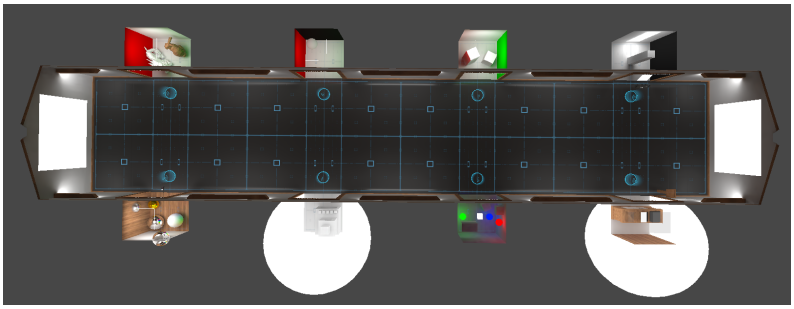


Figure 4.4: The prototype application level design

4.4.4 Tested Level Designs

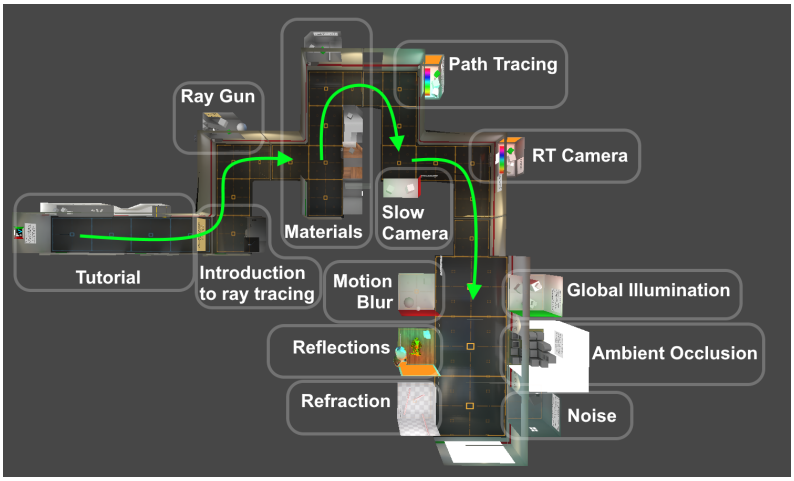


Figure 4.5: The level used in the first user test.

With the prototype being a purely exploratory experience and no goals to be fulfilled, the level design was redesigned into a more linear sequence of booths separated by gates that required goals to be completed before the user could progress. Figure 4.5 and Figure 4.6 give an overview over the levels that were used in the user tests.

Instead of a straight hallway with booths on each side, the hallway was turned into a series of 90-degree turns, with a booth cut into the wall at each turn. This made the booths more visible to the user while progressing through the application, and at the same time prevented the user from having to turn 180 degrees to see a new booth. Colored stripes in the theme of the Cornell Box were added to the walls to make corners more visible, with the left walls being colored red and the right walls colored green as shown in Figure 4.7.

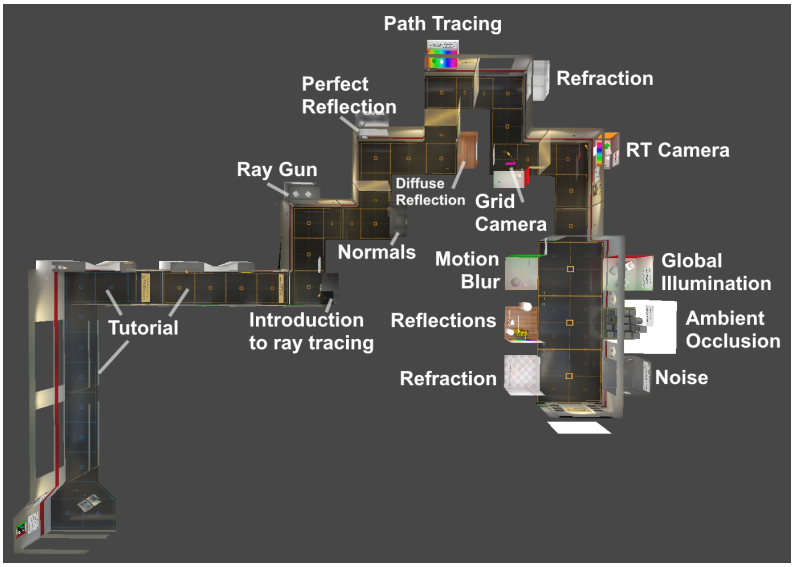


Figure 4.6: The level used in the second user test.

The gallery-like presentation of the prototype was reduced to a gallery room that serves as the final area of the application, where the user gets to explore graphical effects that are possible with the path tracing algorithm they learned about. This gallery room has 6 booths instead of the 8 featured in the prototype.

4.4.5 Tutorial

To be able to explore the learning materials available in PathVis, a user first needs to know how to use the application. A tutorial was designed and developed to meet the Tutorial Functional Requirements as specified in Section 4.3. The first iteration of the tutorial was tested in the first user test, and the gained feedback was used to improve the tutorial into the second iteration used in the second user test.

When PathVis is launched, the user appears in the tutorial area, with text and graphical instructions on the walls describing how to proceed. The teleporting prefab from the SteamVR plugin vibrates the VR controllers and shows a tooltip indicating what button is used for teleporting. The first iteration of the tutorial was very short, and introduced teleporting, grabbing and releasing sticky objects all in the same area. This proved to be an issue, and as a result the tutorial was extended into three areas teaching the concepts one by one. The second iteration also included audio narration as a supplement to the text descriptions on the walls, as described in Section 4.4.11.

How to use teleportation is communicated to the user in multiple ways: a picture in the starting area shows how the teleport beam should be aimed at the floor,

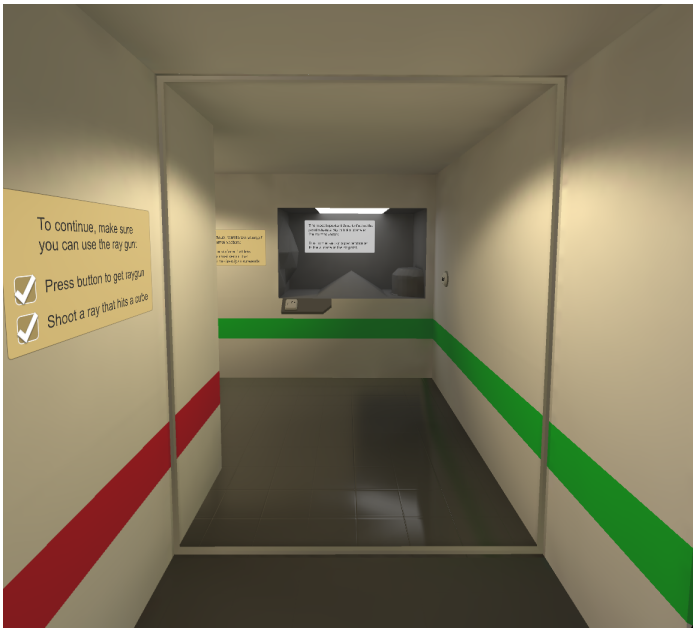


Figure 4.7: Booth located in the corner of a 90 degree turn in the hallway.

and a picture of what button on the VR Controller to use. To progress to the next part of the tutorial the user has to successfully teleport at least twice.

The next part of the tutorial introduces how to grab objects, with a floating holographic hand with an arrow indicating that the user should put their hand close to a cube before pressing the specified button on the VR controller, as shown in Figure 4.2. A gate blocks the way to the next area, to open it the user has to pick up a cube and place it in a yellow square that is 4 meters away. Because of the great distance, the user has to teleport while holding the cube. When the cube is placed in the square, the gate opens and the next area is made eligible for teleportation (Figure 4.8).

The third part of the tutorial teaches the user how to drop objects that remain attached to the hand. Like the second part, the user has to place an object in a yellow square to open a gate, but the object is a green cylinder, to differentiate it from the cubes. When the second gate opens, the user has completed the tutorial, having used teleportation both with and without holding an object, and also how to release objects that stick to the hand. While the tutorial does not explicitly require the user to interact with a virtual button, buttons are interacted with in the same way as picking up cubes.

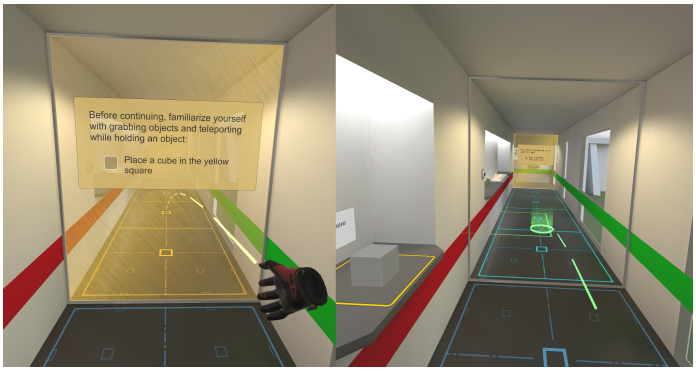


Figure 4.8: The yellow gate is only opened after a cube has been placed in the yellow square.

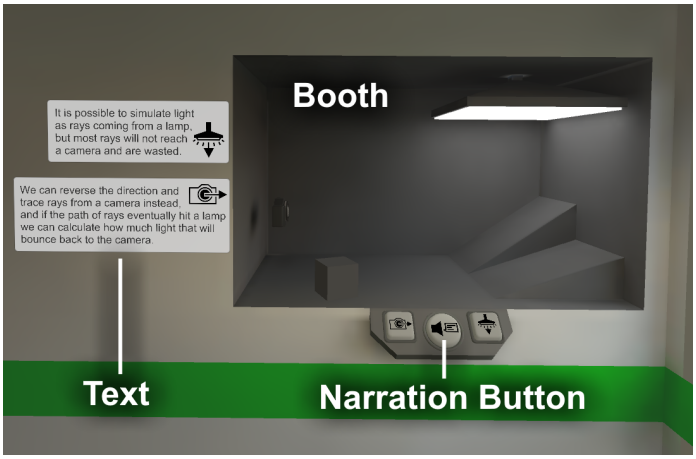


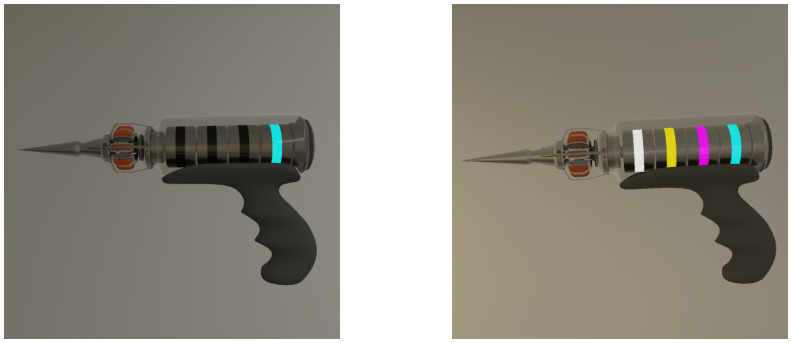
Figure 4.9: The booth that introduces path tracing to the user.

4.4.6 Introducing Path Tracing

The first booth that the user reaches after the tutorial is a booth that gives a simple overview of the path tracing algorithm used in PathVis. The booth (Figure 4.9) consists of a big lamp and a virtual 3D camera model, and the user has access to three buttons on a control panel in front of the booth. The left button is labeled with a camera icon, the center button is a narration button, and the right button is labeled with a lamp icon.

Pressing the lamp button makes a yellow "ray" of light be drawn from the lamp, and upon hitting a surface it will bounce until it reaches a maximum number of bounces. Pressing the camera button makes a cyan ray be drawn from the camera, and it will bounce around the booth in the same way. If a ray from the camera hits the lamp directly or indirectly, the path consisting of all the rays turn white, indicating that a successful path to a light source was found. The concept introduced here is that light can be simulated with rays originating either from a camera or from a light source, but that it is more useful to use the camera as the origin.

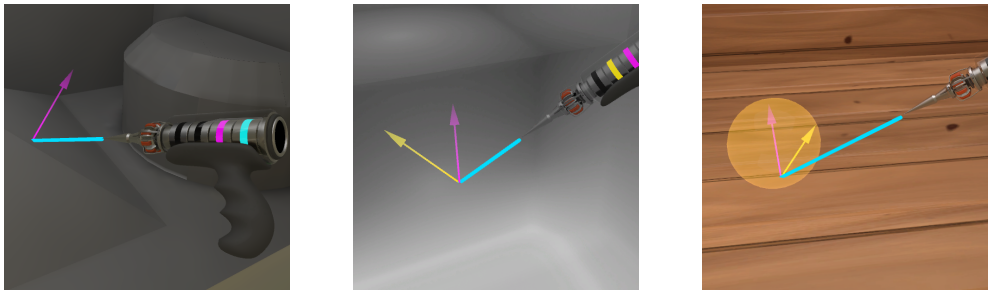
4.4.7 Ray Gun



(a) Ray gun in its initial state **(b)** Fully upgraded ray gun

Figure 4.10: The ray gun as it appears in VR.

A good use of VR controllers is to give the user something interactive to hold. The ray gun in PathVis literally puts the path tracing algorithm in the hands of the user. The ray gun was designed to appear like a high tech instrument, with a sharp point where rays are shot from. To provide sensory curiosity, the ray gun makes a distinct sound when a ray is fired, and a different sound is made when a ray hits a surface. The circular display at the back of the gun shows the color of the light that was recorded from the path tracing algorithm used by the gun. In a way, the gun can be perceived as a camera with a resolution of a single pixel. The model for the gun was created in Blender by the author.



(a) Normal Vector **(b)** Reflection Vector **(c)** Lambertian Reflection

Figure 4.11: The vectors that the ray gun can visualize.

When the user pulls the trigger on their VR controller while holding the gun, a ray is fired into the scene with a distinct cyan line. The ray is not drawn instantly, but is instead gradually drawn from the origin. The ray gun has several modes of operation, with the most basic one only letting it trace single rays that end when they hit a surface inside a booth. If a ray does not hit any booth, the cyan line is drawn in a faded way.

Visualizing Normals and Reflection

The ray gun uses the *VisualTracer* component which instantiates objects with *LineRenderer* components as shown in Figure 4.3. The drawn lines are used to visualize rays and vectors. The width of the line was defined by a custom script component that made the line gain an appearance of an arrow.

Normal vectors are visualized as purple arrows, and are perpendicular to the surface that a ray from the ray gun hits as shown in Figure 4.11a. Reflection vectors are visualized as yellow arrows (Figure 4.11b), and their direction depends on the properties of the material of the hit surface. Surfaces with Lambertian reflectance were given an additional visualization of the sphere where a random point was picked to determine the direction of the reflection vector (Figure 4.11c).

Path Tracing

Path tracing is visualized by shooting an additional ray from the point that the first ray hits. This additional ray is shot in the direction of the reflection vector as determined by the material on the hit object. This process continues until 6 rays are traced, which form a path of rays. When the entire path is drawn, the light that is found is propagated back down the path, coloring each ray in the path until reaching the gun, and then displaying the result on the circular display at the back of the gun Figure 4.12.

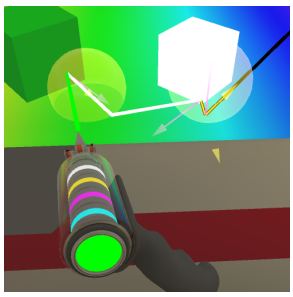


Figure 4.12: Visualized traced path from the ray gun. Light from the emissive white cube is reflected twice before reaching the gun, and is colored green when reflecting off of the green cube.

Upgrading the Ray Gun

The ray gun gets upgraded with more functionality as the user progresses through PathVis, and the current state of the gun is represented with colored rings of light inside the gun. You can see the colored rings in Figure 4.10, with the different colors corresponding to the following functionality:

- The cyan ring indicates the ability to trace rays.
- The purple ring indicates the ability to visualize normal vectors.
- The yellow ring indicates the ability to visualize reflection vectors.
- The white ring indicates the ability to perform path tracing and record light.

The method used to upgrade the gun was changed based on the feedback gathered from the first user test. In the version of PathVis used for the first user test, the ray gun was upgraded by having the user shoot and hit colored targets. Each target had an icon indicating the feature to unlock, starting with *normal vectors*, then *reflection vectors* and finally *path tracing*.

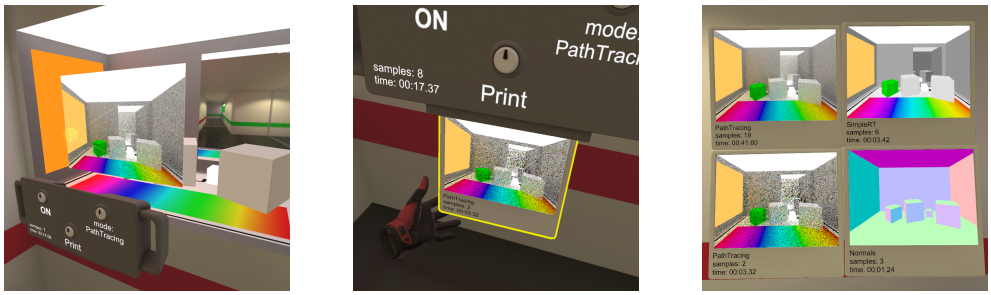
In the version used for the second user test, the ray gun is upgraded automatically when the user arrives at a booth where a new feature is required. Furthermore, the speed of the ray gun is increased when the user reached the pixel grid camera (Section 4.4.9) to reduce the time the user has to wait between tracing each path.

Changes from the Prototype

The prototype version of the ray gun (Figure 3.2a) displayed the returned color in a "color ball" on top of the gun, as well as colors for each ray segment of the traced paths being displayed in similar color balls over each hit point. The ray gun had a different display at the back that would show data from the *RaycastHit* such as the XYZ values of the normal vector at the hit point as well as number of ray bounces in the path. The text on this display was hard to read as well as not being

used for anything, so the display was removed and a new circular display was put in place, serving the role of the color ball.

4.4.8 Ray Tracing Camera



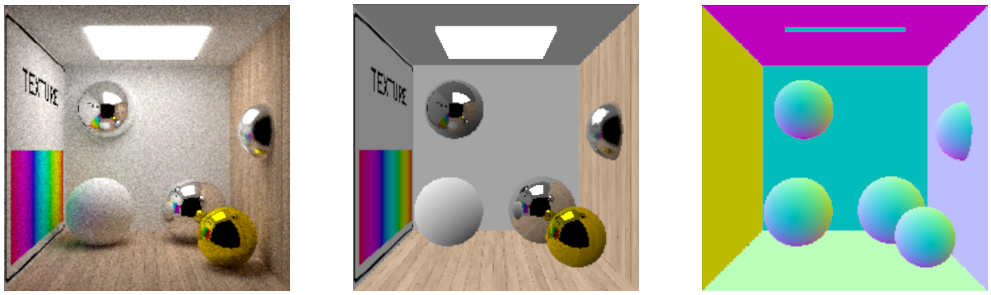
(a) Rendering a booth **(b)** Printing a picture **(c)** Comparing prints

Figure 4.13: The RT camera and its functionality.

Since rendering an image with path tracing requires tracing a substantial amount of paths, a virtual camera was made to do it automatically. The ray tracing camera (RT camera for short) uses the *RasterPixels* component which performs a similar path tracing routine as the ray gun, but renders an image by storing the results in a texture that is displayed in the virtual viewport (Figure 4.13a). The camera appears as a viewing frustum containing the viewport and a floating translucent ball representing the ray origin. A control board is located below the origin, with three knobs and text labels.

The camera is activated by interacting with the **ON/OFF** knob. When the camera is turned on it operates continuously like a progressive path tracer, displaying the number of samples per pixel as well as the elapsed time. A second knob is used to change between three different rendering modes. The camera can be picked up by the handles on each side of the control board, and the ray origin can be grabbed directly and moved to create lens effects like tilt-shift, wide angle or zoom of the rendered image. Rendering is restarted each time the camera or ray origin moves, or when the rendering mode changes.

As rendering restarts each time the camera is moved, an option to print a rendered picture onto a grabbable picture frame was implemented. Pressing the *Print* knob (Figure 4.13b) on the control board creates a picture frame with a copy of the rendered texture, which will slide out of the bottom of the control board. The picture frame can be grabbed by the user and will remain floating in the air when released. The picture frame includes a text label showing the rendering mode, elapsed time and number of samples at the time of printing. By printing multiple pictures with different modes and number of samples, the results can be compared visually by the user as seen in Figure 4.13c.



(a) PathTracing **(b)** SimpleRT **(c)** Normals

Figure 4.14: The different rendering modes of the RT camera.

The mode knob on the control board cycles between three rendering modes:

- PathTracing (Figure 4.14a): Renders images using the path tracing algorithm as described in Section 4.4.2.
- SimpleRT (Figure 4.14b): Renders images with solid colors, textures, reflections and refractions, with simple shading based on surface normals.
- Normals (Figure 4.14c): Displays the XYZ coordinates of surface normals as RGB color values.

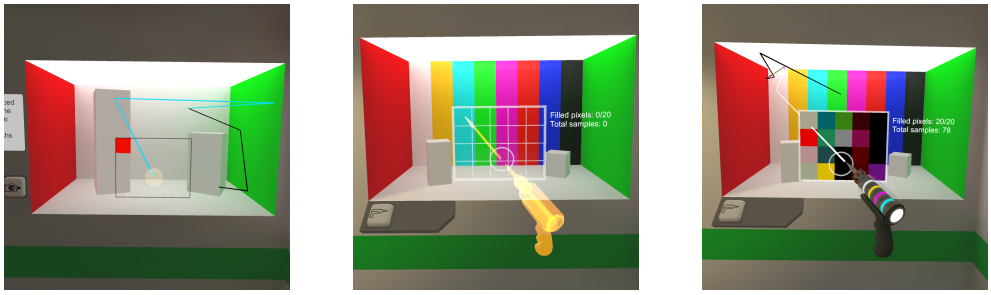
For performance reasons, the RT camera renders pictures at a resolution of 320x240. The rendering process takes place in a coroutine that renders a number of rows of pixels every frame. In the prototype application, only a single row of pixels would be rendered each frame. To speed up rendering without affecting VR performance too much, a maximum frame time value was implemented, letting rows of pixels continue to be rendered until the frame time limit is hit. To guide the user in placing the camera, the camera switches to real-time ray tracing using the SimpleRT mode at a resolution of 32x24 while the camera or ray origin is held by the user.

Changes from the Prototype

The prototype RT camera did not have the printing feature, instead having a button that would cycle between resolutions for the rendered image. An additional "UVs" rendering mode was also featured, displaying X,Y texture coordinates as R,G color values. This mode was removed as the application would not teach texture coordinates, as well as a lack of texture coordinates for some of the collider types used for interactive ray tracing, which would appear black in this render mode.

4.4.9 Pixel Grid Camera

To bridge the gap between the ray gun that can trace individual paths and the RT camera that can render images with thousands of pixels, a visualization of rays being traced through a grid of pixels was deemed necessary. The initial design



(a) First version **(b)** Second version **(c)** Second version, filled

Figure 4.15: The two versions of the pixel grid camera

(Figure 4.15a) was a simplified version of the RT camera with a much lower resolution, which would remain static and could be turned on and off from a button on the wall. This camera visualized rays in the same way as the ray gun, tracing a path for one pixel at a time but but at a higher speed than the ray gun.

Based on observations of users during the first user test, the pixel grid camera was redesigned to be more interactive and to let the user do the path tracing themselves by shooting rays from the ray gun through the pixel grid. A floating circle was added as an indicator for the camera origin and where to use the ray gun from. A floating holographic model of the ray gun with an arrow (Figure 4.15b) was added to indicate the correct use of the pixel grid camera. The new pixel grid camera also presented the opportunity to add a goal for the user to fulfill; to render a picture by tracing paths with the ray gun as shown in Figure 4.15c. The pixel grid camera gives an element of sensory curiosity as the grid of pixels is filled by the ray gun.

Each of the pixels in the pixel grid camera are implemented with an *InteractivePixel* component and a box collider. A larger box collider is placed between the camera origin circle and the pixels (Figure 4.3). When the *VisualTracer* component on the ray gun traces a ray that intersects both the large collider and one of the pixel colliders, it is registered as a valid hit and the color of the pixel is updated once the visualized path has finished drawing. A separate *InteractivePixelTracker* component is notified on each valid hit, and keeps track of the number of filled pixels as well as total samples.

4.4.10 Gallery Room

In order to showcase the graphical effects that are possible with the path tracing algorithm used in PathVis, a large "gallery room" was designed and implemented. The gallery room has six booths along the left and right wall as shown in Figure 4.17c. The intended purpose of this room is for the user to use the RT camera to explore the behavior of its rendering algorithms. These booths are larger than

any of the previous booths, extending all the way to the floor and letting the user teleport or move themselves into a booth if they wish. Some booths have objects that can be picked up and moved by the user.

The prototype application was an early version of the gallery room, having eight booths instead of six. Some booths were cut entirely and others were changed to become the versions used in the user tests. An explicit goal was added before the second user test, requiring the user to render and print at least one picture of each of the six booths in order to complete the playthrough. The printed pictures also appear on the far wall of the gallery room, with a text label indicating the graphical effect showcased (Figure 4.16).

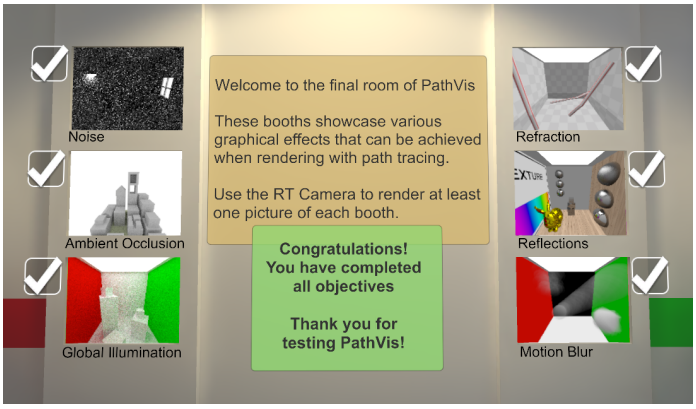


Figure 4.16: The gallery room picture wall after a completed playthrough

Each booth is designed to showcase a different graphical effect, some requiring the PathTracing rendering mode of the RT camera while others can also be demonstrated with the SimpleRT mode. The user can also use the ray gun in any booth to explore while letting the RT camera render continuously. The design and purpose of each booth is described in the following sections.

Global Illumination

This booth is a variant of the Cornell Box, a popular model for testing rendering algorithms. The left and right walls are colored red and green respectively and a large lamp is situated in the ceiling. Two tall white boxes are placed within the booth as well as two small cubes that the user can pick up. The strongly colored walls will diffusely reflect light onto the white boxes, giving them a slight color tint. This booth is intended to be used with the PathTracing mode on the RT camera. The ray gun can be used to find paths where light bounces between a colored wall and a white object before reaching the gun.

Ambient Occlusion

A simple way to achieve ambient occlusion is to treat rays as entering the sky when no intersections are found, and letting the sky act as a light source. This causes open areas to be well lit since most reflected rays will enter the sky, but areas that are less open to the sky will be shaded. This booth is designed to appear like a simple city with "buildings" of different heights. Instead of walls, this booth has a large white sky that acts as a light source. When rendered with the PathTracing mode on the RT camera, areas between the "buildings" will appear more shaded than the "rooftops". The ray gun can also be used to verify that rays hitting surfaces in open areas will bounce into the sky more often than rays hitting surfaces in covered areas.

Noise

The path tracing algorithm used in PathVis is a simple brute force algorithm based on the one implemented in the standalone ray tracer (Section 3.2.1). For diffuse materials, a random reflection direction is chosen each time, unaffected by the location of light sources in the scene. As a result of this, scenes with small light sources are difficult to render, appearing very noisy and requiring many more samples to converge to an acceptable result. All other booths in PathVis deliberately use large light sources to hide this issue, but this booth showcases the limitations of PathVis by having two very small light sources; a lamp and a window. The RT camera can be used to see the noisy renders in practice, and the ray gun can be fired into the scene to gain an intuition for why the result is so noisy.

Refraction

A common real-life example of refraction is a straw placed in a glass of water. If the straw is positioned at an angle, the straw will appear to bend below the surface of the water. This booth was designed to appear like a large tub of water, with three large straws placed within. When rendered with the RT camera in either the SimpleRT or PathTracing mode, the straws appear to bend as the rays refract at the water surface.

Reflection

To showcase reflections and rough reflections, this booth has spheres with reflective materials. Rough reflections are achieved by modifying the reflection vector by a small random component. The ray gun can be used to see the how the behavior of the reflection vectors differ between the spheres. To give the spheres something interesting to reflect, the booth has a wooden texture on the floor and right wall, and a rainbow texture on the left wall. A simplified version of the Stanford bunny [35] is also placed in the booth with a reflective golden material. Both

the SimpleRT and PathTracing modes of the RT camera can be used to see the reflections.

Motion Blur

Motion blur is made possible with ray tracing by distributing rays across time. In PathVis the same effect is achieved by letting objects move during rendering. The three objects in this booth move continuously and in different patterns. In the top left is a sphere that moves to randomly chosen positions within a small space. The sphere in the middle moves back and forth on a diagonal line, and the cube in the bottom right rotates. When rendered with the RT camera, the objects will first appear to smear across the image as rows of pixels are rendered, but over time more samples will be taken with the objects in different positions, and the result averages out to a smooth blur.

4.4.11 Audio Narration

Text can be difficult to read in VR, depending on the HMD that is used and the size and placement of the text. To present a lot of information in VR, it can be more suitable to utilize audio narration. An audio narration system was designed and implemented in preparation for the second user test of PathVis. The system consists of recorded speech that is played when the user interacts with a narration button, as shown in the left of Figure 4.2.

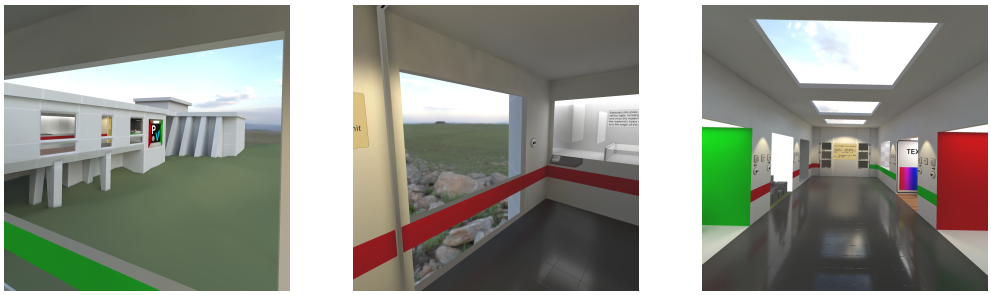
Buttons were chosen as the method to initiate playback, as then the user can control when and where they want to hear narration. Narration buttons turn green while narration is playing, and playback can be stopped by pressing a green narration button. Only one narration button can be playing at a time. The narrated audio is played from an *AudioSource* component that is localized within the relevant booth. This way, the user can hear where the sound is coming from, as if the booth is speaking to them.

Each booth in PathVis has one narration button associated with it, and the recorded speech reads out loud the text and tasks for that particular booth. The audio narration goes into more detail than the text boxes in each booth and sometimes provides hints for the tasks the user has to perform, as the text boxes were deliberately made brief to make reading less of a burden in VR.

4.4.12 Exterior

To make the hallway in PathVis appear less closed and maze-like, large windows were cut into the walls in several areas (Figure 4.17). The outside world was given a backdrop consisting of a spherical panoramic texture depicting a grassy and rocky field [36]. As the gallery room and other parts of the hallway would be

visible through the windows, a simple 3D exterior was modeled in Blender, making the hallway appear to be located within a concrete building. The windows also provide light to the virtual environment, preventing it from appearing too dark and secluded.



(a) Tutorial windows **(b)** View of grassy field **(c)** Gallery windows

Figure 4.17: The exterior as viewed through windows in the walls and ceiling

Chapter 5

Results

This chapter gives a brief description of the final application and its performance including video demonstrations and a link to the application itself. The evaluation data from the user tests is then provided and will be discussed in the next chapter.

5.1 The PathVis Application

The final application is the result of the implementation process as described in Chapter 4. This is also the version of PathVis that was tested in the second user test.

Video Playthrough

Two recorded playthroughs of PathVis were made for demonstration purposes. The first is a quick playthrough without the use of narration buttons, and the second one is a longer video where all narration buttons are played back. Both videos were recorded using the same version of PathVis as the one that was tested in the second user test.

- No Narration (6:46): `https://youtu.be/ho7HpW9-PHs`
- With Narration (15:20): `https://youtu.be/a7GscTKB0hw`

Download the Final Version

The final version of PathVis can be downloaded from the link below. The application requires a PC with SteamVR installed and a VR HMD connected in order to run. The Unity project for the application can also be downloaded from the Github repository [10].

`https://github.com/rikeri/Pathvis/releases/tag/build`

(a) RT camera off **(b)** RT camera on

Figure 5.1: SteamVR Frame time performance graph: The top and bottom graphs show CPU and GPU timings respectively.

Performance

The environment in PathVis is mostly static meshes lit with baked light. This makes the application run well on any VR-ready system when the RT camera is not in use. Figure 5.1a shows the SteamVR frame timing performance graph while PathVis is running. The blue area is the time used by the CPU and GPU for each rendered frame, and the light green area is the unused time budget. As long as the CPU and GPU timings stay within the green area, the application is able to deliver frames at the full refresh rate of the VR HMD. When the RT Camera is active, some frames are dropped leading to slight lag and judder that can be observed when moving the virtual controllers around.

5.2 Evaluation

In this section, the results from the two user tests are presented. The first user test took place on April 1st 2022, with the second user test being divided over 3 days between May 19th to 24th 2022.

5.2.1 First User Test

The first user test had 7 test users that tried the version of PathVis as shown in Figure 4.5. All users had a background in computer science, but not necessarily experience with computer graphics. Most users were completely new to VR or had tried VR only a few times in the past.

The result from the survey is presented in Figure 5.2 while the SUS scores are shown in Figure B.1 in Appendix B. Calculating the SUS score as described in [33] gives an average score of 78.2 for the first user test. A SUS score of 68 is considered average, and a score between 68 and 80.3 has grade B and is considered good [37].

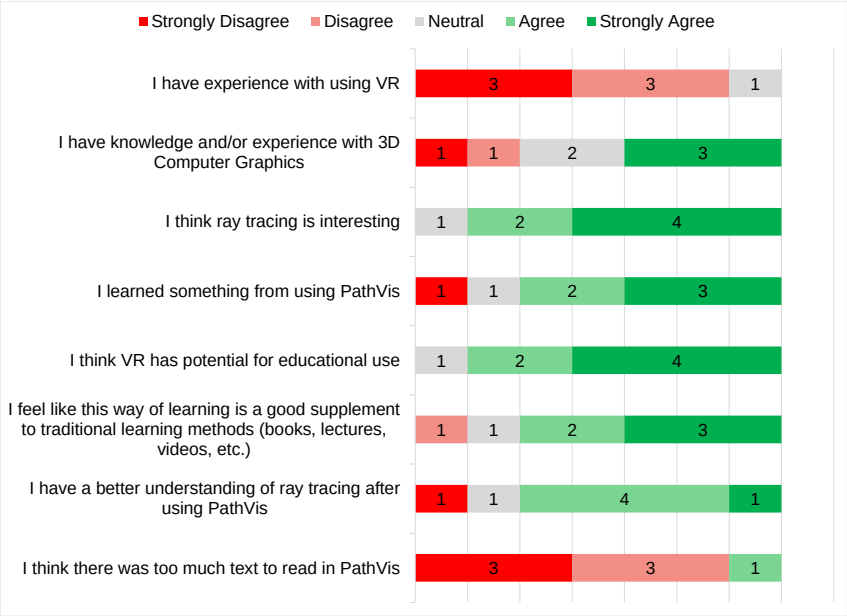


Figure 5.2: Responses for the general questions of the first survey.

Text Responses

The survey included four questions where the user could write a text response. The four questions focus on the users' experience with the tutorial, ray gun, ray tracing camera and overall suggestions for potential improvements that could be made to PathVis. The full answers are provided in Appendix B.1.2, and a summary of the responses are provided in this section.

- **Tutorial:** Most users managed to pass through the tutorial without issues. Some users had trouble grabbing or releasing objects, and wished for a better introduction to the controls.
- **Ray Gun:** All users expressed a positive experience with the ray gun, finding it fun to use and experiment with. Users liked to see the visualized rays and paths that the ray gun produced.
- **Ray Tracing Camera:** Users liked the ability to render and print pictures, but that the camera could be improved by having it serve an actual use. Some users found the camera to be difficult to move in order to take pictures of the booths in the gallery room. One user mentioned that the path tracing rendering of the camera was rather slow.
- **Improvement Suggestions:** Users suggested a more iterative form of teaching the learning material, and a more clear distinction between ray tracing and path tracing. One user expressed frustration with reading text in VR, and suggested the addition of narration. Another suggestion was to give the gallery room an explicit goal of printing pictures with the camera.

Observations

Observing the users during the tests provided valuable insight for features that could be improved before the next user test. Some issues were discovered as multiple users would consistently run into the same problems. The list below gives an overview of some of the observations, and changes that were made before the second user test.

- Multiple users tried to pick up objects by using the teleport beam, or did not understand why the beam could not be used on the walls.
    - The tutorial was expanded to feature an area focusing on only teleporting without objects, to make users comfortable with teleportation before introducing object pickups.
- Users would quickly shoot each of the upgrade targets for the ray gun, without experimenting with the functionality that they unlocked. One of the upgrade targets was located in a branch of the hallway, which multiple users had trouble finding.
    - The branch in the hallway was removed, and more booths were introduced to teach concepts one at a time. Upgrading the ray gun was made to happen automatically as users reached new booths. More

gates were added, requiring the user to use the new ray gun functionality before proceeding.

- Users tried shooting the ray gun through the pixel grid camera.
  - The pixel grid camera was redesigned to be interactive and usable with the ray gun.
- Users were not sure what to do in the gallery room, as there was no clear goal.
  - A goal to render and print a picture of each booth was added.
- Users often skipped reading text and passed by booths that did not indicate a goal for the user to fulfill.
  - Narration buttons were added for each booth, and more booths were given explicit goals.

5.2.2 Second User Test

The version of PathVis shown in Figure 4.6 was tested by 11 users, including 6 of the 7 testers from the first user test. Like the first user test, many testers were new to or had limited experience with VR. Of the 11 users, 10 responded to have a background in computer science.

The result from the survey in the second user test is presented in Figure 5.3 with the SUS scores shown in Figure B.2. The calculated average SUS score for the second user test is 74.1 which corresponds to a grade B. An additional SUS score was calculated for the returning testers only, with a result of 79.6.

Text Responses

Like the survey used in the first test, the second survey includes questions where the user can write a text response. The same four questions from the original test were used, and a question relating to the use of narration button was added. The full answers are provided in Appendix B.2.2, and a summary of the responses are provided in this section.

- **Tutorial:** Most users managed to pass through the tutorial without issues. Some users suggested a better introduction to the controls.
- **Ray Gun:** Users found the gun easy and fun to use. One user suggested that the gun use a laser pointer to make it easier to aim. Another user suggested the rays and vectors could be made more distict by using dotted lines in addition to different colors.
- **Ray Tracing Camera:** Users found the RT camera nice to use, but clunky at times. Some users said the task of taking pictures was repetitive, and the camera could benefit from having additional tasks.

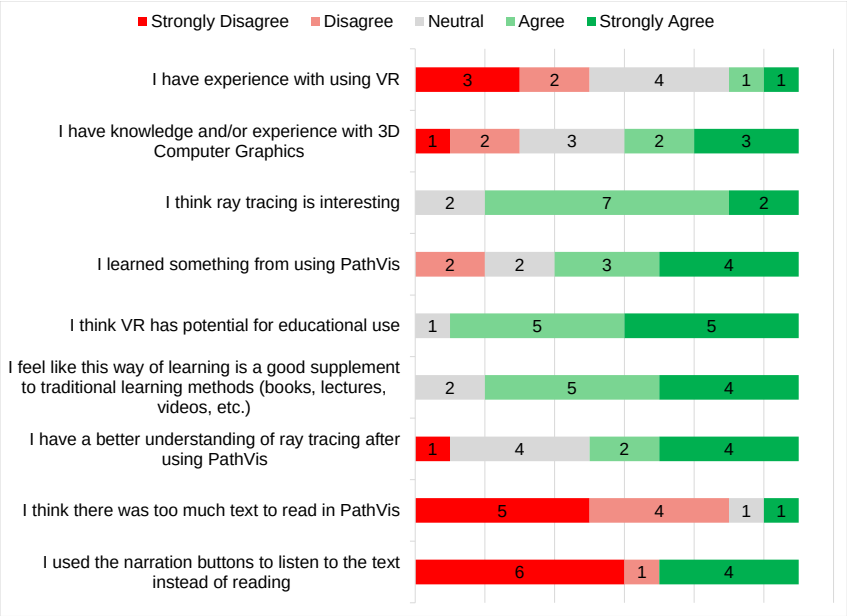


Figure 5.3: Responses for the general questions of the second survey.

- **Narration Buttons:** Of the users who responded that they used the narration buttons, they enjoyed being able to listen to content instead of reading, which could be difficult with glasses or contact lenses.
- **Improvement Suggestions:** Users appreciated the use of VR to teach path tracing, but felt that the current state of PathVis was more like a tutorial and wished there was more of a "game" with more complex and open ended tasks for the user to perform.

Observations

The users seemed to run into fewer problems during the second user test, based on observing how the users behaved outside and inside the application. The list below gives and overview of the observations made during the second user test.

- Some users skipped past the very first area that introduced VR controls and narration buttons, either accidentally or on purpose.
- Several users played around with the cubes in the tutorial, throwing them around to see how they would react.
- How much a user would move in the physical world greatly differed among the participants. Some users would stand in a single spot and use in-game teleporting to move around. Other users would walk until they reached the bounds of the physical space, sometimes lightly bumping into a table or a chair.
- Users who were not using narration buttons did not immediately learn of the goal in the gallery room, instead ending up exploring each booth without taking a picture, and having to go back when they read the goal on the far wall.
- At the pixel grid camera, some users would not shoot through the camera origin circle, and appeared to be confused when some of the rays would not fill in pixels.
- Booths with no gates and goals for the user to fulfill were sometimes skipped entirely.
- Some users printed pictures in the gallery room with an unintended RT camera render mode. For instance, the ambient occlusion booth would be rendered with SimpleRT which does not produce ambient occlusion.
- The goal at the booth introducing normal vectors was to hit a surface whose normal vector pointed downwards. All users eventually understood that they needed to hit the ceiling in the booth for the goal to be fulfilled.
- The goal in the diffuse reflection booth was to hit the same spot three times, but proved to be difficult to fulfill as the required precision was too high.
- Users would not teleport into any of the gallery booths on purpose, and would not pick up or interact with any of the interactive objects within booths.
- The printed pictures from the RT camera would not be used for anything, and users tended to only grab pictures in order to print more for the goal.

Chapter 6

Discussion

This chapter discusses the results from the two user tests that were presented in Chapter 5, then the author reflects on the development process and various elements and limitations of the PathVis application. At the end is a discussion of the requirements and research questions for the project.

6.1 User tests

During development, the PathVis application was only tested by the author and the supervisor a few times. The first user test was the first time that valuable feedback could be gathered for the project. The feedback was used to improve features in PathVis that were difficult to use, and to add additional functionality that was then tested in the second user test.

6.1.1 Participants

The original target audience for PathVis was to be students who were studying computer graphics, but due to time constraints it was not easy to find a lot of testers from this group. As a result, the users who participated in the tests were mostly people who had studied or were currently studying computer science. The supervisor was responsible for scheduling the test and contacting the test participants for both the user tests.

6.1.2 Survey Responses

This section will discuss the statements with structured responses from the surveys used in both user tests. As both surveys used the same questions, responses for both are discussed at the same time. The responses were given in a 5-point scale ranging from Strongly Disagree to Strongly Agree. Two average scores were given as value between 1 to 5 respectively.

I have experience with using VR

Average Score: 1.7 (1st test) | 2.5 (2nd test)
Most testers responded as having little to no experience with VR, which meant that valuable insight could be gathered for how to make PathVis accessible to as many people as possible. The second user test had a few users who agreed or strongly agreed to this statement. As an educational tool it is important that the user quickly learns how to use the tool, so they can then focus on the learning material.

I have knowledge and/or experience with 3D Computer Graphics

Average Score: 3.4 (1st test) | 3.4 (2nd test)
This statement was used to determine whether the testers fell within the original target audience for PathVis or not. Prior experience with computer graphics is not required to play through PathVis, as there are no complex tasks that the user has to carry out. If PathVis were to be developed further in the direction of an educational tool, the addition of tasks that test the user on more in-depth knowledge would be necessary.

I think ray tracing is interesting

Average Score: 4.4 (1st test) | 4.0 (2nd test)
Similar to the previous statement, an interest in ray tracing is possibly an indicator for the target audience. A person with zero interest in ray tracing would have little benefit or motivation to play through PathVis. An interest can also mean that the user is more likely to experiment with the tools available in the application to explore how ray tracing works.

I learned something from using PathVis

Average Score: 3.9 (1st test) | 3.8 (2nd test)
PathVis can teach the user how a simple path tracing algorithm works in practice, which is the aim of this statement. It is also possible that users who were new to VR felt like they learned how to use VR after playing through PathVis, as PathVis starts with a tutorial that teaches concepts used in many VR games and applications. Some users strongly disagreed to this statement, likely because they already had knowledge of ray tracing and PathVis did not introduce anything new.

I think VR has potential for educational use

Average Score: 4.4 (1st test) | 4.4 (2nd test)
Most users responded agreed to this statement, indicating that VR can be used in an educational context. As virtual reality devices become more accessible like the standalone Meta (formerly Oculus) Quest 2, it is possible that VR will see more use as an educational tool.

I feel like this way of learning is a good supplement to traditional learning methods (books, lectures, videos, etc.)

Average Score: 4.0 (1st test) | 4.2 (2nd test)
The majority of the test participants agreed with this statement, possibly related to the visual presentation of the learning material within PathVis. VR can be used to introduce a concept within computer graphics by visualizing it in 3D, giving the user an intuitive understanding that can then be followed up by more traditional methods.

I think there was too much text to read in PathVis

Average Score: 1.9 (1st test) | 1.9 (2nd test)
Most users disagreed to this statement, with approximately the same average score for both user tests. Text in VR can be difficult to read for several reasons, mainly resolution and visual clarity. The resolution of the displays in VR HMDs has increased over time, but the lenses can be equally important for visual clarity. The lenses used in VR HMDs typically give a "sweet spot" with clarity in the center, but worse visual clarity near the edges. As a result the user has to move their head to focus on text as they read it, which can lead to fatigue. This was taken into account when developing PathVis, and the text boxes on walls were kept short and concise.

I used the narration buttons to listen to the text instead of reading

Average Score: 2.5 (2nd test)
This statement was only used for the second user test survey, as the narration buttons were a new addition. Some users preferred to read text themselves, but other users seemed to like using the narration buttons. The response to this statement indicates that users either did not use the narration buttons at all, or they used them every time. It is possible that the extra step of having to reach the narration button on the wall made it more of a hassle to use. Another possible reason for testers avoiding narration buttons could be the fact that the tutorial did not explicitly demonstrate the usage of a narration button, but from observation it seemed that all users would eventually press a narration button at least once to see what it did. The reason for using a button format was that users could decide themselves when to listen to narration, but perhaps the usability could be improved if a button on the VR controller was used to start the narration playback.

6.1.3 Observations and Text Responses

This section will discuss the responses to the open questions as well as observations of the users. Like the previous section, results from both user tests will be discussed at the same time, highlighting differences between the two tests.

Tutorial

The tutorial section was put into PathVis to teach the user how to move themselves around in the environment and how to interact with objects and buttons. The first version of the tutorial tried introducing too many concepts at once, causing confusion for some users. The second tutorial worked better, as users first learned how to teleport themselves before they were taught how to pick up objects. Multiple users played around with the cubes in the tutorial, throwing them around and seeing how they interacted with each other. This may have been a case of users exploring VR technology for the first time.

While the second tutorial had better results than the first, some users still had issues with remembering the buttons to use on the VR controllers. The controllers are represented by virtual hands in PathVis, which may have been distracting for users who did not have any previous experience with VR controllers. Some users also tried to pick up a cube by squeezing it between the virtual hands, when the intended method is to pull the trigger on a controller when a hand is close enough. The SteamVR plugin for Unity has a feature for highlighting a button on the virtual controller, but this feature proved to be difficult to use in practice, as the virtual controller model turned invisible when an object was picked up.

Ray Gun

Most users had a positive impression of the ray gun, finding it interesting and fun to use. Some users kept holding the ray gun even in areas where it was not intended, such as when reaching the RT camera. A small poster was added on the wall for the second user test to remind users that the ray gun can be dropped. The idea behind the gun was that the user gets to experiment with ray tracing interactively, but some users would only use the ray gun when explicitly required.

When users reached the booth that introduced reflection vectors, they would often shoot the ray gun forwards onto the floor of the booth, making it difficult to notice the yellow reflection vector as it would appear behind the normal vector. This booth did not have a goal to fulfill, and it would probably be more engaging if the user was required to shoot a ray that would be reflected in a specific direction to hit a target indirectly.

The diffuse reflection booth had a goal of hitting the same exact spot three times, in order to demonstrate that the yellow reflection vector was chosen randomly every time. The script component responsible for tracking the hits was made to only count hits if they fell within a certain distance from the first hit. This distance was chosen so that the user would need to focus in order to fulfill the goal. In practice, this distance was made too small, and many users had to

make many attempts before fulfilling the goal. Increasing the distance or designing a different goal for demonstrating diffuse reflection would likely improve this booth.

Ray Tracing Camera

The RT camera had a more mixed reception. Users liked rendering and printing images but felt that the camera was not as useful as the ray gun. The printing feature was not used in a meaningful way other than having pictures show up on the picture wall of the gallery room. Printing multiple pictures required the user to grab and remove the printed picture before printing new ones, which some users did not like. To complete the goal in the gallery room, some users would quickly print a picture of each booth, without seeming interested in the actual rendered result.

To make the RT camera more engaging to use, the user could be tasked to assemble a scene by stacking light sources and objects on top of each other, and then rendering the result with the camera. This was already possible with the grabbable objects located in the various booths, but users did not interact with these objects, as it was not clearly communicated which objects were grabbable and which were static parts of the environment.

Some users found the rendering speed of the RT camera to be slow and expressed a wish for better sampling. As mentioned in Section 4.4.2, the rendering routine takes place in a script component and is limited to the time available for a frame. A more efficient implementation of interactive ray tracing could be beneficial, as the brute force path tracing routine is unable to handle point light sources.

Improvement Suggestions

The overall impression of PathVis seemed to be that teaching ray tracing and other computer graphics topics is a good use case for VR. However, some users said the PathVis application felt like a tutorial without a "game" to follow up what the user learned. Booths that did not have any explicit goals were often skipped, and the application could be made more engaging by adding more goals that give better feedback to the user.

6.2 Development Process

Developing a VR application alone can be a challenge, as there are many different types of skills required. The game engine and VR SDK with its examples and prefabs can help for part of the process, but the developer is still responsible for assembling the scenes and creating the scripts and making sure that they work

well when used in VR. When starting the project, the author already had experience with creating 3D models and textures with the Blender software, as well as experience with assembling static scenes in Unity. This made development of the application easier, as the author could focus on becoming familiar with creating script components and how to use the interaction components provided by the SteamVR plugin.

Development followed a general process of designing, implementing and testing individual features based on the requirements. Since the interactive ray tracing elements were core features, they were developed first with the rest of the application being a simple test environment resembling an early concept of the "booths" that are featured in the tested version of PathVis. After the core features were mostly implemented, the prototype environment was modeled and put into place. Further development of the core ray tracing features continued, ensuring that it worked with the environment as expected.

Perhaps the most difficult part of developing for VR was the interaction system. While the SteamVR plugin provided a good basis for teleporting and grabbing objects, there was no easy way to create a UI with the Unity Canvas UI components. A custom laser pointer interaction system for pressing UI buttons was developed but later cut, as it required a lot of manual setup to make each button work. Had the development of the project started today, it is likely that the XR Interaction Toolkit would be chosen as the SDK, as it has a wider set of interaction components that are better integrated with Unity.

6.3 Visualization in VR

A key benefit of using VR in an educational application is the added immersion offered by the 6DoF stereoscopic view made possible by the HMD. This gives the user depth perception and makes it easier to gain a spatial understanding of a visualization compared to one that is viewed on a flat 2D screen. The visualization of rays in PathVis could be more difficult to understand when viewed on a 2D screen, as the lack of depth perception and ability to move around naturally in space makes it hard to see where exactly rays exist within 3D space. The sentiment from the test users supports the notion that visualizing ray tracing is a good use case for VR.

If PathVis is launched on a system with no HMD is plugged in, it will run with a basic desktop interaction scheme as provided by the SteamVR plugin for Unity. Using the mouse and keyboard, it is possible to "fly" through the application and interact with buttons by clicking on them. It is not possible to play through the entire application however, as the ray gun will not work in the basic desktop interaction scheme and instead will remain stuck to the cursor, preventing any further

interactions. PathVis was designed and tested as a VR-only application, as the visualizations work best in VR.

6.4 Booths

The learning material in PathVis is presented as text and narration associated with each booth. Booths were chosen as they offer an easy way to differentiate between the environment of PathVis itself and the specific areas where interactive ray tracing can be used. The smaller booths in the narrow hallway part of PathVis start at kitchen counter top height and extend upwards, giving the user the ability to interact with objects inside. The text description for each booth is typically located on the back wall of the booth. Putting the text outside the booth would prevent the visualized rays from obscuring the text, but it would require the user to look outside the booth to read. The audio narration gives a more comfortable presentation of material, and users who used the narration buttons seemed to have a smoother experience with PathVis. However, many users preferred not to listen to audio, which means that the option for both text and audio narration is necessary.

To improve the booths, they could be redesigned to appear more like a museum exhibit, with text located on an angled board near the user, with the contents of the booth on display above the text. Another option is to find a different format to present the ray tracing concepts. The advantage of a booth is that it has walls and a ceiling that rays will bounce off of - without them, rays would often escape into space. The booths in the gallery room let the user teleport into the booth if they wish, but no users would intentionally teleport into a booth, possibly because they had become accustomed to the smaller booths in the hallway section. A potential solution is to make a booth into a room for the user to explore. The room will contain rays just like the booth, and can be linked to other rooms via doors. Each room could then be used to teach its own concept within ray tracing, with specific objects and tasks for the user to solve, similar to the escape room format used by Sølve [9].

6.5 Educational Games

The environment of PathVis facilitates both sensory and cognitive curiosity: sensory curiosity is achieved through its semi-realistic appearance of the interior and exterior, as well as the booths along the walls that the user needs to teleport to to see the contents of. The functionality of the ray gun and RT Camera provoke cognitive curiosity, especially if the user has some notion of what ray tracing is but doesn't necessarily know how it works in practice. Watching the RT Camera render images in its scan-line fashion can also appeal to curiosity, as the user watches

how the first noisy samples are taken and anticipates how the image will look as it gets refined over time.

While PathVis does have goals that the user is required to fulfill to proceed, it is unclear if this can be considered "gameplay". There is no score given for fulfilling goals and there is no clear skill that the user learns, other than being able to use the ray gun and RT camera. Educational games typically have some kind of fantasy element to make the experience more engaging. Some users wished there was more to PathVis, and expected more of a game after reaching the gallery room. PathVis in its current state is more of a demonstration of ray tracing elements, but it does let users experiment with the ray gun and picking up objects if they desire. Multiple users seemed to have fun playing around with the cubes in the tutorial, indicating that there is potential to use the grabbing mechanic as a fun gameplay element.

The question is how to design fun gameplay that also teaches ray tracing in line with the curriculum in computer graphics courses. The path tracing algorithm in PathVis is simple, lacking many of the more advanced concepts like shadow rays and bounding volume hierarchies (BVHs). A way to visualize a BVH was planned for the prototype, but later deemed out of scope. The author has a number of ideas for potential tasks that could be implemented to teach ray tracing in a more engaging way, which will be described in the following sections.

Playing with Bounding Volume Hierarchies

The purpose of BVHs is to reduce the number of intersection tests for a ray. To teach the concept to users, they could be tasked to define bounding volumes by drawing boxes within boxes with the VR controllers. A number of rays could then be shot into the scene, registering how many intersection tests were required and giving the user a score. This would challenge the user while simultaneously demonstrating how BVHs are used.

Playing with Reflection and Refraction

Reflection and refraction is already visualized in the current version of PathVis, appearing as rays bouncing or getting bent when hitting surfaces. Paths of rays are static, remaining in place after being fully drawn. For a user to experiment with different angles of reflection they need to keep firing the ray gun. One user expressed a wish for a laser pointer on the gun that would always be on, making it easier to aim. Such a "laser pointer" that traces a full path every frame can be used to showcase reflections in real time. The ray gun did function in this fashion during development, but with diffuse reflection, the traced paths would be different every frame, leading to an uncomfortable amount of visual noise.

A task that demonstrates reflection and refraction could be designed with a laser pointer that the user has to direct to a target, by placing reflective and refractive objects to direct the laser beam around obstacles. With the beam updating in real time, the user could gain an intuition for how a refractive object bends the beam depending on how the user holds the object with the controller. Another possible task to challenge the user could be a guessing game, where the user has to guess the correct angle of refraction or reflection by moving the tip of an arrow, or arrange elements in a booth to obtain a given effect.

6.6 Limitations

The interactive path tracing in PathVis relies on the Physics Raycast function in Unity, which tests for intersections on colliders rather than the meshes that are rendered in-game. Mesh colliders *can* use the same mesh as the rendered meshes, but colliders do not support interpolated surface normals which results in a faceted look when rendered with the PathVis path tracer. On the other hand, parametric colliders like sphere and capsule colliders can be rendered smoothly, but these colliders do not return any texture coordinates to the RaycastHit so they cannot be textured.

The algorithm of PathVis is a simplified version of the standalone ray tracer (3.2.1), and does not implement depth of field, antialiasing or volumetric materials. The standalone ray tracer was based on the Ray Tracing In One Weekend[18] book, and as such it is a naive brute force path tracer that does not shoot shadow rays or favor shooting rays towards light sources. In practice this means that scenes with a small light source will appear very noisy, as few paths end up hitting the light source. This makes PathVis unable to render scenes lit by a directional or point light source, e.g. a sunny day. The rendering engine of PathVis could be improved by using more sophisticated sampling methods.

The booths in PathVis were made with these limitations in mind: the light sources in each exhibit are large, none of the spheres have textures and the visual mesh of the Stanford bunny was made faceted to correspond with the path traced results. Objects with refractive materials were given an inner copy with inverted surface normals and IOR to enable refraction when a ray exits the object.

Performance

The PathVis path tracer runs as a script component on the RT Camera, which means that it executes in the main game thread. The first iterations of the rendering script traced paths for all pixels in the render texture every frame, but this only allowed for very low resolution images to be rendered. The rendering routine was made into a coroutine that yields control back to the main thread after rendering

a number of pixels in the render texture. The prototype version of PathVis had the coroutine render only one row of pixels each frame, but this lead to a slow RT camera and underutilized CPU performance. To make the RT camera faster, the coroutine was made to render rows of pixels until nearing a defined maximum frame time. This did indeed speed up rendering, but the timekeeping of the script is not precise enough and leads to the CPU exceeding the time budget, as shown in Figure 5.1b. When the application uses too much time to deliver a frame, it may be dropped and the VR runtime will reproject the last rendered frame to preserve the user's comfort.

One possibility of speeding up the path tracing is to Utilize the Unity C# job system, which offers multithreading for enhanced performance. The standalone ray tracer (3.2.1) saw increased performance when its rendering routine was parallelized by using multiple threads to render different pixels of the image. Parallelizing the rendering routine of PathVis might lead to issues or complexity that gets in the way of the interactivity, and PathVis is made to be a simple path tracer rather than a performant one.

6.7 Teaching Computer Graphics

The current version of PathVis could be expanded to include more concepts within computer graphics, especially those that are already used to some extent within PathVis, like UV coordinates and texture mapping. There may also be potential for visualizing basic concepts like vectors and vector operations; 2D vectors are easily visualized with traditional technology, but 3D vectors may be easier to understand when viewed in VR with depth perception. In addition, the interaction that VR offers can give users an engaging way to learn 3D vector operations. The tracked VR controllers can be used to define points and vectors that the user can play with and see the results of different vector operations, and how it changes as vectors are moved around.

To teach a wider range of topics, the addition of a menu system and level loading would be required, as there is currently no way to restart the application without exiting it completely. Instead of extending the PathVis application, it is also possible to bring just the interactive ray tracing functionality to other Unity projects; PathVis application was developed in Unity using the SteamVR plugin, but most of the interactive ray tracing functionality was implemented as scripts that are not dependent on the components from the plugin.

6.8 Fulfilling Requirements

This section discusses how the PathVis application fulfills the requirements as defined in Section 4.3.

Non-Functional Requirements

- **NFR1 (VR Framerate)** The environment in PathVis is designed as a static mesh with baked lighting, making the application run well on most GPUs. The framerate is smooth throughout the application, but slight judder and lag may be experienced when the RT camera is active. When lag happens, the VR runtime is responsible for reprojecting frames. No users expressed any discomfort relating to this lag.
- **NFR2 (Immersive Environment)** The virtual environment is designed as a hallway with windows showing a backdrop of a grassy field. Baked light and reflection probes give a semi-realistic appearance of the environment. Windows help make the hallway feel more open and less like a maze.
- **NFR3 (Unity Prefabs)** Prefabs were widely used during development to implement the goal and gate system, narration buttons, ray gun and RT camera retriever buttons and more. Once prefabs were in place, the original prefab could be modified and the changes would propagate to all the other linked prefabs.
- **NFR4 (Reusable Scripts)** Script components were made generalized where possible, such as the *RaytracingParticipator* component used on every object that the ray gun could hit. The component responsible for tracing visual rays for the ray gun was reused for the ray tracing introduction booth and the first version of the pixel grid camera. Some tasks and functionality in PathVis required new script components that were created in an ad-hoc fashion, as the functionality was not necessary in more than one area.

Functional Requirements

All the functional requirements are successfully fulfilled by the functionality available in PathVis. The ray gun fulfills **FR1** and **FR2**, with the RT camera fulfilling **FR3**. By developing the application for the SteamVR runtime, **FR4** is fulfilled as SteamVR works with most consumer VR HMDs.

Tutorial Functional Requirements

The fulfillment of these requirements was tested in the two user tests.

- **T-FR1** Users are taught to teleport by requiring them to do so to proceed. The first version of the tutorial had some users confused, but the second version worked better and all users managed to learn how to teleport.
- **T-FR2** Picking up objects is taught to the user by a holographic hand pointing towards a cube, indicating that the user needs to move the virtual hand close to the cube to grab it. Some users tried picking up the cube by squeezing it with both hands, but eventually all users managed to pick up the cube the correct way.
- **T-FR3** In order to proceed through the first gate in the tutorial, the user has to successfully teleport with an object. Users would try using the teleport

beam with the same hand they were holding an object with, but eventually learned to use the other hand.
- **T-FR4** Learning how to drop sticky objects was not clearly communicated in the first version of the tutorial. The second version added a picture indicating the button to use, and users managed to learn how to drop objects more quickly.
- **T-FR5** Narration buttons in the tutorial introduce the user to buttons, though they are not required to interact with them to proceed from the tutorial. However, interacting with buttons is done in much the same way as picking up an object, and users seemed to be able to press the buttons in the booth immediately after the tutorial.

6.9 Research Outcome

This section will discuss how the findings from developing and user testing the PathVis application can answer the four research questions as defined in Section 1.2

RQ1: How can virtual reality be used for teaching ray tracing?

The method used for teaching ray tracing in the PathVis application is to let a user explore a virtual environment and interactively trace rays by using a ray gun. By letting the user be in control of the act of tracing a ray, the experience is made more engaging.

RQ2: Can a ray tracing rendering algorithm be visualized in virtual reality?

The PathVis application can visualize a simple brute force path tracing rendering algorithm, by drawing paths of rays as lines in 3D space. Ray tracing and path tracing are closely related, with path tracing being a more specific use of ray tracing. The PathVis application also features a virtual camera that uses the same rendering algorithm to render pictures. A "pixel grid camera" is also featured, where the ray gun can be used to trace paths and storing the result in pixels, which visualizes the process of rendering a picture using ray tracing.

RQ3: What do users think about learning in virtual reality compared to traditional learning methods?

Users generally liked seeing the visual learning materials in the PathVis application, and felt that this type of learning could work well as an introduction to a topic before learning more via traditional learning methods. Users expressed that using the ray gun felt like a good way to learn ray tracing. Users that had never tried VR before quickly become comfortable with moving around in VR.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The current wave of VR technology has seen successful applications for teaching and visualizing. Ray tracing is a powerful method of rendering effects that are difficult or impossible with traditional rasterization techniques, and is widely used in the movie industry to render high quality. The project of this thesis was to implement a VR application that could visualize key concepts within ray tracing and evaluate how this could help with teaching.

A prototype application was developed but could not be evaluated by user testing due to the COVID-19 pandemic restrictions at the time. The prototype application featured a ray gun and a virtual camera that the user could interact with. The ray gun could visualize paths of rays, and the camera could render pictures of virtual "booths". In the second stage of development, the application was extended to feature a tutorial and further build upon the features from the prototype. The new environment was designed as a zig-zag hallway that introduced ray tracing concepts one by one and requiring the user to fulfill goals to proceed.

Two versions of the PathVis applications were tested by users, with feedback from the first user test being used to implement changes and improvements to the application before the second test. The user tests showed that there is great potential for using VR to teach ray tracing and other computer graphics topics, as the virtual environment can give a more intuitive understanding of concepts like rays and paths when they are visualized as lines in a 3D environment. Most users felt like they learned something from the application, and that this type of learning could be a good supplement to traditional methods.

7.2 Future Work

The PathVis application in its current version can act as a good introduction to ray tracing, and as a demonstration of a simple path tracing rendering algorithm. However, there are several areas that could be developed further.

Target Audience

The initial target audience of the application was to be students of computer graphics, but the participants of the user test were mostly people with a background in computer science but not necessarily graphics. For PathVis to be used as an educational tool for computer graphics courses, it would be beneficial to test the application with students from such a course. Feedback from a test like this can then be used to determine where to develop the application further.

Tasks and Gameplay

Tasks in the current version of PathVis are limited in scope, and most tasks do not test the user on their knowledge of the ray tracing concepts, instead focusing on demonstrating and letting the user experiment on their own. This makes it accessible to people who are not students of computer graphics, and it can serve as a good introduction for ray tracing. In order to teach more substantial subject matter, more engaging tasks and gameplay elements should be explored.

Standalone VR

PathVis was developed using the Unity game engine and SteamVR plugin, and can be used with any SteamVR compatible VR HMD. A PC with SteamVR installed is required to run the application, which makes the application less practical to use compared to the ease of use of a standalone VR system. As standalone VR hardware becomes more powerful, it should be possible to port some or all of the functionality of PathVis to an application that can run on a standalone VR device.

Bibliography

[1] B. Ashworth, *What is ray tracing? the latest gaming buzzword explained*, https://www.wired.com/story/what-is-ray-tracing/, Accessed: 2020-06-12, Jun. 2019.

[2] T. Whitted, "An improved illumination model for shaded display," in *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, 1979, p. 14.

[3] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984, pp. 137–145.

[4] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 143–150.

[5] L. Fascione, J. Hanika, R. Pieké, C. Hery, R. Villemin, T.-W. Schmidt, C. Kulla, D. Heckenberg, and A. Mazzone, "Path tracing in production-part 2: Making movies," in *ACM SIGGRAPH 2017 Courses*, 2017, pp. 1–32.

[6] V. Sanzharov, A. Gorbonosov, V. Frolov, and A. Voloboy, "Examination of the nvidia rtx," in *Proceedings of the 29th International Conference on Computer Graphics and Vision (GraphiCon 2019)*, vol. 2485, 2019, p. 7.

[7] *Nvidia turing architecture whitepaper*, https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, 2019.

[8] T. X.-K. Kong and A. M. Kruke, "Teaching computer science algorithms through virtual reality," M.S. thesis, NTNU, 2018.

[9] S. R. B. Hunvik, "Using virtual reality for artificial intelligence education," M.S. thesis, NTNU, 2020.

[10] R. S. Eriksen, *Pathvis*, /urlhttps://github.com/rikeri/Pathvis, Accessed: 2022-06-15.

[11] T. Mazuryk and M. Gervautz, "Virtual reality-history, applications, technology and future," 1996.

[12] H. McLellan, "Virtual realities," *Handbook of research for educational communications and technology*, pp. 457–487, 1996.

[13] C. Anthes, R. J. García-Hernández, M. Wiedemann, and D. Kranzlmüller, "State of the art of virtual reality technology," in *2016 IEEE Aerospace Conference*, IEEE, 2016, pp. 1–19.

[14] B. Lang, *Gdc 2014: Oculus rift developer kit 2 (dk2) pre-orders start today for $350, ships in july*, https://www.roadtovr.com/oculus-rift-developer-kit-2-dk2-pre-order-release-date-specs-gdc-2014/, Accessed: 2020-07-12, Mar. 2014.

[15] V. S. Dante D'Orazio, *Valve's vr headset is called the vive and it's made by htc*, https://www.theverge.com/2015/3/1/8127445/htc-vive-valve-vr-headset, Accessed: 2020-07-12, Mar. 2015.

[16] V. Angelov, E. Petkov, G. Shipkovenski, and T. Kalushkov, "Modern virtual reality headsets," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, IEEE, 2020, pp. 1–5.

[17] T. W. Malone, "What makes things fun to learn? heuristics for designing instructional computer games," in *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, 1980, pp. 162–169.

[18] P. Shirley, *Ray tracing in one weekend*, https://raytracing.github.io/books/RayTracingInOneWeekend.html, Oct. 2020.

[19] *Ray tracing: Rendering a triangle*, https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/, Accessed: 2020-12-05.

[20] S. J. Koppal, "Lambertian reflectance," in *Computer Vision: A Reference Guide*. Boston, MA: Springer US, 2014, pp. 441–443, ISBN: 978-0-387-31439-6. DOI: 10.1007/978-0-387-31439-6_534. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6_534.

[21] *Refractive index*, https://www.britannica.com/science/refractive-index, Accessed: 2020-12-06, Dec. 2019.

[22] T. Theoharis, G. Papaioannou, N. Platis, and N. M. Patrikalakis, *Graphics and visualization: principles & algorithms*. CrC Press, 2008.

[23] *Steamvr plugin*, https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647, Accessed: 2022-02-24.

[24] *Xr interaction toolkit*, https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/index.html, Accessed: 2022-06-02.

[25] P. Smutny, "Learning with virtual reality: A market analysis of educational and training applications," *Interactive Learning Environments*, pp. 1–14, 2022.

[26] R. Molina-Carmona, M. L. Pertegal-Felices, A. Jimeno-Morenilla, and H. Mora-Mora, "Virtual reality learning activities for multimedia students to enhance spatial ability," *Sustainability*, vol. 10, no. 4, p. 1074, 2018.

[27] W. Usher, P. Klacansky, F. Federer, P.-T. Bremer, A. Knoll, J. Yarch, A. Angelucci, and V. Pascucci, "A virtual reality visualization tool for neuron tracing," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 994–1003, 2017.

[28] X. Wang, C. Guo, D. A. Yuen, and G. Luo, "Geovreality: A computational interactive virtual reality visualization framework and workflow for geophysical research," *Physics of the Earth and Planetary Interiors*, vol. 298, p. 106 312, 2020.

[29] F. Grivokostopoulou, I. Perikos, and I. Hatzilygeroudis, "An innovative educational environment based on virtual reality and gamification for learning search algorithms," in *2016 IEEE Eighth International Conference on Technology for Education (T4E)*, IEEE, 2016, pp. 110–115.

[30] D. Hamilton, J. McKechnie, E. Edgerton, and C. Wilson, "Immersive virtual reality as a pedagogical tool in education: A systematic literature review of quantitative learning outcomes and experimental design," *Journal of Computers in Education*, vol. 8, no. 1, pp. 1–32, 2021.

[31] P. Shirley, *Ray tracing: The next week*, `https://raytracing.github.io/books/RayTracingTheNextWeek.html`, Oct. 2020.

[32] P. Shirley, *Ray tracing: The rest of your life*, `https://raytracing.github.io/books/RayTracingTheRestOfYourLife.html`, Oct. 2020.

[33] *System usability scale (sus)*, `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`, Accessed: 2022-03-15.

[34] *Unity scripting: Physics.raycast*, `https://docs.unity3d.com/ScriptReference/Physics.Raycast.html`, Accessed: 2020-09-29.

[35] *The stanford 3d scanning repository*, `http://graphics.stanford.edu/data/3Dscanrep/`, Accessed: 2020-12-04.

[36] *Kloppenheim 06 hdri*, `https://polyhaven.com/a/kloppenheim_06`, Accessed: 2022-03-28.

[37] *Measuring and interpreting system usability scale (sus)*, `https://uiuxtrend.com/measuring-system-usability-scale-sus/`, Accessed: 2022-03-15.

Appendix A

Survey

This chapter contains all the text and questions used in the second user test survey, since this survey was identical to the one used in the first user test, apart from the few additional questions relating to the first user test and narration buttons. Responses to the questions and statements could have one of three types:

- A binary choice between *Yes* and *No*.
- A 5-point scale from *Strongly disagree* to *Strongly agree*.
- A text field where the user could write a text response.

A.1 Second User Test Survey

PathVis user testing feedback

Thank you for trying out PathVis! Please give feedback from your experience with the app by filling out this online questionnaire.

PathVis was created by Rikard Storheil Eriksen for their master thesis project at the Norwegian University of Science and Technology (NTNU).

All collected data is anonymous and will be used for research purposes in the master thesis report.

- Did you participate in the first user test of PathVis (Friday April 1st 2022)? (Yes/No)
- Do you study or have you studied computer science? (Yes/No)
- Did you reach the final room in your playthrough of PathVis? (Yes/No)

We need to know about users' experience with VR and Computer Graphics. Please choose the options that describe you best.

- I have experience with using VR (5-point scale)
- I have knowledge and/or experience with 3D Computer Graphics (5-point scale)

- I think ray tracing is interesting (5-point scale)

Usability

How much do you agree with the following statements about the usability of PathVis?

- I think I would like to use this application frequently (5-point scale)
- I found the application unnecessarily complex (5-point scale)
- I thought the application was easy to use (5-point scale)
- I think that I would need the support of a technical person to be able to use this application (5-point scale)
- I found the various functions in this application were well integrated (5-point scale)
- I thought there was too much inconsistency in the application (5-point scale)
- I would imagine that most people would learn to use this application very quickly (5-point scale)
- I found the application cumbersome to use (5-point scale)
- I felt very confident using the application (5-point scale)
- I needed to learn a lot of things before I could get going with this application (5-point scale)

PathVis and learning

How much do you agree with the following statements about what you could learn from PathVis?

- I learned something from using PathVis (5-point scale)
- I think VR has potential for educational use (5-point scale)
- I feel like this way of learning is a good supplement to traditional learning methods (books, lectures, videos, etc.) (5-point scale)
- I have a better understanding of ray tracing after using PathVis (5-point scale)
- I think there was too much text to read in PathVis (5-point scale)
- I used the narration buttons to listen to the text instead of reading (5-point scale)

Open questions

Please answer each question as well as you can.

- What did you think about the tutorial area? Did you have any problems with getting to the next areas? (text response)
- What did you think about the ray gun? What did you like/dislike about it? (text response)
- What did you think about the ray tracing camera? What did you like/dislike about it? (text response)

- Did you use the narration buttons? If so, what did you like/dislike about the narration? (text response)
- Do you have any suggestions for how PathVis could be improved? (optional) (text response)

Appendix B

Evaluation

This chapter contains the extra results from the surveys that were used in the two user tests. The results for the general questions were presented in Chapter 5.

B.1 User Test 1

B.1.1 SUS Survey Results

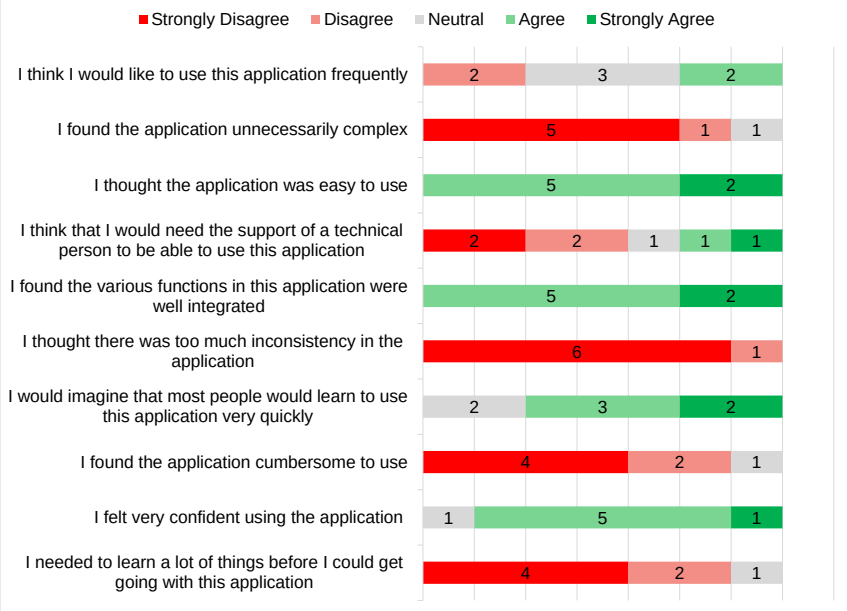


Figure B.1: The SUS scores from the first user test

B.1.2 Text Responses

The survey also included questions with text responses where users could describe their experiences of elements in PathVis. The first three questions were required, with the last question being optional.

What did you think about the tutorial area? Did you have any problems with getting to the next areas?

- Grabbing objects doesn't seem obvious when you never tried VR before
- It would've been nice to get a better introduction to the buttons available on the controller before using the headset. It's hard to figure out what buttons these things have if I can't see them :)
- It took me a bit to get the hang of it but it was on afterwards
- Decent, though I didn't realize where the grip button was at first
- It was well made
- The tutorial area was very friendly and relaxing. The movement from an area to the other was pretty easy and straightforward.
- The texts were easy to understand, not confusing. No issues on getting to the next area.
- Decent, though I didn't realize where the grip button was at first

What did you think about the ray gun? What did you like/dislike about it?

- Cute
- It's super intuitive to use once you know how to do it. Sometimes I wished I could shoot faster, i.e. that the rays don't take as much time traveling.
- I liked the tracing thing, but was not very sure on what to expect from it
- Excellent idea
- It was very useful and fun. I can imagine that people using this app will love to experiment with it. It provides a great idea regarding all the scattering that takes place in a 3D scene. Nothing to dislike really.
- The ray gun was cool. It was easy to handle and made the understanding of Ray casting interesting and easy to learn.
- I found it very cool, nice to see the way the light gets reflected and bent

What did you think about the ray tracing camera? What did you like/dislike about it?

- It's nice to see all the possibilities
- I didn't really find it useful for anything. I knew before how ray tracing works and I had coded some basic ray tracing algorithm before, so I didn't really have any benefit. (There's a lot of lower-level things about rendering which I don't know yet, but this was nothing new.)
- I liked it. I think it's neat to know what happens with actual light

- Some explanation of the different views would be nice, maybe something analogous to a tooltip that pops up as you cycle through the options
- The interface of the camera is very nice and easy to follow. I have a comment though for the cameras provided for the small exhibition rooms at the final area: I could not easily "move" the cameras in order to focus on the corresponding exhibition room. I have managed to do that one or two times but I could not understand how I actually did that.
- The on/off button of the camera was a bit confusing. In the beginning, it was not clear that the text underneath the button showed the current state (on/off). I really liked that you could take photos.
- The path tracing was rather slow :)))) I think it worked really well though

Do you have any suggestions for how PathVis could be improved? (optional)

- It explains ray tracing at one level of difficulty. All the concepts were explained in a simple way. It would be cool to start even simpler and then to dive even deeper into the topic, i.e. to explore it in multiple iterations. I'm unsure if this matches what you have in mind for the tutorial, but generally I find iterative learning approaches to be superior.
- Maybe add some form of instructions or directions in text on what button does what in the physical controller
- If you could somehow reduce the amount of reading you have to do. I find it a bit frustrating to read in VR. Narration could maybe be an option.
- Same as my latest comments. It would be good if there was an explicit moving manipulation for the cameras at the final area so that the user can easily focus the camera to the desired exhibition scene.
- Overall it was a great app. A good detail was the windows with the open sky. Two suggestions could be: make a clear suggestion to the user to print more photos at the last room. Highlight the end of the process (eg an exit sign)
- Clarify distinction between path tracing and ray tracing

B.2 User Test 2

B.2.1 SUS Survey Results

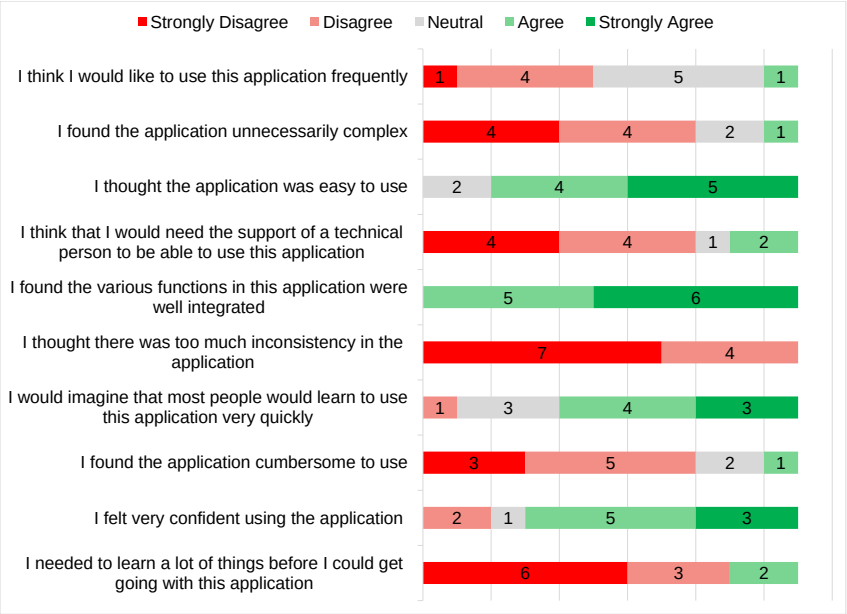


Figure B.2: The SUS scores from the second user test

B.2.2 Text Responses

The second user test asked the same questions, with an additional one relating to the narration buttons.

What did you think about the tutorial area? Did you have any problems with getting to the next area?

- No
- It was easy to get started. A nice addition could be a visualization of the VR controller with instructions on how to use each button.
- It was ok. A bit or feedback when you fail a task would be nice. For example if you're not hitting the same spot every time
- Path is straightforwardn no problem to go to the next area
- I already have experience with VR, so no

- The tutorial was improved. Nice to see visuals in addition to text. I do see that you got quite insistent in telling people about the grip button.
- No, there were no problems. Sometimes it would've been nice to know the task not just at the end of the room, but rather at the beginning, but I wouldn't call this a problem. I just had to teleport back and forth one time more often than I could've.
- It was clear what I was supposed to do and what tasks I had to complete.
- I had no problems here.
- Once I figured out how to use the teleporter, I had no problems.
- I strongly suggest, It should be a small introduction about the controls. I was struggling to find the buttons in the control. Before putting the glass, I preferred to look at it and try it. There are a lot of unnecessary texts that I was confused what is the goal now.

What did you think about the ray gun? What did you like/dislike about it?

- I liked it, but maybe a light/translucent "laser pointer" that is always on would make it easier to shoot it accurately.
- Liked it, nothing to complain about! Everything is more fun when you get to shoot things :)
  At the start I was a bit confused of the different colors that the rays got. E.g. at one point I forgot the normal vector was pink. Perhaps you could separate them with something else than only colors? For example, dotted line.
- it's easy to use and the fact that you can get it when you need it is quite useful
- Easy to use, good to experiment with while listening to explanations
- I'm not sure if the ray colors were intuitive all the time
- Returning black rays from areas with color seemed odd, but most of the time it was fine.
- The gun worked flawlessly.
  Generally it would be nice to be able to hold something and have a gun and teleport myself, but well, I only have two hands.
- It was good
- I liked it. Good example of theory
- Nice 3D model and good responsiveness to the control.
- It was really interesting. With ray gun I start to like the test. I had more confident for next levels.

What did you think about the ray tracing camera? What did you like/dislike about it?

- It feels a bit clunky at times with the manual pointers that must be interacted with through vr only (no hotkeys), but otherwise good. You could also maybe do more with it, e.g. create some task for the user that incentives them to use it for more than just taking a photo.

- Cool! the Path tracing was a bit slow, so it ended up that I often did not wait for it to "finish" rendering.
- I likes that you could switch modes. I don't like that you need to grab a picture before printing another one
- Nice but repetitive
- As a still image it was good, but it was hard to get different angles
- A little slow :P It's cool though, and I like that you can adjust the perspective easily.
- I think it's the right concept to explain ray tracing. I wonder if it can be extended a bit. For example, there was still a gap between me shooting the rays through the grid and the camera doing everything autonomously. Maybe add an animation where the camera shoots all the rays?
- I think it had a great performance, especially in comparison with the previous run.
- Not as meaningful as the ray gun, but a cool gadget.
- Like: The ball to change the focal length Dislike: Slow sampling
- The button for camera was behind a lot of other options. At first I was confused that what I should do in this step and asked for guide.

Did you use the narration buttons? If so, what did you like/dislike about the narration?

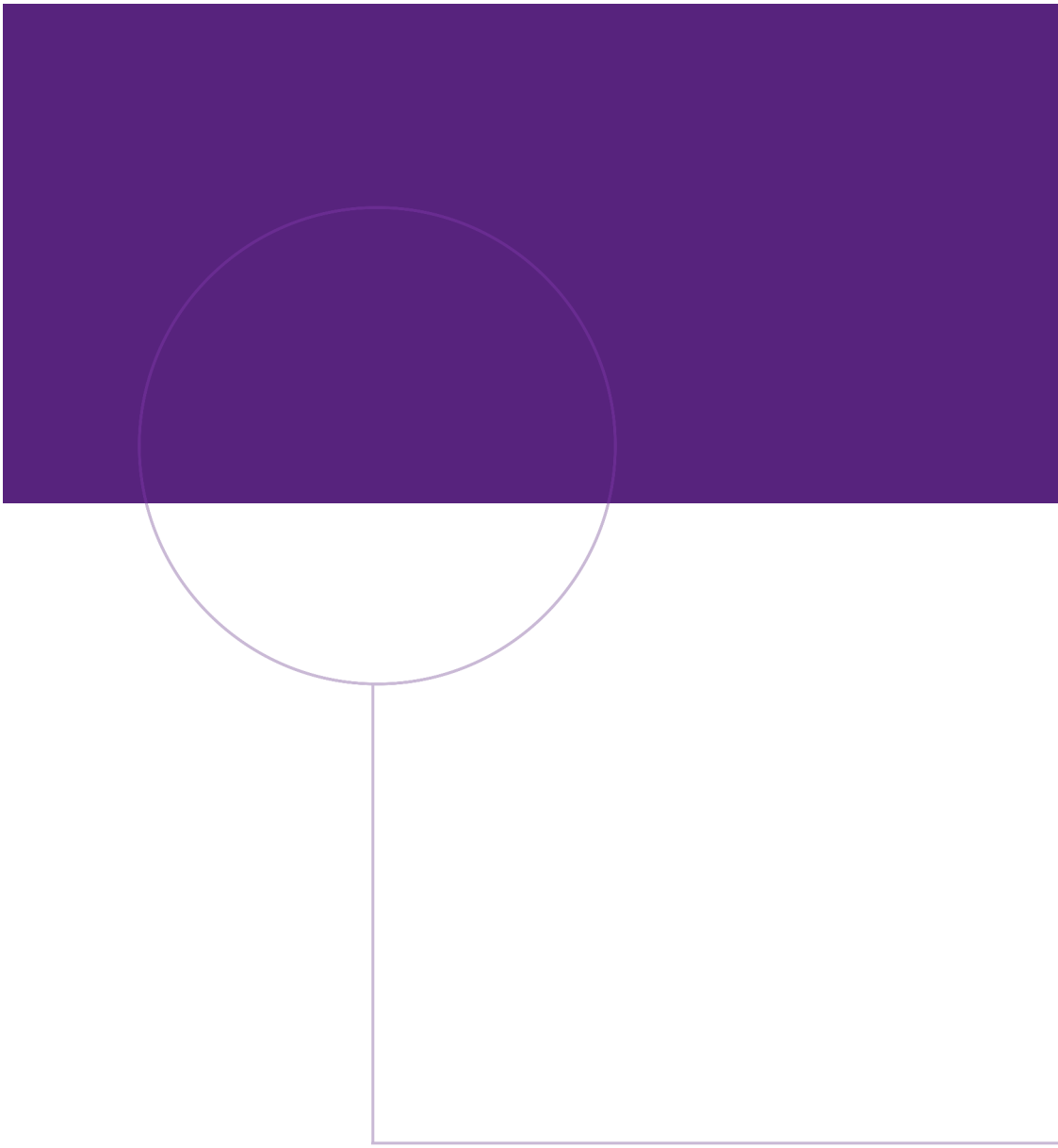
- Narration was solid! Maybe add some hotkey to rewind x seconds in case you miss something.
- Used it once or twice and it was good!
- I liked them because it makes the run faster. They complement the whole experience
- Reading in VR can be a bit annoying especially with glasses, narration makes it easier
- Not really
- Get a better mic... or just get some professional to do the narration. It's fine though.
- I think it's very useful for people who prefer to listen to content rather than reading it, so it's cool that the buttons were there. I'm not that person.
- I tried to, but I could not listen anything
- No
- Yes, I liked the narration button and used it a lot. This helped me because sometimes the text didn't look very sharp (possibly a contact-lens-related problem?).
- I used but then I turned off. I found it a lot of details. I preferred to read

Do you have any suggestions for how PathVis could be improved? (optional)

- It feels like a tutorial, but when you get to the end, there's no "game". If you ever have the time, maybe consider adding some sandbox-like area in

the end where the user gets to do more. For instance, some way to create your own scene would be nice. You could motivate them to play around by giving them more complex/open ended tasks than the simple ones required to get through the initial areas.

- Nothing I can think of now, but I think this is a very good application for using VR in education! Often the VR applications for education could easily been done in a different format, but using this to teach graphics/path tracing was a very good use case :)
- The wood room where I had to shoot three rays at the exact same place was hard. I think it could benefit from less precision
- Reduce the sensitivity in the "shoot rays at exact same spot" part. Took quite a few tries to get it right.
- Just the one suggestion for the camera two questions earlier.
- - I couldn't clearly identify the vector directions in the early experiments. - Being new to the VR stuff: I was overwhelmed by the functions and information input (teleporting, looking for the narration button and trying to activate it, the virtual fence) -> Most of my attention was on the "gameplay" instead of learning about the rays. - The reflected rays from the ray gun looked 2D to me (similar problem as mentioned before with the vectors)
- I preferred some introduction or one test as a demo to explain the test better. As a user the first time and especially at first was a bit confusing but at the end I was so fast and knew what I need to do. I should mention I don have much experience with VR and gaming.



NTNU

Norwegian University of
Science and Technology