OCULUS RIFT STARRING NATURE

By

GC Hussain Murad

NC Mahnoor Ehtisham

NC Noman Ali Bajwa

NC Zainab Batool

Project Supervisor: Lt. Col Dr.Hasnat Khurshid

Submitted to
Faculty of Department of Electrical Engineering,
Military College of Signals, National University of Science and Technology, Islamabad,
in partial fulfillment for the requirements of the degree of

B.E. in Electrical (Telecommunication) Engineering

JULY 2020

DECLARATION OF ORIGINALITY

We hereby declare that the work contained in this report and the intellectual content of tis report are the product of our work. This thesis report has not been formerly published in any structure nor does it include any verbatim of the published resources which could be treated as violation of the international copyright decree. We also affirm that we do recognize the terms ‘plagiarism’ and ‘copyright’ and that in case of any copyright infringement or plagiarism established in the thesis, we will be held fully accountable of the consequences of any such violation.

CERTIFICATE OF APPROVAL AND CORRECTNESS

This is to officially state that the thesis work contained in this report titled "OCULUS RIFT STARRING NATURE" is carried out by 00000186097 GC Hussain Murad, 00000186814 NC Mahnoor Ehtisham, 00000201401 NC Noman Ali Bajwa, 00000175895 NC Zainab Batool under my supervision and that in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Electrical Engineering from Military College Of Signals, National University of Sciences and Technology (NUST) and is original with 9% of plagiarism only.

Approved by:

Lt Col Dr.Hasnat Khurshid

Military College of Signals (MCS),

National University of Sciences and Technology (NUST),

Islamabad, Pakistan

ABSTRACT

Exercise on treadmill is a boring activity and is usually augmented by installing LCD television sets in front of the treadmill. LCD sets do not provide VR experience and there is a requirement of a 3 Dimensional and 360-degree VR environment for a treadmill user. This work aims at creation of an immersive video where viewers can pan around a full 360 degrees by moving their heads inside Oculus Rift. The technical problem to be solved is concerned with making videos for Oculus Rift that can engage as well as immerse viewers in virtual reality and creation of complete VR environment via software.

The project is implemented using Unity software and all coding of virtual reality environment is done using C#. The Head Mounted Display used to view the 3D 360 video is Oculus Rift. Videos are shoot using Samsung Vuze 3D 360 camera. A virtual reality application is developed for Windows using unity software.

The project will be of great interest for gymnasium owners who can attract more people to their gyms by providing a virtual reality experience. It will also be of interest for the people who use treadmills at home and find it laborious, by using this app they will be able to enjoy the tedious walking hours. Moreover this work will be helpful in future advancements in the field of virtual reality.

Dedicated to our beloved parents and our supervisor, who proved to be a source of motivation for us. It is their unconditional support and words of encouragement that led us to complete a work of such magnitude.

.

ACKNOWLEDGEMENTS

We would first like to thank Allah Almighty for bestowing upon us His countless blessings and without whom nothing is possible.

We are extremely thankful to our project supervisor Lt Col Dr.Hasnat Khurshid from Military College of Signals (MCS) who in addition to providing us with valuable technical help and guidance also provided us with moral support and encouragement throughout the development of the project.

Our special gratitude and acknowledgments are there for the faculty and staff of MCS who supported and guided us during our course work. Their advice, knowledge insight, training and guidance facilitated us in completing this difficult task.

We would also like to extent our gratitude to our families, friends, colleagues and a long list of well-wishers whose prayers and faith in us propelled us towards our goal.

Finally we are indebted to Electrical (Telecomm) Department of the Military College of Signals, NUST.

TABLE OF CONTENTS

TABLE OF FIGURES

ABBREVIATIONS

VR	Virtual Reality
3D	3-Dimensional
IVE	Integrated Virtual Environment
AR	Augmented Reality
MR	Mixed Reality
HTC	High Tech Computer Corporation
PC	Personal Computer
XR	Extended Reality
DoF	Degree of Freedom
SDK	Software Development Kit
GUI	Graphical User Interface
HMD	Head Mounted Display
LED	Light Emitting Diode
OS	Operating System
UHD	Ultra High Density
GPU	Graphic Processing Unit
UI	User Interface
GUI	Graphical User Interface

CHAPTER 1

1. INTRODUCTION

1.1. Overview

This is an era of innovation. Science, technology, and research has been the major interest of everyone. Every second something striking is being created in the name of science and technology in the neck of the woods and it is unpredictable. These developments and discoveries are considered as an achievement. From typewriters to Supercomputer the world has changed drastically. Now the artificial intelligence and Virtual Reality (VR) has emerged as a new trend. Virtual reality is a technology that makes people believe the unbelievable. This technology is ready to take over the world and fragment the definition of reality. The thinking of the human mind has become possible in the real world with the help of virtual reality.

Oculus Rift Starring Nature is one such project that will make humans experience the most exotic rides while passing their monotonous treadmill hours. The project will be capable of creating a complete 3-Dimensional (3D) immersive environment where users can pan around a full 360-degrees by moving their head inside Oculus Rift.

1.2. Problem Statement

Any type of exercise machine that challenges the cardio-respiratory system serves as a great way to boost energy levels, strengthening the condition of heart and lungs, bringing more blood to the muscles, brain and rest of the body as well as leading to greater energy all day long. But walking on a treadmill sometimes seem more repetitious and is usually augmented by installing LED televisions in front of treadmill but for a better experience a person needs to find a machine that not only meet the level of fitness and impact but also provides entertainment because a person only gets energy from doing something that he enjoys. Therefore, there is a requirement of a 3-Dimensional and a 360-degree VR environment for a treadmill user that makes his tedious treadmill hour fun.

1.3. Approach

For the development of a complete 3-Dimensional 360-degree environment for users Unity software has been used. All the coding is done using C# programming language. A Virtual Reality application is developed for windows. The headset that is used for getting the output is Oculus Rift along with its sensors and controllers. The Oculus Integration has been imported in Unity software for the interaction of the headset with the computer. Video shooting is done using a 3-Dimensional, 360-degree Samsung Vuze camera and Vuze application is used for the stitching and the orientation of the video in a 3D plane.

1.4. Scope

This project has a very vast scope in gymnasiums where hundreds of people come regularly to keep themselves fit and want to escape from the reality in the middle of a tough workout. Huffing and puffing on treadmill surrounded by hordes of other people counting down the minutes until they have met their goal, is rarely an exciting experience. Immersing in virtual reality can be quite appealing for gymnasts as well as it can help the gym owners attract more people. Moreover, the project finds its scope for travelers who love to explore new places but due to limited resources they cannot afford to go everywhere. The VR application can help them survey these places while staying at their homes.

1.5. Aim & Objectives

Virtual reality is an immersive adventure that digitally invigorate a distant environment. VR applications have been spread in diverse domains such as entertainment, rehabilitation, engineering and robotics, fine arts, archeology and digital marketing.

The main objectives that this project focuses to achieve are the following:

- Invigorating 3-Dimensional walk for a treadmill user.
- Creation of an immersive 360-degree view. Capturing the breath-taking sceneries and acquainting them.
- Increasing the immersion and sense of presence when using Oculus headset in Integrated Virtual Environment (IVEs).
- Development of a virtual reality stand-alone application for windows operating system.
- Making use of Unity and C# language to create a virtual environment.

1.6. Contributions

This project is designed and developed mainly for the gymnasiums and as such would contribute the most towards social, environmental and global issues.

1.7. Organization

The first part of the thesis is the abstract which gives a basic insight of what the project is about. It is the crux of the thesis.

The division of the chapters and the material included in them are as under:

Chapter 1 This is the introduction section where a brief overview of the project has been given. It deals with the problem statement, approach to be followed, scope and objectives of the project.

Chapter 2 This section provides us with the literature review regarding how VR has evolved in 20th and 21st century. The concept of Augmented Reality (AR), Mixed Reality (MR) and how they differ from VR has also been explained. Moreover, the components of Virtual Reality have also been highlighted.

Chapter 3 The design and the development part demonstrates the flow chart exhibiting the various stages involved in this project. Also included is a description of the hardware and software requirements of the project.

Chapter 4 The analysis and evaluation part gives a detail of the techniques used to create an immersive environment. The working of Windows application has also been depicted.

Chapter 5 The future work gives further improvements that can be incorporated in the existing project and the additional developments that can be made to enhance the scope of the project.

The conclusion, appendices and bibliography are given at the end of the document.

CHAPTER 2

2. LITERATURE REVIEW

2.1. Introduction

An extensive research was conducted on our part before undertaking the design and development phase of the project. This section deals with a brief review of what Virtual Reality is, how it is different from augmented reality and mixed reality, and the work already done in the field of Virtual Reality.

2.2. Virtual Reality

The term Virtual Reality is made out of two distinct words 'Virtual' and 'Reality', where processing meaning of Virtual methods not genuinely gives us the exact same feel but all things considered yet gives an impression of physical measurements utilizing PC projects. Reality is what is knowledgeable about the sense organs. So Virtual Reality is a figuring aptitude that can give the vibe of reality, of any for all intents and purposes made condition. In fact, Virtual Reality is a 3-dimensional 360-degree, PC created condition which can be experienced and collaborated by an individual with the assistance of physical gadgets, for example, headset and controllers

Virtual Reality has allowed the users to immerse themselves in an artificial environment simulated by computer. This system let the user navigate through the environment and interact with the elements created. The user experiences a different degree of immersion, depending on the hardware used to generate this environment and the software that generates the simulation of this environment. One promise of VR is to rearrange the way people and digital word interact with each other. This one can possibly open better approaches for collaboration that are presently unrealistic. The creation of new experience with VR has opened new expectations towards how to use the current technology. An example of promise of VR is the possibility of using VR to take virtual tours of distant locations.

2.2.1. Virtual Reality in 20th Century

The term computer generated reality was broadly utilized in mid 1980s when diverse sort of innovation like gloves, goggles, and riggings to encounter augmented reality were being created. In any case, before that there were numerous researchers who were building up a mimicked situation. Among them, Sensorama in 1956 was a major accomplishment in the field of computer-generated reality when Morton Heilig needed to perceive how individuals feel to encounter his film in genuine. The Sensorama reenacted a genuine city experience riding a bicycle utilizing distinctive multi-tactile reproduction. He likewise planned the main head mounted showcase gadget which is the establishment of the apparatuses being grown at this point. In 1965 a PC researcher names Ivan Sutherland built up an HMD that owes an obligation of appreciation from the current designers of Virtual Reality. (Dormehl, 2017).

2.2.2. Virtual Reality in 21st Century

Augmented Reality in the 21st century is developing at a quick speed. The accessibility of reasonable cell phones has helped in the propelled improvement of convenient Virtual Reality gadgets. The organizations like Google, Facebook and Samsung are taking a shot at Virtual Reality. Microsoft, HTC (High Tech Computer Corporation), Sony are likewise attempting to bring their innovation with Virtual Reality applications.

The principal decade of the 21st century is known as the VR winter. There was not much media consideration until organizations like Oculus or HTC began giving individual VR headset, yet the innovation has just taken its pace since the start of the 21st century. Since the significant result of VR is the headset, numerous organizations are creating distinctive sort of headsets. Present day headsets would now be able to be described in three kinds:

versatile, fastened, and independent. Portable headsets utilize a cell phone to transform it into a VR gadget. These headsets do not require any additional links. Fastened gadgets like Oculus Rift and HTC Vive are associated with the PC (Personal Computer) with the assistance of a link which can give a total VR experience. Independent VR headset is autonomous of outside gadgets however gives a constrained ability when contrasted with fastened headsets.

2.3. Virtual Reality verses Augmented Reality

Augment means to include or build the benefit of something by adding something to it. So Augmented Reality is a PC innovation that creates the new computerized condition adding something to the article that dwells in reality. Enlarged reality manages the genuine object of this present reality utilizing diverse AR cameras of the cell phones or by including PC vision and gives a vibe of another condition with improved sound, video, and picture to the clients. While Virtual Reality gives an altogether new encounter of the virtual world, Augmented Reality accentuates this present reality adding something to it. (Animate, 2016)

2.4. Virtual Reality verses Mixed Reality

Mixed Reality is a mix of Virtual Reality and Augmented Reality to give an encounter of a virtual domain as well as give connection between the virtual items and the client. MR makes the chance of communications between the physical and computerized universes. (Rogers, 2018)
Extended Reality (XR) is the umbrella term utilized for various sort of advances that may occur later that are comparable yet further developed then Virtual Reality and Augmented Reality. XR covers every one of those spectra of Real and Virtual conditions.

2.5. Components of Virtual Reality

The essential part of each PC innovation is equipment and programming. Computer generated Reality is additionally made from programming and equipment like some other processing gadget. Equipment is the physical segment utilized for the distinctive working of PC innovation. In Virtual Reality, this equipment is utilized to deliver upgrades that supersedes the feeling of the client. VR gadgets comprise of info and yield gadgets and sensors which help in the connection between physical gadgets and clients. PC, cell phones and consoles are significant parts of Virtual Reality.

VR association is intended to inundate the clients into the virtual world to give the impression of reality however much as could reasonably be expected. The total drenching into the virtual world requires phenomenal coordination of physical body developments, input information, handling units and other equipment sensors.

Since VR motor can enter information from different sources, for example, body signals, discourse, locating the correct coordination between segments is imperative.

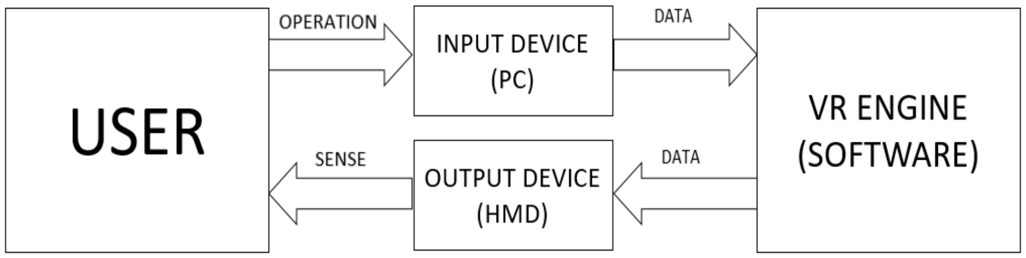


Figure 2.1: Components of Virtual Reality

2.6. Input Devices

The physical gadgets that are utilized to enter information or data in computers are called input gadgets. Info gadgets in VR are utilized to give a feeling of submersion to the clients. The information gadgets in this innovation are limited to the physical gadgets as well as incorporate pieces of human body and exercises, for example, vision, hearing, development, breathing and pulses. With the assistance of information gadgets client can control himself and explore in the virtual world.

Information gadgets are straightforwardly identified with the sensors and as a rule sensor themselves fill in as info gadgets to procure data from the development of the client. Figure 2.2 shows input gadgets from HTC and Oculus.

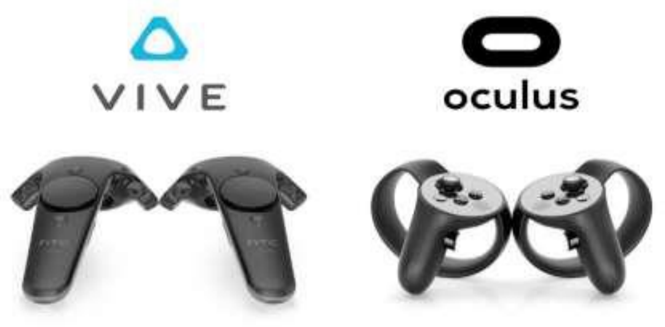


Figure 2.2: Input Devices from HTC and Oculus

2.7. Handheld Controllers

The same number of organizations are making VR headsets, a ton of headway has been made in innovation of info gadgets. Since a major piece of VR is created by gaming organizations so Joystick has been utilized as a typical information gadget however now Virtual Reality is moving towards a free world and utilization of Joystick is practically out of date and rather supplanted by free wandering gadgets. The level of opportunity is basic in VR innovation as it identifies with the development of an article in the virtual world. For the cooperation of client and article in virtual world two tomahawks, rotational and translational, are utilized. Three rotational tomahawks and three directional tomahawks make the level of opportunity regularly known as Degree of Freedom (DoF). 6 DoF permits a client to follow both translational and rotational developments, forward or in reverse, along the side or vertically, all over. These give an increasingly vivid and practical view to the client.

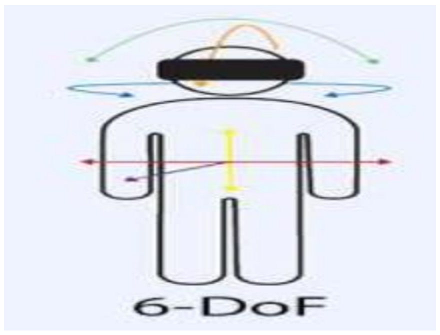


Figure 2.3: 6-DoF (rotational and translational movement)

2.7.1. Output Devices

Each PC innovation has a yield gadget to show the outcomes carefully in various structures like sound and video. Yield gadgets utilized in Virtual Reality incorporate video show screen and a headset to picture the virtual condition. Yield gadgets animate the sense organs introducing the virtual condition to the client producing a vivid inclination. Huge organizations like Oculus, Samsung, HTC, and Microsoft are creating propelled headsets for a superior encounter for the client.

2.7.2. Visuals

The fundamental factor in Virtual Reality is the making of a artificial environment in a reasonable manner with the goal that the client can encounter a genuine virtual world. Visual registering is a term utilized in software engineering that handles each visual substance saw on screens. 3D models, for example, PC illustrations, AI, video preparing, human PC connection has been made conceivable because of visual registering. The essential and straightforward visual creation to make a vivid encounter is 360-degree video. Typically, recordings are seen in two measurements so 3D displaying is utilized to make an item in three measurements utilizing particular programming.

2.7.3. Software

Software can be characterized as a computer program or a lot of directions that help a computer to do an assignment to secure a specific outcome rather than a related equipment gadget. As the VR advertise is growing quickly there are a ton of apparatuses and stages accessible to the engineers. The most usually utilized work area apparatuses for Virtual Reality are Unity 3D and Unreal motor. Numerous Software Development Kits (SDKs) are accessible with these products that help in better cooperation among programming and equipment.

2.8. Human Perception

The job of humans in VR is not only a client yet he is additionally a piece of the virtual condition. The fundamental motivation behind VR is to inundate into a virtual domain keeping up a harmony between with the assistance of equipment and programming parts. There are various perspectives on people with respect to Virtual Reality innovation. It is essential to comprehend the physiology of human body to make an optical figment or submerge the sense organs of people so most extreme sentiment of reality can be experienced. It is significant that the framework synchronizes all boosts with clients' activity for legitimate working of VR framework. (Guenard, 2017)

CHAPTER 3

3. DESIGN AND DEVELOPMENT

3.1. Introduction

This section clarifies the undertaking design and venture specifications including the equipment required and programming required for the implementation of this project. It explains in detail the proposed methodology and how the application will work. Furthermore, the brief description of the programming language used is given at the end of the chapter.

3.2. Project Design

The client provides an order to the Graphical User Interface made for the Windows application, the Oculus SDK senses the data from the controllers and accordingly sends the command to the game engine which in this case is Unity 3D software. The software executes the instructions in accordance with the scripts running in background and opens a virtual scene that is demanded by the user (GawnYong Park, 2017). Unity with the help of Oculus SDK and the Oculus sensors, senses the position of the user and sends the data to the Head Mounted Display (HMD) where the user is able to view the 3D 360-degree video in the headset.

The schematic outline of the venture configuration is delineated in the figure 3.1.

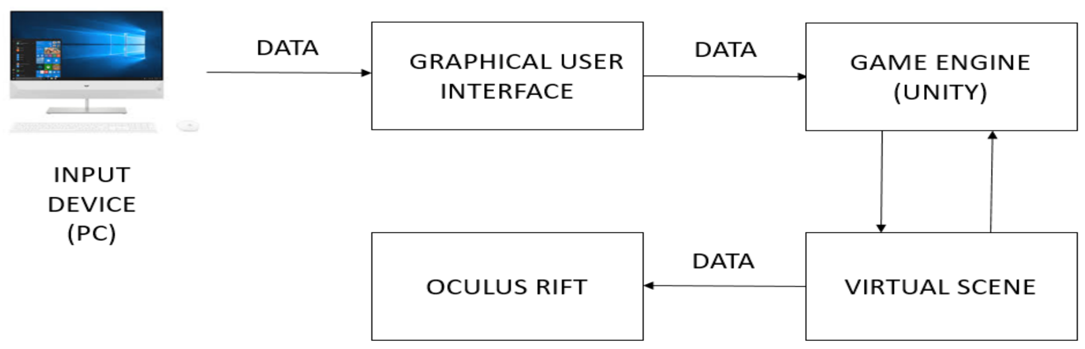


Figure 3.1: Project Design

3.3. Hardware Specification

The following hardware has been used for the implementation of the design depicted in figure 3.1

- Oculus Rift
- Vuze 3D 360 camera
- Computer

The description of each device is given below

3.3.1. Oculus Rift

An Oculus Rift is a head-piece equipment for display produced by Oculus' VR group. It is expressly planned Virtual Reality show that utilizes ultra-current showcase frameworks, optics, and revive/refresh rates to give a significant level of visual accuracy and a vivid wide field of view. It has a latest used technology for displays joined with a low inertness following framework to give an actual existence like understanding to the user. It fills in as a wearable headset fit for letting the clients experience a virtual domain.

The specifications of Oculus Rift are

- Head-gear display

- Light Emitting Diode (LED) display technology
- 2160 × 1200 resolution
- 90 Hz fast refresh rate
- 110 degrees wide view field (nominal)
- Microsoft Windows compatibility (plans for Operating System (OS) X and Linux compatibility)

The Oculus Rift comprises of a coordinated sound VR that gives a 3D sound impact. It additionally incorporates rotational and positional tracking skilled with the assistance of infrared sensors. In this manner it very well may be utilized while clients are sitting, standing, or trekking.

Figure 3.2 shows the Oculus Rift along with its sensors and controllers.

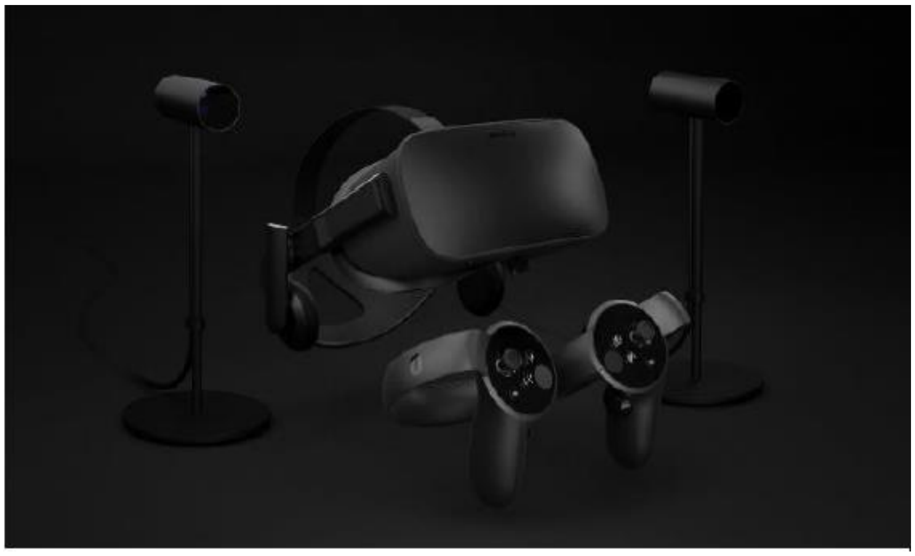


Figure 3.2: Oculus Rift, Sensors and Controllers

3.3.2. Samsung Vuze Camera

The camera that has been used to acquire immersive 360-degree videos is the Vuze 360 camera. It features eight cameras spread out evenly in pairs across its four sides. Each camera pair is intended to mimic human eyesight and each individual camera image sensor can record Ultra-high definition (UHD) 4K video at 30p. It also contains four built-in microphones for capturing audio.

Figure 3.3 shows the Vuze 360 camera

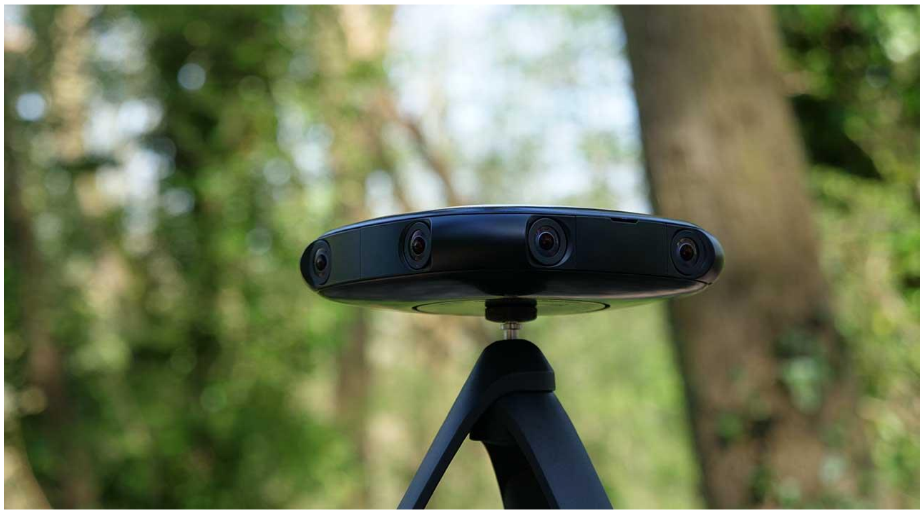


Figure 3.3: Samsung Vuze 360 Camera

3.3.3. Core i7 Processor with GPU

Since VR gaming and video shooting is very demanding therefore a powerful gaming desktop is required to run the tiles smoothly on the Oculus Rift. Graphic card plays the most vital role in providing VR experience hence a discrete video card is required that is powerful enough to drive VR.

The processor used in this project is Intel core i7 with NVidia GeForce GTX 1060 Graphic Processing Unit (GPU). It provides 90 fps frame rate. Since frame rates that waffle below this mark can be nauseating for some users so this consideration has been considered while choosing the hardware.

3.4. Software Specification

The following software has been used for the implementation of the design depicted in figure 3.1.

- Unity 3D
- Samsung Vuze phone application
- Vuze VR studio

The description of the software is given below

3.4.1. Unity 3D

The software utilized in this project for the advancement of a 3D domain is Unity 3D since it is the most generally utilized platform for VR development. Unity has a profoundly enhanced rendering pipeline and fast iteration capacities that makes innovative XR vision a reality. It supports all the latest and greatest platform including Oculus Rift. It has super high frame rates and the tools further optimize the stereoscopic rendering pipeline (Moreno, 2017).

Initially when Unity 3D is opened it has the structure shown below in figure 3.4.

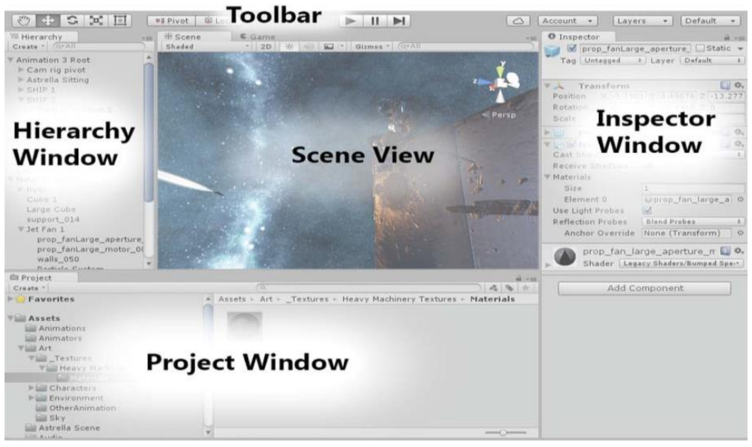


Figure 3.4: Unity 3D Editor

The components of Unity are explained on next page.

3.4.1.1. Hierarchy Window

The hierarchy window contains all the game articles in the present scene. A portion of these are immediate cases of the asset records like 3D models and other are cases of prefabs. It is conceivable to choose objects in the hierarchy window and drag one item on to the next to utilize parenting. As items are included the scene, they show up in the hierarchy chain too.

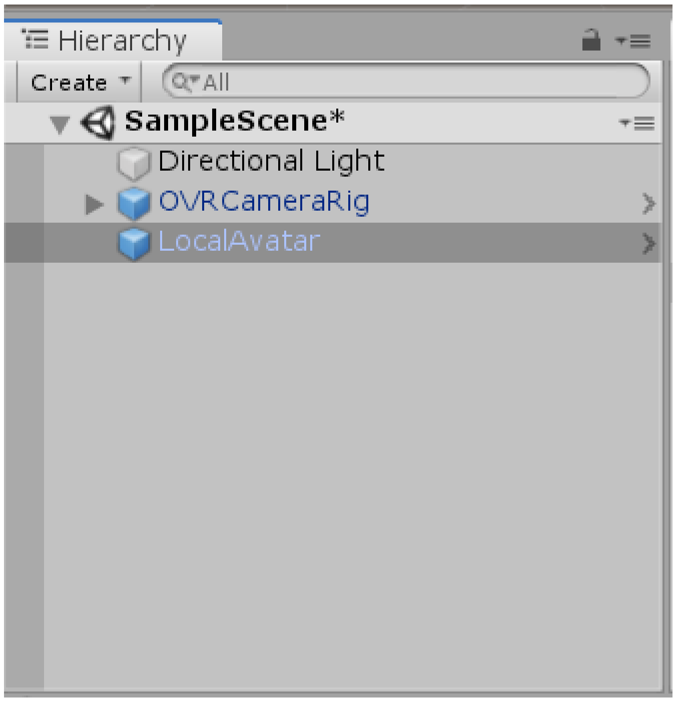


Figure 3.5: Hierarchy Window

3.4.1.2. Scene View

This window is the intuitive view in the designed world. Whatever is displayed in this window can be viewed in the head mounted display. It utilizes view to choose and situate landscape, camera lights, characters, and all other sort of game items. Objects can be manipulated and modified in this view.

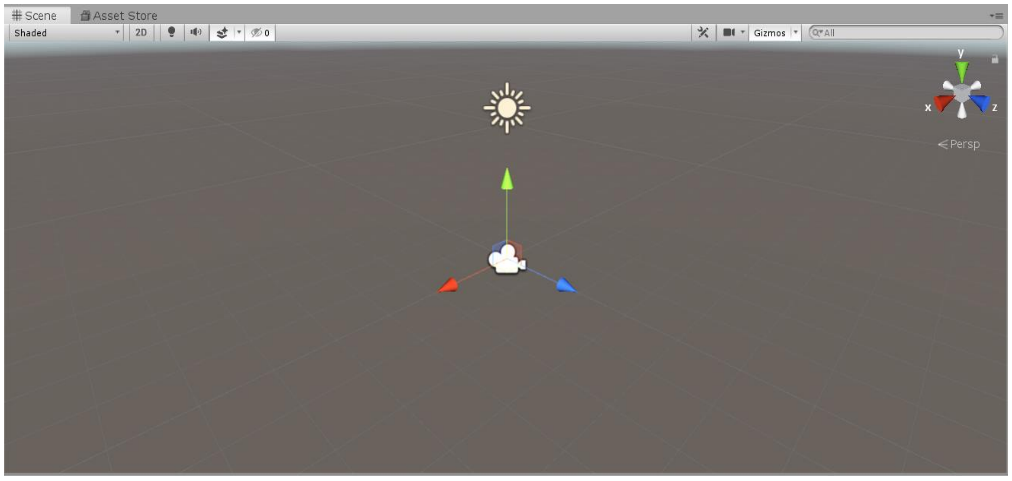
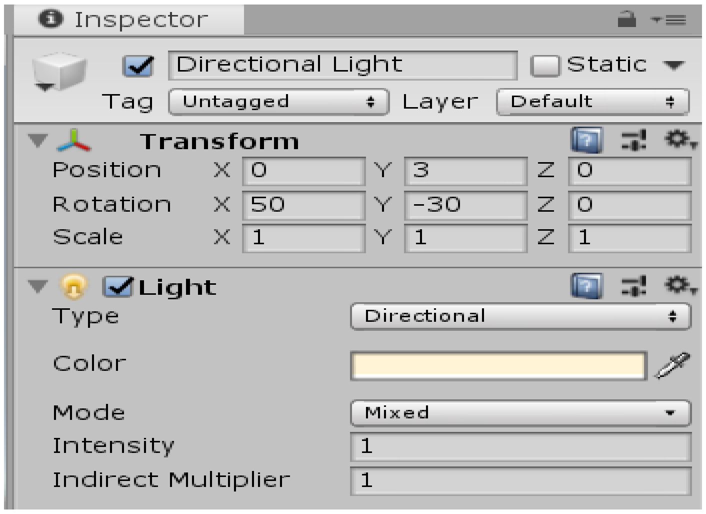


Figure 3.6: Scene View

3.4.1.3. Inspector Window

The inspector window is utilized to alter and see the characteristics of Game Objects, Assets and different inclinations and settings in the editor. At the point when a game article is chosen in the hierarchy window or scene, inspector will show the component's properties and materials on that object.



Inspector

Directional Light | Static
Tag: Untagged | Layer: Default

Transform

	X	Y	Z
Position	0	3	0
Rotation	50	-30	0
Scale	1	1	1

Light

Type	Directional
Color	
Mode	Mixed
Intensity	1
Indirect Multiplier	1

Figure 3.7: Inspector Window

3.4.1.4. Game Window

The game view is provided from the camera in the project. It is the final portrayal of how the project looks and functions. It is important to utilize at least more than one camera to control what the player really observes when they are running the venture. When in play mode any progressions made are temporary and will be reset when leave play mode. The Editor User Interface (UI) obscures to remind this.

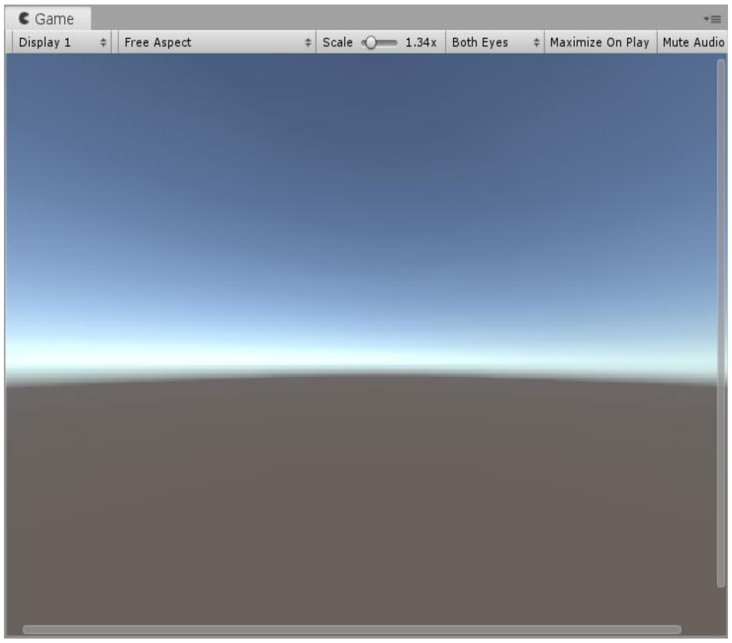


Figure 3.8: Game Window

3.4.1.5. Project Window

The left panel shows the folder structure of the task as a various leveled list. At the point when the organizer is chosen its substance are appeared in the board to the right. The independent resources are appeared in the right panel as symbols that shows their kind. The left of the slider shows the right now chosen thing, including a full way to the thing if a hunt is being performed.

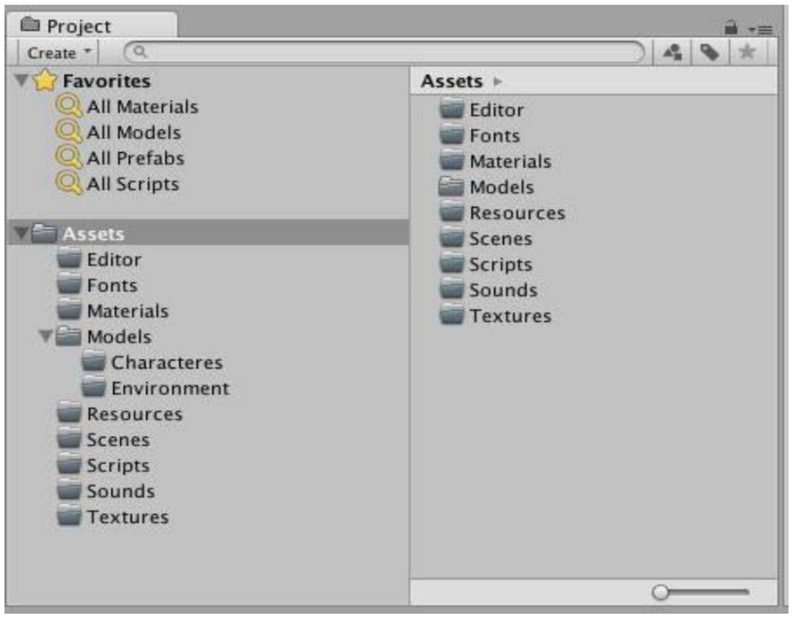


Figure 3.9: Project Window

3.4.2. Samsung Vuze Phone Application

Since Vuze 360 camera is being used to shoot videos therefore to operate the Vuze camera the company app is downloaded and installed on phone. It enables one to record videos, preview the content and have large amount of control over the images, from exposure control to the ability to switch between lenses and change the field of view. The graphical overlay of the Vuze tells whether the camera is level or not. If it is not at a level the app highlights the section of the camera that needs to be adjusted to make it level.

3.4.3. Vuze VR studio

As the Vuze camera do not stitch the footage in camera so to render what is shot, VR software is used. It includes features like 2D videos at 60 fps, native photos, and horizon stabilization and zoom tools. The software consists of three panels: import, preview and edit, and render. In ‘import’ footage is downloaded from the Vuze camera. In ‘preview and edit’ footage can be watched either in the stereo or from left or right eye perspective. ‘Rendering’ allows one to choose between 4K or 2K resolution and 16:9 or 2:1 aspect ratio. Rendering can be done in 3D or 2D.

3.5. Programming Language

In Unity one can utilize scripts to grow basically all aspects of the real-time intelligent and interactive content. Unity underpins scripting in C#. Consequently, C# language is utilized all through the project to assemble codes. It is an industry level language and has a few similitudes to C++ and Java. It is an overseen language as it naturally does the management of memory: assigning unassigning memory, covering memory spills and so forth. Since scripting advises the Game Objects how to carry on accordingly an item situated language is required for the advancement of the codes. Unity runs in a major circle, it peruses the entirety of the information in the game scene, for instance perusing the lights, the lattices, what the practices are and forms the entirety of the data. Subsequently a statically composed language is required so the code is checked before it is transformed into an application. This makes it simpler to discover blunders which is especially valuable for the designers (Wang, 2020).

CHAPTER 4

4. PROJECT ANALYSIS AND EVALUATION

4.1. Introduction

After the software and hardware acquisition, the task of integration and evaluation was undertaken. The Oculus Rift was integrated with Unity. An immersive 3D 360-degree environment was originated by using panoramic skybox material. Videos were captured and the stitching of the 3D videos was performed using Vuze VR studio. Moreover, the video was integrated with the unity software and finally an application for windows was developed.

The project had been completed in five stages which are shown below in figure 4.1.

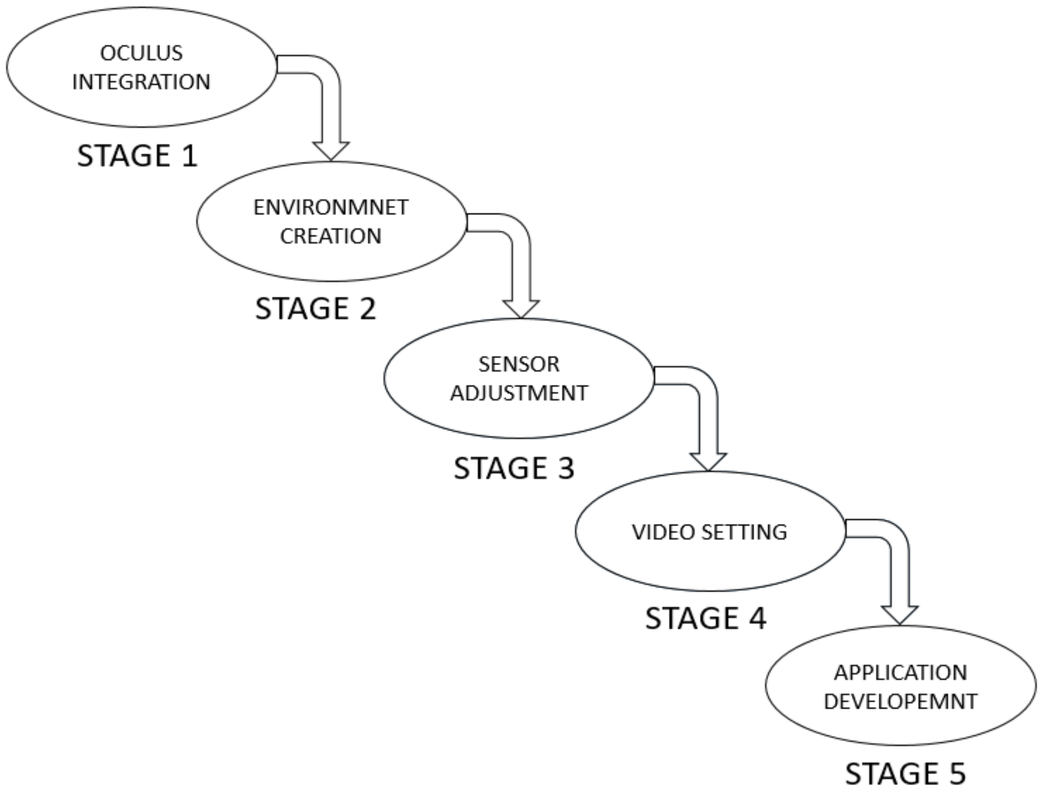


Figure 4.1: Stages of the Project

4.2. Oculus Integration

Since Rift is a very popular headset so it was used as an HMD in the project. So, the first step needed was to familiarize the development software Unity with this headset. For this purpose, Oculus SDK was used. It is a Unity plugin that enables us to make apps for Oculus with the right framework. The following set of operations was performed to integrate Oculus with unity.

4.2.1. Importing Oculus Integration plugin

Figure 4.2 shows the Oculus integration plugin which was imported to Unity using the Unity Asset store.

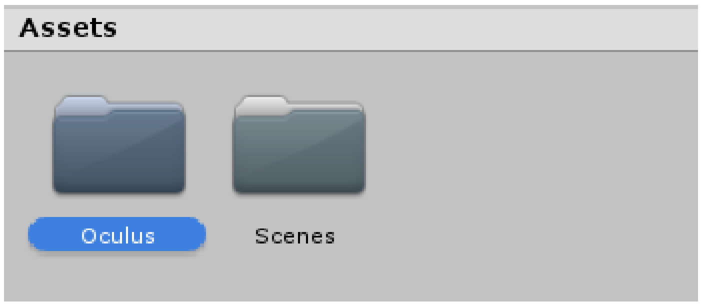
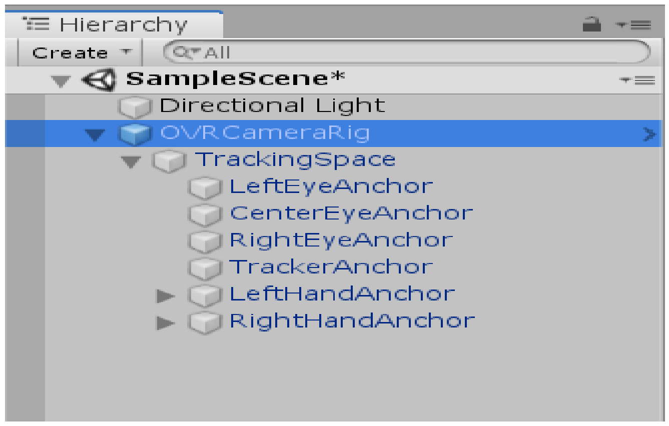


Figure 4.2: Oculus Integration

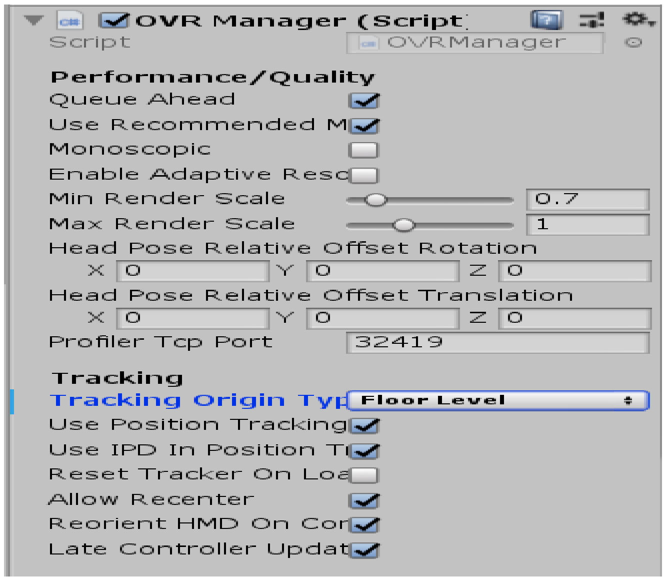
After that the main camera of unity was deleted from the scene and rather the OVRCameraRig for Oculus was dragged into the scene.



SampleScene*
Directional Light
OVRCameraRig
TrackingSpace
LeftEyeAnchor
CenterEyeAnchor
RightEyeAnchor
TrackerAnchor
LeftHandAnchor
RightHandAnchor

Figure 4.3: OVRCameraRig

The properties of the OVRCameraRig were adjusted as shown in the figure 4.4.



OVR Manager (Script)
Script OVRManager

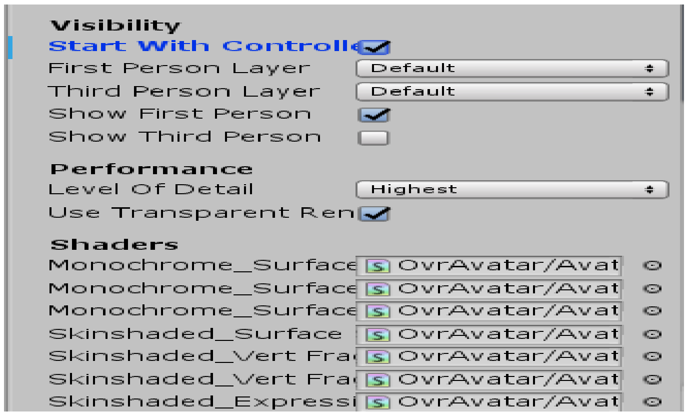
Performance/Quality
Queue Ahead ✓
Use Recommended M ✓
Monoscopic
Enable Adaptive Reso
Min Render Scale 0.7
Max Render Scale 1
Head Pose Relative Offset Rotation
X 0 Y 0 Z 0
Head Pose Relative Offset Translation
X 0 Y 0 Z 0
Profiler Tcp Port 32419

Tracking
Tracking Origin Typ Floor Level
Use Position Tracking ✓
Use IPD In Position Tr ✓
Reset Tracker On Loa
Allow Recenter ✓
Reorient HMD On Cor ✓
Late Controller Updat ✓

Figure 4.4: OVRCameraRig Properties

4.2.2. *Oculus Controllers setup*

Cameras in Unity represent the player's eyes. After the cameras, controllers were set which represented the player's hands. To allow the user to use the Oculus Avatar hands, the Local Avatar prefab was dragged in the scene and its properties were adjusted as shown in figure 4.5.



Visibility
Start With Controlle ✓
First Person Layer Default
Third Person Layer Default
Show First Person ✓
Show Third Person

Performance
Level Of Detail Highest
Use Transparent Ren ✓

Shaders
Monochrome_Surface OvrAvatar/Avat
Monochrome_Surface OvrAvatar/Avat
Monochrome_Surface OvrAvatar/Avat
Skinshaded_Surface OvrAvatar/Avat
Skinshaded_Vert Fra OvrAvatar/Avat
Skinshaded_Vert Fra OvrAvatar/Avat
Skinshaded_Expressi OvrAvatar/Avat

Figure 4.5: Local Avatar Properties

The complete coding of Oculus Integration is attached in Appendix B provided at the end of the document.

4.3. Environment Creation

The Skybox Panoramic Shader had been used in this project since it exhibits both the 180-degree and 360-degree videos in either equirectangular or cube map layout as a scene backdrop. The video in any of these formats can be played by the Video Player component and output to a render texture.

To use this shader a new render texture was created. The resolution of the render texture was adjusted in accordance with the resolution of the video. After that to create a video player in the scene the video that had been captured was dragged into the scene. A player was automatically created with the video allotted to it. The render mode of the video player was changed to render texture and the render texture from the project view was dragged into the target texture field (Subash, 2019).

Next the skybox material was assigned to the render texture in equirectangular i.e. the latitude-longitude format. The complete coding for the skybox is attached in Appendix C provided at the end of this document.

4.4. Sensor Adjustment

To create a natural experience, the movement of the HMD and the controllers must be traced in real time otherwise the environment is perceived as artificial. Therefore, the position of the sensors was adjusted as to give maximum sense of immersion. For this purpose, the Oculus application was used. The standing height was set so that the Rift calculates the distance from the ground. It only needs to be set once after which anyone can use the Rift. The standing height adjustment makes the experience in VR more realistic. It was made sure that nothing blocks the line of sight between HMD and the sensors. Also, the glossy side of the lenses was made to point towards the game window.

4.5. Video Capturing, Stitching and Integration with Unity

To capture the video Samsung Vuze 360 had been used. The videos were recorded in mp4 format. Since there are eight lenses in the Vuze camera so eight different recordings were acquired but the camera does not stitch the videos. Since the purpose of stitching the videos is to create an ideal panoramic video from a set of overlapping videos therefore, Vuze VR studio was used to stitch the videos and the center of the video was adjusted accordingly. After stitching the video, it was rendered using the same Vuze VR studio application. Later the video was imported in Unity to be used in the VR application built for Unity.

4.6. Application Development

An interactive Windows stand-alone application was developed using Unity to be used with Oculus Rift headset. For that purpose, the GUI screen was designed in Adobe Photoshop (Ahola, 2019). The interface of the application was kept simple and understandable.

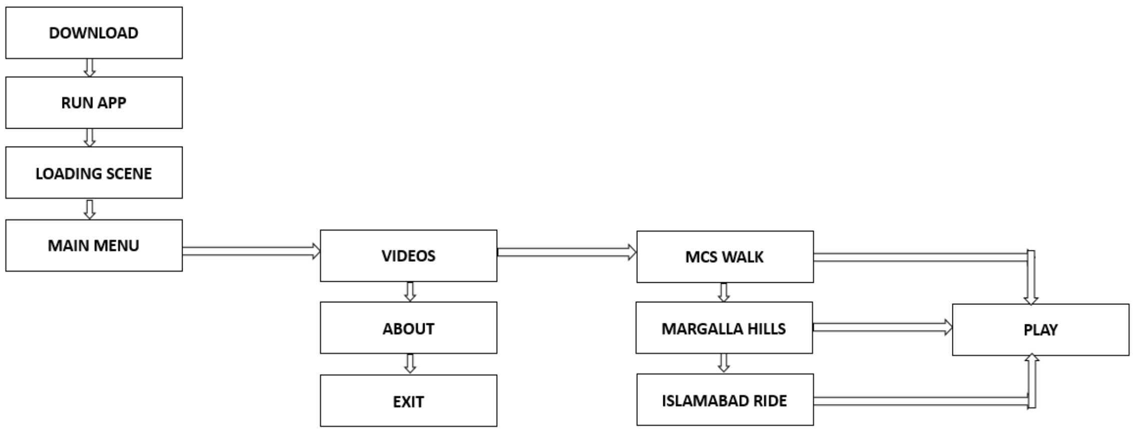


Figure 4.6: Layout of Application

Figure 4.6 present above shows the main menu is the first screen that appears in front of the user after executing the application. The Main menu has three options. ‘VIDEOS’ button will take the user to a next scene which displays all the videos that the user can play in Oculus Rift. The ‘ABOUT’ button takes the user to a text document that explains the working and other details of the application. While pressing the ‘EXIT’ button the user gives the consent of leaving the application. Figure 4.7 below represents the GUI of the proposed application.

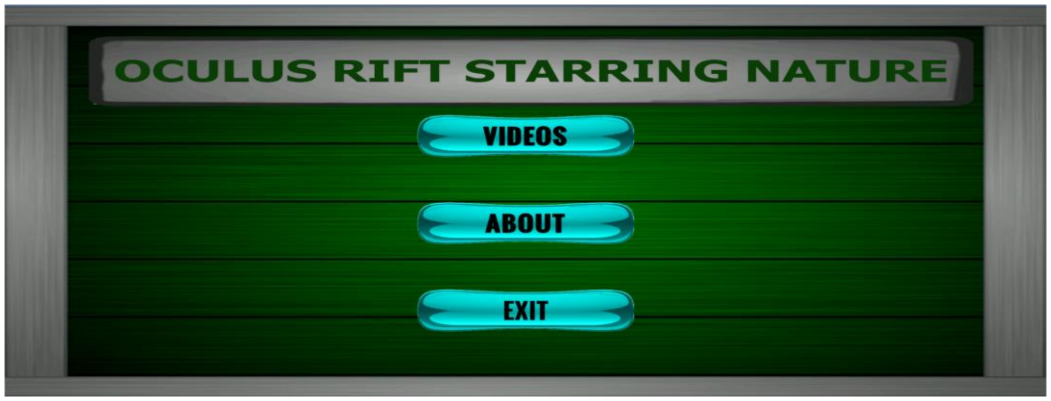


Figure 4.7: Graphical User Interface

The complete coding for the Graphical User Interface (GUI) of the application is attached in Appendix D provided at the end of the document.

CHAPTER 5

5. FUTURE WORK

5.1. Future Work

VR technology is becoming more accessible with each passing day. Its use is not only restricted to games and entertainment but it has proved to be an amazing tool that can be used successfully in tourism, marketing, recruitment and training, creating ideas and forecasting trends, pain management, digital marketing, architectural designs, social science, occupational safety and clinical therapies.

The scope of the proposed project was limited to the exploration and creation of a 3D 360-degree environment for the user and the development of Windows application that can be used with HMDs. However, by venturing deeper into the project and exploring the depth of the process we can achieve some additional objectives which are discussed below.

5.2. Automatic Speed Control

Wearing and not wearing HMDs to adjust the speed of the video with the user's own motion can be frustrating for the user so by using motion sensors such as pedometer and accelerometer, speed of the video can be automatically matched with the speed of the user in motion. The foot count from the motion sensor can be fed as input to Unity where it calculates the speed in meters per second and plays the video at that speed.

5.3. Application Development

The project develops an application for Windows. Since different people are using different operating systems so VR applications can also be developed for Android, iOS, Linux, and other operating systems. Moreover, applications can be developed that reduces the memory utilization by making videos available at run-time. In this way the Central Processing Unit (CPU) and GPU resources can be saved. This can also enable people to download the VR application on their mobile devices as well (Aseeri, 2013).

5.4. Rehabilitation

In the proposed project the VR application is developed for healthy users to engage their interest in keeping themselves fit but it is not restricted to only healthy beings, it can be a very effective tool for the rehabilitation process. It can provide opportunity to assimilate stimulation and enjoyment into the rehabilitation process. A virtual environment can be created for the patients using an HMD to help them overcome the discomfort and suffering. Patients exercising their lower limbs can find a treadmill walk equivalent to a walk in a park or countryside with the help of VR. It can be used to treat stroke patients, people suffering from back pain and having any sort of disorders. VR can show a spike in patient’s motivation thus improving the patient’s coordination skills when doing physical therapy (Larsen, 2018).

5.5. Marketing

Applications like this can be explicitly used for marketing purposes, especially in tourism. Hotels can have interactive 3D tours of their rooms, lobby, and restrooms to attract tourists. Hotel owners can let everyone visit their hotels through VR before booking the room. This can help them gain the confidence of more clients.

5.6. Military

It can be used for the training of soldiers. Simulated environments depicting the enemy’s area can be created which can give an insight to the soldiers about the opponent’s land. Moreover, they can practice working together in realistically replicated environments before they enter the opponent’s area. This mesmeric environment can be significantly important because the training that captures the consciousness of the learner is often retained longer and better understood.

5.7. Conclusion

Virtual Reality has come a long way. It is a comprehensive medium of interaction that discloses information with an extra spectacular dimension. This thesis has made a minute effort in bringing to light another possibility of taking advantage of this technology in field of entertainment. In the future, new probable platforms of computation, virtual reality, augmented reality, and mixed reality can play a remarkable role in fields of entertainment, scientific research and business. With the implementation of this thesis, it has been illustrated that a virtual reality application for a treadmill user is not only plausible and meaningful but also feasible.

Of course, this prototype has still a long way to go off on a tangent, there are still number of refinements to be made. For example, the application should not only be available for Windows but for other operating systems as well. It is not difficult to device this feature anymore. Game engines like Unity have integrated high level APIs. Moreover, third parties developed SDKs and APIs can also be integrated with Unity for application development.

Another important dimension is the performance. Due to the limitation in the efficiency of application, the frame rate is significantly dropped at run-time. So, optimization needs to be done for smooth VR experience. Future improvements should be taken on rationalizing and optimizing the application. For example, when all the scripts are executed at run-time, it consumes huge amount of central processing unit and GPU resources and ultimately results in low frame rates and motion sickness. Thus, there is a need to boost the performance of the VR application.

To conclude, this thesis has provided an outline of Virtual Reality and has explained the composition of VR and how all elements can be integrated to create an application that can help fitness enthusiasts to boost up their exercises. Technologies such as VR and AR holds unlimited possibilities in scientific research and development. In the upcoming year virtual reality seems to reach new heights and as per prediction, in 2022, 121 million virtual reality and mixed reality devices and applications will be sold.

6. APPENDIX

6.1. Appendix A: Synopsis

OCULUS RIFT STARRING THE NATURE

Extended Title: Creation of Oculus Rift featured treadmill incorporating Pakistan's vista.

Brief Description of The Project / Thesis with Salient Specifications: This work aims at creation of an immersive video where viewers can pan around a full 360 degrees by moving their heads inside oculus rift. The technical problem to be solved is concerned with making videos for oculus rift that can engage as well as immerse viewers in virtual reality and creation of complete VR environment via software.

Scope of Work: The oculus rift featured treadmill finds its application in gyms where many people show interest in 3D walking and home to facilitate the monotonous walk hour.

Academic Objectives:

- Attaining software programming skills
- Proficiency in 3D design skills
- Getting insight of Unity3D, the current development choice among game engines

Application / End Goal Objectives: To create a complete 360-degree 3D experience for treadmill users by using oculus rift.

Previous Work Done on The Subject:

- Digital Signal Processing
- Digital Image Processing
- Object Oriented Programming
- Data Structures and Algorithms
- 3D project using AutoCAD software

Material Resources Required:

- 3D camera
- Core i7 laptop with graphic card
- Oculus Rift
- Treadmill
- Unity 3D software
- SDK for oculus rift
- C#

No of Students : 4 **Group Members:** Hussain Murad Mahnoor Ehtisham Noman Ali Bajwa Zainab Batool
Special Skills Required: • C# • Unity 3D software • Image Processing • App development

<u>Approval Status</u>

Supervisor Name & Signature: LT COL Dr. HASNAT KHURSHID

____

Assigned to: Hussain Murad
Mahnoor Ehtisham
Noman Ali Bajwa
Zainab Batool

HoD Signature ____________________

<u>R&D SC Record Status</u>

File # _______________ Coordinator Signature __________________

6.2. Appendix B: Oculus Integration

6.2.1. OVR Headset Configuration

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OVRHeadsetEmulator : MonoBehaviour {
        public enum OpMode
        {
                Off,

             EditorOnly,
             AlwaysOn
      }

      public OpMode opMode = OpMode.EditorOnly;
      public bool resetHmdPoseOnRelease = true;
      public bool resetHmdPoseByMiddleMouseButton = true;

      public KeyCode[] activateKeys = new KeyCode[] { KeyCode.LeftControl,
KeyCode.RightControl };

      public KeyCode[] pitchKeys = new KeyCode[] { KeyCode.LeftAlt, KeyCode.RightAlt };

      OVRManager manager;

      const float MOUSE_SCALE_X = -2.0f;
      const float MOUSE_SCALE_X_PITCH = -2.0f;
      const float MOUSE_SCALE_Y = 2.0f;
      const float MOUSE_SCALE_HEIGHT = 1.0f;
      const float MAX_ROLL = 85.0f;
```

```
        private bool lastFrameEmulationActivated = false;

        private Vector3 recordedHeadPoseRelativeOffsetTranslation;
        private Vector3 recordedHeadPoseRelativeOffsetRotation;

        private bool hasSentEvent = false;
        private bool emulatorHasInitialized = false;

        // Use this for initialization
        void Start () {



        }

        // Update is called once per frame
        void Update () {
                if (!emulatorHasInitialized)
                {
                        if (OVRManager.OVRManagerinitialized)
                        {
                                Cursor.lockState = CursorLockMode.None;
                                manager = OVRManager.instance;
                                recordedHeadPoseRelativeOffsetTranslation =
manager.headPoseRelativeOffsetTranslation;
                                recordedHeadPoseRelativeOffsetRotation =
manager.headPoseRelativeOffsetRotation;
                                emulatorHasInitialized = true;
                        }
                        else
                                return;
                }
                bool emulationActivated = IsEmulationActivated();
                if (emulationActivated)
                {
                        Cursor.lockState = CursorLockMode.Locked;
```

```
                    if (!lastFrameEmulationActivated && resetHmdPoseOnRelease)
                    {
                        manager.headPoseRelativeOffsetTranslation =
recordedHeadPoseRelativeOffsetTranslation;
                        manager.headPoseRelativeOffsetRotation =
recordedHeadPoseRelativeOffsetRotation;
                    }

                    if (resetHmdPoseByMiddleMouseButton && Input.GetMouseButton(2))
                    {
                        manager.headPoseRelativeOffsetTranslation = Vector3.zero;
                        manager.headPoseRelativeOffsetRotation = Vector3.zero;
                    }
                    else
                    {
                        Vector3 emulatedTranslation =
manager.headPoseRelativeOffsetTranslation;
                        float deltaMouseScrollWheel = Input.GetAxis("Mouse
ScrollWheel");
                        float emulatedHeight = deltaMouseScrollWheel *
MOUSE_SCALE_HEIGHT;
                        emulatedTranslation.y += emulatedHeight;
                        manager.headPoseRelativeOffsetTranslation =
emulatedTranslation;

                        float deltaX = Input.GetAxis("Mouse X");
                        float deltaY = Input.GetAxis("Mouse Y");

                        Vector3 emulatedAngles =
manager.headPoseRelativeOffsetRotation;
                        float emulatedRoll = emulatedAngles.x;
                        float emulatedYaw = emulatedAngles.y;
```

6.2.2. OVR Camera Rig

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

#if UNITY_2017_2_OR_NEWER
using InputTracking = UnityEngine.XR.InputTracking;
using Node = UnityEngine.XR.XRNode;
#else
using InputTracking = UnityEngine.VR.InputTracking;
using Node = UnityEngine.VR.VRNode;
#endif

/// <summary>
/// A head-tracked stereoscopic virtual reality camera rig.
/// </summary>
[ExecuteInEditMode]
public class OVRCameraRig : MonoBehaviour
{
	/// <summary>
	/// The left eye camera.
	/// </summary>
	public Camera leftEyeCamera { get { return (usePerEyeCameras) ?
_leftEyeCamera : _centerEyeCamera; } }
	/// <summary>
	/// The right eye camera.
	/// </summary>
	public Camera rightEyeCamera { get { return (usePerEyeCameras) ?
_rightEyeCamera : _centerEyeCamera; } }

	/// <summary>
	/// Provides a root transform for all anchors in tracking space.
	/// </summary>
	public Transform trackingSpace { get; private set; }
	/// <summary>
	/// Always coincides with the pose of the left eye.
	/// </summary>
	public Transform leftEyeAnchor { get; private set; }
	/// <summary>
	/// Always coincides with average of the left and right eye poses.
	/// </summary>
	public Transform centerEyeAnchor { get; private set; }
```

```
/// <summary>
/// Always coincides with the pose of the right eye.
/// </summary>
public Transform rightEyeAnchor { get; private set; }
/// <summary>
/// Always coincides with the pose of the left hand.
/// </summary>
public Transform leftHandAnchor { get; private set; }
/// <summary>
/// Always coincides with the pose of the right hand.
/// </summary>
public Transform rightHandAnchor { get; private set; }
/// <summary>
/// Anchors controller pose to fix offset issues for the left hand.
/// </summary>
public Transform leftControllerAnchor { get; private set; }
/// <summary>
/// Anchors controller pose to fix offset issues for the right hand.
/// </summary>
public Transform rightControllerAnchor { get; private set; }
/// <summary>
/// Always coincides with the pose of the sensor.
/// </summary>
public Transform trackerAnchor { get; private set; }
/// <summary>
/// Occurs when the eye pose anchors have been set.
/// </summary>
public event System.Action<OVRCameraRig> UpdatedAnchors;
/// <summary>
/// If true, separate cameras will be used for the left and right eyes.
/// </summary>
public bool usePerEyeCameras = false;
/// <summary>

        /// If true, all tracked anchors are updated in FixedUpdate instead of Update to
favor physics fidelity.
        /// \note: If the fixed update rate doesn't match the rendering framerate
(OVRManager.display.appFramerate), the anchors will visibly judder.
        /// </summary>
        public bool useFixedUpdateForTracking = false;
        /// <summary>
        /// If true, the cameras on the eyeAnchors will be disabled.
        /// \note: The main camera of the game will be used to provide VR rendering. And
the tracking space anchors will still be updated to provide reference poses.
```

```
	/// </summary>
	public bool disableEyeAnchorCameras = false;


	protected bool _skipUpdate = false;
	protected readonly string trackingSpaceName = "TrackingSpace";
	protected readonly string trackerAnchorName = "TrackerAnchor";
	protected readonly string leftEyeAnchorName = "LeftEyeAnchor";
	protected readonly string centerEyeAnchorName = "CenterEyeAnchor";
	protected readonly string rightEyeAnchorName = "RightEyeAnchor";
	protected readonly string leftHandAnchorName = "LeftHandAnchor";
	protected readonly string rightHandAnchorName = "RightHandAnchor";
	protected readonly string leftControllerAnchorName = "LeftControllerAnchor";
	protected readonly string rightControllerAnchorName =
"RightControllerAnchor";
	protected Camera _centerEyeCamera;
	protected Camera _leftEyeCamera;
	protected Camera _rightEyeCamera;

#region Unity Messages
	protected virtual void Awake()
	{
		_skipUpdate = true;
		EnsureGameObjectIntegrity();
	}

	protected virtual void Start()
	{
		UpdateAnchors(true, true);
		Application.onBeforeRender += OnBeforeRenderCallback;
	}
	protected virtual void FixedUpdate()
	{
		if (useFixedUpdateForTracking)
			UpdateAnchors(true, true);
	}

	protected virtual void Update()
	{
		_skipUpdate = false;

		if (!useFixedUpdateForTracking)
			UpdateAnchors(true, true);
	}
```

```
        protected virtual void OnDestroy()
        {
                Application.onBeforeRender -= OnBeforeRenderCallback;
        }
#endregion

        protected virtual void UpdateAnchors(bool updateEyeAnchors, bool
updateHandAnchors)
        {
                if (!OVRManager.OVRManagerinitialized)
                        return;

                EnsureGameObjectIntegrity();

                if (!Application.isPlaying)
                        return;

                if (_skipUpdate)
                {
                        centerEyeAnchor.FromOVRPose(OVRPose.identity, true);
                        leftEyeAnchor.FromOVRPose(OVRPose.identity, true);
                        rightEyeAnchor.FromOVRPose(OVRPose.identity, true);

                        return;
                }

                bool monoscopic = OVRManager.instance.monoscopic;
                bool hmdPresent = OVRNodeStateProperties.IsHmdPresent();

                OVRPose tracker = OVRManager.tracker.GetPose();



                trackerAnchor.localRotation = tracker.orientation;

                Quaternion emulatedRotation = Quaternion.Euler(-
OVRManager.instance.headPoseRelativeOffsetRotation.x, -
OVRManager.instance.headPoseRelativeOffsetRotation.y,
OVRManager.instance.headPoseRelativeOffsetRotation.z);

                //Note: in the below code, when using UnityEngine's API, we only update
anchor transforms if we have a new, fresh value this frame.
                //If we don't, it could mean that tracking is lost, etc. so the pose should not
change in the virtual world.
```

```
			//This can be thought of as similar to calling InputTracking
GetLocalPosition and Rotation, but only for doing so when the pose is valid.
			//If false is returned for any of these calls, then a new pose is not valid and
thus should not be updated.
			if (updateEyeAnchors)
			{
				if (hmdPresent)
				{
					Vector3 centerEyePosition = Vector3.zero;
					Quaternion centerEyeRotation = Quaternion.identity;

					if
(OVRNodeStateProperties.GetNodeStatePropertyVector3(Node.CenterEye,
NodeStatePropertyType.Position, OVRPlugin.Node.EyeCenter, OVRPlugin.Step.Render,
out centerEyePosition))
						centerEyeAnchor.localPosition =
centerEyePosition;
					if
(OVRNodeStateProperties.GetNodeStatePropertyQuaternion(Node.CenterEye,
NodeStatePropertyType.Orientation, OVRPlugin.Node.EyeCenter,
OVRPlugin.Step.Render, out centerEyeRotation))
						centerEyeAnchor.localRotation =
centerEyeRotation;
				}


				else
				{
					centerEyeAnchor.localRotation = emulatedRotation;
					centerEyeAnchor.localPosition =
OVRManager.instance.headPoseRelativeOffsetTranslation;
				}

				if (!hmdPresent || monoscopic)
				{
					leftEyeAnchor.localPosition =
centerEyeAnchor.localPosition;
					rightEyeAnchor.localPosition =
centerEyeAnchor.localPosition;
					leftEyeAnchor.localRotation =
centerEyeAnchor.localRotation;
					rightEyeAnchor.localRotation =
centerEyeAnchor.localRotation;
				}
				else
				{
					Vector3 leftEyePosition = Vector3.zero;
```

```
                                        Vector3 rightEyePosition = Vector3.zero;
                                        Quaternion leftEyeRotation = Quaternion.identity;
                                        Quaternion rightEyeRotation = Quaternion.identity;

                                        if
(OVRNodeStateProperties.GetNodeStatePropertyVector3(Node.LeftEye,
NodeStatePropertyType.Position, OVRPlugin.Node.EyeLeft, OVRPlugin.Step.Render,
out leftEyePosition))
                                                leftEyeAnchor.localPosition = leftEyePosition;
                                        if
(OVRNodeStateProperties.GetNodeStatePropertyVector3(Node.RightEye,
NodeStatePropertyType.Position, OVRPlugin.Node.EyeRight, OVRPlugin.Step.Render,
out rightEyePosition))
                                                rightEyeAnchor.localPosition = rightEyePosition;
                                        if
(OVRNodeStateProperties.GetNodeStatePropertyQuaternion(Node.LeftEye,
NodeStatePropertyType.Orientation, OVRPlugin.Node.EyeLeft,
OVRPlugin.Step.Render, out leftEyeRotation))
                                                leftEyeAnchor.localRotation = leftEyeRotation;
                                        if
(OVRNodeStateProperties.GetNodeStatePropertyQuaternion(Node.RightEye,
NodeStatePropertyType.Orientation, OVRPlugin.Node.EyeRight,
OVRPlugin.Step.Render, out rightEyeRotation))
                                                rightEyeAnchor.localRotation = rightEyeRotation;
                                }
                        }

                        if (updateHandAnchors)
                        {
                                //Need this for controller offset because if we're on OpenVR, we
want to set the local poses as specified by Unity, but if we're not, OVRInput local
position is the right anchor
                                if (OVRManager.loadedXRDevice ==
OVRManager.XRDevice.OpenVR)
                                {
                                        Vector3 leftPos = Vector3.zero;
                                        Vector3 rightPos = Vector3.zero;
                                        Quaternion leftQuat = Quaternion.identity;
                                        Quaternion rightQuat = Quaternion.identity;

                                        if
(OVRNodeStateProperties.GetNodeStatePropertyVector3(Node.LeftHand,
NodeStatePropertyType.Position, OVRPlugin.Node.HandLeft, OVRPlugin.Step.Render,
out leftPos))
```

```
leftHandAnchor.localPosition = leftPos;
                                        if
(OVRNodeStateProperties.GetNodeStatePropertyVector3(Node.RightHand,
NodeStatePropertyType.Position, OVRPlugin.Node.HandRight,
OVRPlugin.Step.Render, out rightPos))
                                                  rightHandAnchor.localPosition = rightPos;
                                        if
(OVRNodeStateProperties.GetNodeStatePropertyQuaternion(Node.LeftHand,
NodeStatePropertyType.Orientation, OVRPlugin.Node.HandLeft,
OVRPlugin.Step.Render, out leftQuat))
                                                  leftHandAnchor.localRotation = leftQuat;
                                        if
(OVRNodeStateProperties.GetNodeStatePropertyQuaternion(Node.RightHand,
NodeStatePropertyType.Orientation, OVRPlugin.Node.HandRight,
OVRPlugin.Step.Render, out rightQuat))
                                                  rightHandAnchor.localRotation = rightQuat;

                              }
                              else
                              {
                                        leftHandAnchor.localPosition =
OVRInput.GetLocalControllerPosition(OVRInput.Controller.LTouch);
                                        rightHandAnchor.localPosition =
OVRInput.GetLocalControllerPosition(OVRInput.Controller.RTouch);
                                        leftHandAnchor.localRotation =
OVRInput.GetLocalControllerRotation(OVRInput.Controller.LTouch);
                                        rightHandAnchor.localRotation =
OVRInput.GetLocalControllerRotation(OVRInput.Controller.RTouch);
                              }

                              trackerAnchor.localPosition = tracker.position;

                              OVRPose leftOffsetPose = OVRPose.identity;
                              OVRPose rightOffsetPose = OVRPose.identity;
                             if (OVRManager.loadedXRDevice ==
OVRManager.XRDevice.OpenVR)
                             {
                                       leftOffsetPose =
OVRManager.GetOpenVRControllerOffset(Node.LeftHand);
                                       rightOffsetPose =
OVRManager.GetOpenVRControllerOffset(Node.RightHand);

                                       //Sets poses of left and right nodes, local to the tracking
space.
```

```
	OVRManager.SetOpenVRLocalPose(trackingSpace.InverseTransformPoint(leftC
ontrollerAnchor.position),

	trackingSpace.InverseTransformPoint(rightControllerAnchor.position),
					Quaternion.Inverse(trackingSpace.rotation) *
leftControllerAnchor.rotation,
					Quaternion.Inverse(trackingSpace.rotation) *
rightControllerAnchor.rotation);
				}
				rightControllerAnchor.localPosition = rightOffsetPose.position;
				rightControllerAnchor.localRotation = rightOffsetPose.orientation;
				leftControllerAnchor.localPosition = leftOffsetPose.position;
				leftControllerAnchor.localRotation = leftOffsetPose.orientation;
			}

			RaiseUpdatedAnchorsEvent();
		}

		protected virtual void OnBeforeRenderCallback()
		{
			if (OVRManager.loadedXRDevice == OVRManager.XRDevice.Oculus)
			//Restrict late-update to only Oculus devices
			{
				bool controllersNeedUpdate =
OVRManager.instance.LateControllerUpdate;
#if USING_XR_SDK
				//For the XR SDK, we need to late update head pose, not just the
controllers, because the functionality
				//is no longer built-in to the Engine. Under legacy, late camera
update is done by default. In the XR SDK, you must use
				//Tracked Pose Driver to get this by default, which we do not use.
So, we have to manually late update camera poses.
				UpdateAnchors(true, controllersNeedUpdate);
#else
				if (controllersNeedUpdate)
				if (controllersNeedUpdate)
					UpdateAnchors(false, true);
#endif
			}
		}

		protected virtual void RaiseUpdatedAnchorsEvent()
		{
			if (UpdatedAnchors != null)
			{
				UpdatedAnchors(this);
		}
	}

	public virtual void EnsureGameObjectIntegrity()
	{
		bool monoscopic = OVRManager.instance != null ?
OVRManager.instance.monoscopic : false;

		if (trackingSpace == null)
			trackingSpace = ConfigureAnchor(null, trackingSpaceName);

		if (leftEyeAnchor == null)
			leftEyeAnchor = ConfigureAnchor(trackingSpace,
leftEyeAnchorName);

		if (centerEyeAnchor == null)
			centerEyeAnchor = ConfigureAnchor(trackingSpace,
centerEyeAnchorName);

		if (rightEyeAnchor == null)
			rightEyeAnchor = ConfigureAnchor(trackingSpace,
rightEyeAnchorName);

		if (leftHandAnchor == null)
			leftHandAnchor = ConfigureAnchor(trackingSpace,
leftHandAnchorName);

		if (rightHandAnchor == null)
			rightHandAnchor = ConfigureAnchor(trackingSpace,
rightHandAnchorName);

		if (trackerAnchor == null)


			trackerAnchor = ConfigureAnchor(trackingSpace,
trackerAnchorName);

		if (leftControllerAnchor == null)
			leftControllerAnchor = ConfigureAnchor(leftHandAnchor,
leftControllerAnchorName);

		if (rightControllerAnchor == null)
			rightControllerAnchor = ConfigureAnchor(rightHandAnchor,
rightControllerAnchorName);
```

```
                    if (_centerEyeCamera == null || _leftEyeCamera == null ||
        _rightEyeCamera == null)
                    {
                            _centerEyeCamera =
        centerEyeAnchor.GetComponent<Camera>();
                            _leftEyeCamera = leftEyeAnchor.GetComponent<Camera>();
                            _rightEyeCamera = rightEyeAnchor.GetComponent<Camera>();

                            if (_centerEyeCamera == null)
                            {
                                    _centerEyeCamera =
        centerEyeAnchor.gameObject.AddComponent<Camera>();
                                    _centerEyeCamera.tag = "MainCamera";
                            }

                            if (_leftEyeCamera == null)
                            {
                                    _leftEyeCamera =
        leftEyeAnchor.gameObject.AddComponent<Camera>();
                                    _leftEyeCamera.tag = "MainCamera";
                            }

                            if (_rightEyeCamera == null)
                            {
                                    _rightEyeCamera =
        rightEyeAnchor.gameObject.AddComponent<Camera>();
                                    _rightEyeCamera.tag = "MainCamera";
                            }

                            _centerEyeCamera.stereoTargetEye = StereoTargetEyeMask.Both;
                            _leftEyeCamera.stereoTargetEye = StereoTargetEyeMask.Left;
                            _rightEyeCamera.stereoTargetEye = StereoTargetEyeMask.Right;
                    }


                    if (monoscopic && !OVRPlugin.EyeTextureArrayEnabled)
                    {
                            // Output to left eye only when in monoscopic mode
                            if (_centerEyeCamera.stereoTargetEye !=
        StereoTargetEyeMask.Left)
                            {
                                    _centerEyeCamera.stereoTargetEye =
        StereoTargetEyeMask.Left;
                            }
                    }
                    else
```

```
                {
                        if (_centerEyeCamera.stereoTargetEye !=
StereoTargetEyeMask.Both)
                        {
                                _centerEyeCamera.stereoTargetEye =
StereoTargetEyeMask.Both;
                        }
                }

                if (disableEyeAnchorCameras)
                {
                        _centerEyeCamera.enabled = false;
                        _leftEyeCamera.enabled = false;
                        _rightEyeCamera.enabled = false;
                }
                else
                {
                        // disable the right eye camera when in monoscopic mode
                        if (_centerEyeCamera.enabled == usePerEyeCameras ||
                                        _leftEyeCamera.enabled == !usePerEyeCameras ||
                                        _rightEyeCamera.enabled == !(usePerEyeCameras
&& (!monoscopic || OVRPlugin.EyeTextureArrayEnabled)))
                        {
                                _skipUpdate = true;
                        }

                        _centerEyeCamera.enabled = !usePerEyeCameras;
                        _leftEyeCamera.enabled = usePerEyeCameras;
                        _rightEyeCamera.enabled = (usePerEyeCameras &&
(!monoscopic || OVRPlugin.EyeTextureArrayEnabled));

                }
        }

        protected virtual Transform ConfigureAnchor(Transform root, string name)

        {
                Transform anchor = (root != null) ? root.Find(name) : null;

                if (anchor == null)
                {
                        anchor = transform.Find(name);
                }

                if (anchor == null)
                {
                        anchor = new GameObject(name).transform;
                }

                anchor.name = name;
                anchor.parent = (root != null) ? root : transform;
                anchor.localScale = Vector3.one;
                anchor.localPosition = Vector3.zero;
                anchor.localRotation = Quaternion.identity;

                return anchor;
        }

        public virtual Matrix4x4 ComputeTrackReferenceMatrix()
        {
                if (centerEyeAnchor == null)
                {
                        Debug.LogError("centerEyeAnchor is required");
                        return Matrix4x4.identity;
                }

                // The ideal approach would be using
UnityEngine.VR.VRNode.TrackingReference, then we would not have to depend on the
OVRCameraRig. Unfortunately, it is not available in Unity 5.4.3

                OVRPose headPose = OVRPose.identity;

                Vector3 pos;


               Quaternion rot;
               if (OVRNodeStateProperties.GetNodeStatePropertyVector3(Node.Head,
NodeStatePropertyType.Position, OVRPlugin.Node.Head, OVRPlugin.Step.Render, out
pos))
                       headPose.position = pos;
               if
(OVRNodeStateProperties.GetNodeStatePropertyQuaternion(Node.Head,
NodeStatePropertyType.Orientation, OVRPlugin.Node.Head, OVRPlugin.Step.Render,
out rot))
                       headPose.orientation = rot;

               OVRPose invHeadPose = headPose.Inverse();
               Matrix4x4 invHeadMatrix = Matrix4x4.TRS(invHeadPose.position,
invHeadPose.orientation, Vector3.one);

               Matrix4x4 ret = centerEyeAnchor.localToWorldMatrix * invHeadMatrix;
                  return ret;
          }
 }
```

6.3. Appendix C: Environment Creation

6.3.1. Skybox Panoramic Beta Shader

```
Shader
"Skybox/PanoramicBeta
" {
                          Properties {
                            _Tint ("Tint Color", Color) = (.5, .5, .5, .5)
                            [Gamma] _Exposure ("Exposure", Range(0, 8)) = 1.0
                            _Rotation ("Rotation", Range(0, 360)) = 0
                            [NoScaleOffset] _Tex ("Spherical  (HDR)", 2D) =
                          "grey" {}
                            [KeywordEnum(6 Frames Layout, Latitude Longitude
                          Layout)] _Mapping("Mapping", Float) = 1
                            [Enum(360 Degrees, 0, 180 Degrees, 1)]
                          _ImageType("Image Type", Float) = 0
                            [Toggle] _MirrorOnBack("Mirror on Back", Float) = 0
                            [Enum(None, 0, Side by Side, 1, Over Under, 2)]
                          _Layout("3D Layout", Float) = 0
                          }

                          SubShader {
                            Tags { "Queue"="Background"
                          "RenderType"="Background" "PreviewType"="Skybox"
                          }
                            Cull Off ZWrite Off

                            Pass {

                                CGPROGRAM
                                #pragma vertex vert
                                #pragma fragment frag
                                #pragma target 2.0
                                #pragma multi_compile __
                           _MAPPING_6_FRAMES_LAYOUT

                                #include "UnityCG.cginc"

                                sampler2D _Tex;
                                float4 _Tex_TexelSize;
                                half4 _Tex_HDR;
                                half4 _Tint;
```

```
        half _Exposure;
        float _Rotation;
#ifndef _MAPPING_6_FRAMES_LAYOUT
        bool _MirrorOnBack;
        int _ImageType;
        int _Layout;
#endif

#ifndef _MAPPING_6_FRAMES_LAYOUT
        inline float2 ToRadialCoords(float3 coords)
        {
            float3 normalizedCoords = normalize(coords);
            float latitude = acos(normalizedCoords.y);
            float longitude = atan2(normalizedCoords.z,
normalizedCoords.x);
            float2 sphereCoords = float2(longitude, latitude) *
float2(0.5/UNITY_PI, 1.0/UNITY_PI);
            return float2(0.5,1.0) - sphereCoords;
        }
#endif

#ifdef _MAPPING_6_FRAMES_LAYOUT
        inline float2 ToCubeCoords(float3 coords, float3
layout, float4 edgeSize, float4 faceXCoordLayouts, float4
faceYCoordLayouts, float4 faceZCoordLayouts)
        {
            // Determine the primary axis of the normal
            float3 absn = abs(coords);
            float3 absdir = absn > float3(max(absn.y,absn.z),
max(absn.x,absn.z), max(absn.x,absn.y)) ? 1 : 0;
            // Convert the normal to a local face texture coord
[-1,+1], note that tcAndLen.z==dot(coords,absdir)
            // and thus its sign tells us whether the normal is
pointing positive or negative
            float3 tcAndLen = mul(absdir,
float3x3(coords.zyx, coords.xzy, float3(-
coords.xy,coords.z)));
            tcAndLen.xy /= tcAndLen.z;
            // Flip-flop faces for proper orientation and
normalize to [-0.5,+0.5]
            bool2 positiveAndVCross = float2(tcAndLen.z,
layout.x) > 0;
```

```
            tcAndLen.xy *= (positiveAndVCross[0] ?
absdir.yx : (positiveAndVCross[1] ? float2(absdir[2],0) :
float2(0,absdir[2]))) - 0.5;
            // Clamp values which are close to the face edges
to avoid bleeding/seams (ie. enforce clamp texture wrap
mode)
            tcAndLen.xy = clamp(tcAndLen.xy, edgeSize.xy,
edgeSize.zw);
            // Scale and offset texture coord to match the
proper square in the texture based on layout.
            float4 coordLayout = mul(float4(absdir,0),
float4x4(faceXCoordLayouts, faceYCoordLayouts,
faceZCoordLayouts, faceZCoordLayouts));
            tcAndLen.xy = (tcAndLen.xy +
(positiveAndVCross[0] ? coordLayout.xy :
coordLayout.zw)) * layout.yz;
            return tcAndLen.xy;
        }
#endif

         float3 RotateAroundYInDegrees (float3 vertex, float
 degrees)
         {
            float alpha = degrees * UNITY_PI / 180.0;
            float sina, cosa;
            sincos(alpha, sina, cosa);
            float2x2 m = float2x2(cosa, -sina, sina, cosa);
            return float3(mul(m, vertex.xz), vertex.y).xzy;
         }

         struct appdata_t {
            float4 vertex : POSITION;
            UNITY_VERTEX_INPUT_INSTANCE_ID
         };

         struct v2f {
            float4 vertex : SV_POSITION;
            float3 texcoord : TEXCOORD0;
 #ifdef _MAPPING_6_FRAMES_LAYOUT
            float3 layout : TEXCOORD1;
            float4 edgeSize : TEXCOORD2;
            float4 faceXCoordLayouts : TEXCOORD3;
            float4 faceYCoordLayouts : TEXCOORD4;
            float4 faceZCoordLayouts : TEXCOORD5;
 #else
            float2 image180ScaleAndCutoff : TEXCOORD1;
            float4 layout3DScaleAndOffset : TEXCOORD2;
 #endif
            UNITY_VERTEX_OUTPUT_STEREO
        };

        v2f vert (appdata_t v)
        {
            v2f o;
            UNITY_SETUP_INSTANCE_ID(v);

 UNITY_INITIALIZE_VERTEX_OUTPUT_STEREO(o)
 ;
            float3 rotated =
 RotateAroundYInDegrees(v.vertex, _Rotation);
            o.vertex = UnityObjectToClipPos(rotated);
            o.texcoord = v.vertex.xyz;
 #ifdef _MAPPING_6_FRAMES_LAYOUT
            // layout and edgeSize are solely based on texture
 dimensions and can thus be precalculated in the vertex
 shader.
          float sourceAspect = float(_Tex_TexelSize.z) /
float(_Tex_TexelSize.w);
          // Use the halfway point between the 1:6 and 3:4
aspect ratios of the strip and cross layouts to
          // guess at the correct format.
          bool3 aspectTest =
             sourceAspect >
             float3(1.0, 1.0f / 6.0f + (3.0f / 4.0f - 1.0f / 6.0f)
/ 2.0f, 6.0f / 1.0f + (4.0f / 3.0f - 6.0f / 1.0f) / 2.0f);
          // For a given face layout, the coordinates of the 6
cube faces are fixed: build a compact representation of
the
          // coordinates of the center of each face where the
first float4 represents the coordinates of the X axis faces,
          // the second the Y, and the third the Z. The first
two float componenents (xy) of each float4 represent the
face
          // coordinates on the positive axis side of the cube,
and the second (zw) the negative.
```

```
        // layout.x is a Boolean flagging the vertical cross
layout (for special handling of flip-flops later)
        // layout.yz contains the inverse of the layout
dimensions (i.e.. the scale factor required to convert from
        // normalized face coords to full texture
coordinates)
        if (aspectTest[0]) // horizontal
        {
          if (aspectTest[2])
          { // horizontal strip
            o.faceXCoordLayouts =
float4(0.5,0.5,1.5,0.5);
            o.faceYCoordLayouts =
float4(2.5,0.5,3.5,0.5);
            o.faceZCoordLayouts =
float4(4.5,0.5,5.5,0.5);
            o.layout = float3(-1,1.0/6.0,1.0/1.0);
          }
          Else
          { // horizontal cross
            o.faceXCoordLayouts =
float4(2.5,1.5,0.5,1.5);
            o.faceYCoordLayouts =
float4(1.5,2.5,1.5,0.5);
            o.faceZCoordLayouts =
float4(1.5,1.5,3.5,1.5);
            o.layout = float3(-1,1.0/4.0,1.0/3.0);
          }
        }
        else
        {
          if (aspectTest[1])
          { // vertical cross
            o.faceXCoordLayouts =
float4(2.5,2.5,0.5,2.5);
            o.faceYCoordLayouts =
float4(1.5,3.5,1.5,1.5);
            o.faceZCoordLayouts =
float4(1.5,2.5,1.5,0.5);
            o.layout = float3(1,1.0/3.0,1.0/4.0);
          }
          else
          { // vertical strip
```

```
                    o.faceXCoordLayouts =
    float4(0.5,5.5,0.5,4.5);
                    o.faceYCoordLayouts =
    float4(0.5,3.5,0.5,2.5);
                    o.faceZCoordLayouts =
    float4(0.5,1.5,0.5,0.5);
                    o.layout = float3(-1,1.0/1.0,1.0/6.0);
                  }
                }
                // edgeSize specifies the minimum (xy) and
    maximum (zw) normalized face texture coordinates that
    will be used for
                // sampling in the texture. Setting these to the
    effective size of a half pixel horizontally and vertically
                // effectively enforces clamp mode texture
    wrapping for each individual face.
          o.edgeSize.xy = _Tex_TexelSize.xy * 0.5 /
o.layout.yz - 0.5;
          o.edgeSize.zw = -o.edgeSize.xy;
#else // !_MAPPING_6_FRAMES_LAYOUT
          // Calculate constant horizontal scale and cutoff
for 180 (vs 360) image type
          if (_ImageType == 0)  // 360 degree
            o.image180ScaleAndCutoff = float2(1.0, 1.0);
          else  // 180 degree
            o.image180ScaleAndCutoff = float2(2.0,
_MirrorOnBack ? 1.0 : 0.5);
          // Calculate constant scale and offset for 3D
layouts
          if (_Layout == 0) // No 3D layout
            o.layout3DScaleAndOffset = float4(0,0,1,1);
          else if (_Layout == 1) // Side-by-Side 3D layout
            o.layout3DScaleAndOffset =
float4(unity_StereoEyeIndex,0,0.5,1);
          else // Over-Under 3D layout
            o.layout3DScaleAndOffset = float4(0, 1-
unity_StereoEyeIndex,1,0.5);
#endif
          return o;
    }

    fixed4 frag (v2f i) : SV_Target
    {
```

```
#ifdef _MAPPING_6_FRAMES_LAYOUT
            float2 tc = ToCubeCoords(i.texcoord, i.layout,
i.edgeSize, i.faceXCoordLayouts, i.faceYCoordLayouts,
i.faceZCoordLayouts);
#else
            float2 tc = ToRadialCoords(i.texcoord);
            if (tc.x > i.image180ScaleAndCutoff[1])
                return half4(0,0,0,1);
            tc.x = fmod(tc.x*i.image180ScaleAndCutoff[0],
1);
            tc = (tc + i.layout3DScaleAndOffset.xy) *
i.layout3DScaleAndOffset.zw;
#endif

            half4 tex = tex2D (_Tex, tc);
            half3 c = DecodeHDR (tex, _Tex_HDR);
            c = c * _Tint.rgb * unity_ColorSpaceDouble.rgb;
            c *= _Exposure;
            return half4(c, 1);
        }
        ENDCG
    }
}


CustomEditor "SkyboxPanoramicBetaShaderGUI"
Fallback Off }
```

6.3.2. Skybox Panoramic Beta Shader GUI

```
using
UnityEngin
e;
            using UnityEditor;
            using UnityEditor.AnimatedValues;

            namespace UnityEditor
            {
                internal class SkyboxPanoramicBetaShaderGUI : ShaderGUI
                {
                    readonly AnimBool m_ShowLatLongLayout = new AnimBool();
                    readonly AnimBool m_ShowMirrorOnBack = new AnimBool();
            readonly AnimBool m_Show3DControl = new AnimBool();

            bool m_Initialized = false;

            public override void OnGUI(MaterialEditor materialEditor,
        MaterialProperty[] props)
            {
              if (!m_Initialized)
              {

        m_ShowLatLongLayout.valueChanged.AddListener(materialEditor.Re
        paint);

        m_ShowMirrorOnBack.valueChanged.AddListener(materialEditor.Re
        paint);

        m_Show3DControl.valueChanged.AddListener(materialEditor.Repaint
        );
                  m_Initialized = true;
              }

          // Allow the default implementation to set widths for
  consistency for common properties.
          float lw = EditorGUIUtility.labelWidth;
          materialEditor.SetDefaultGUIWidths();
          ShowProp(materialEditor, FindProperty("_Tint", props));
          ShowProp(materialEditor, FindProperty("_Exposure", props));
          ShowProp(materialEditor, FindProperty("_Rotation", props));
          ShowProp(materialEditor, FindProperty("_Tex", props));
          EditorGUIUtility.labelWidth = lw;

          m_ShowLatLongLayout.target = ShowProp(materialEditor,
  FindProperty("_Mapping", props)) == 1;
          if
  (EditorGUILayout.BeginFadeGroup(m_ShowLatLongLayout.faded))
          {
            m_ShowMirrorOnBack.target = ShowProp(materialEditor,
  FindProperty("_ImageType", props)) == 1;
            if
  (EditorGUILayout.BeginFadeGroup(m_ShowMirrorOnBack.faded))
            {
              EditorGUI.indentLevel++;
```

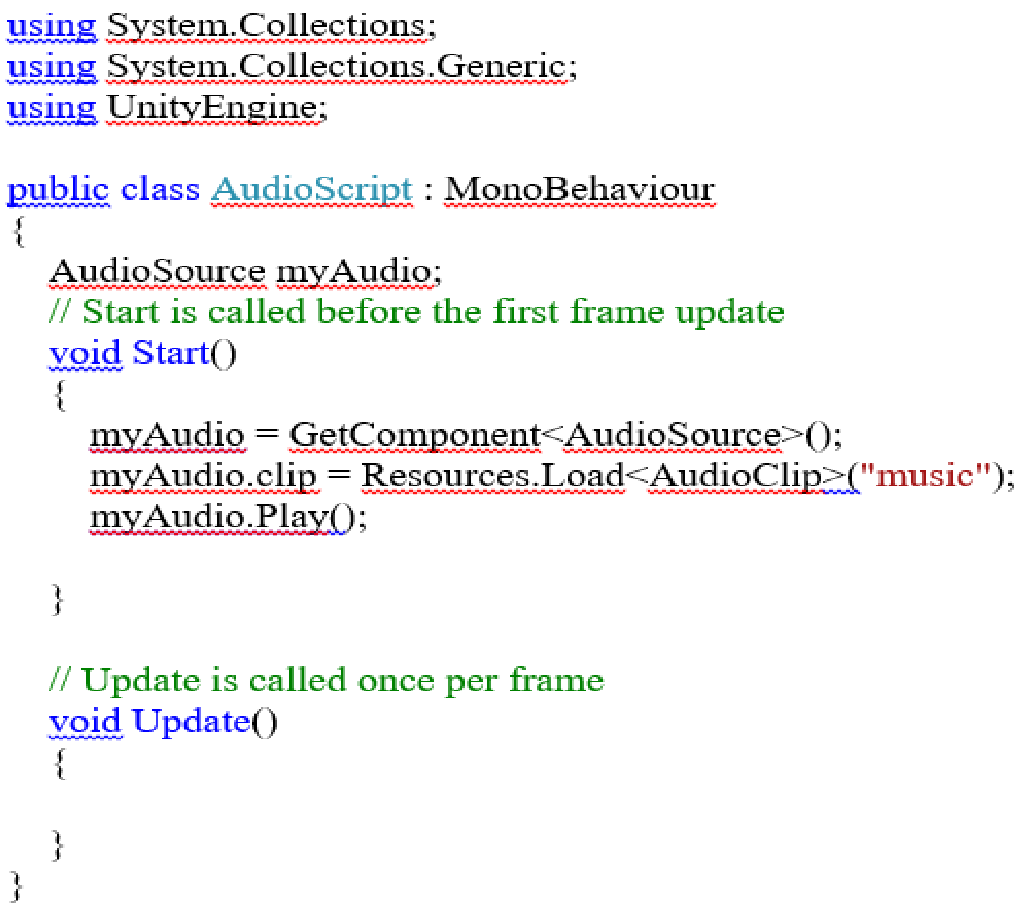
```
                    ShowProp(materialEditor,
FindProperty("_MirrorOnBack", props));
                    EditorGUI.indentLevel--;
                }
                EditorGUILayout.EndFadeGroup();

                // No 3D settings unless PlayerSettings have VR support.
                m_Show3DControl.value =
PlayerSettings.virtualRealitySupported;
                if
(EditorGUILayout.BeginFadeGroup(m_Show3DControl.faded))
                    ShowProp(materialEditor, FindProperty("_Layout",
props));
                EditorGUILayout.EndFadeGroup();
            }
            EditorGUILayout.EndFadeGroup();

            // Let the default implementation add the extra shader
properties at the bottom.
            materialEditor.PropertiesDefaultGUI(new
MaterialProperty[0]);
        }

        private float ShowProp(MaterialEditor materialEditor,
MaterialProperty prop)
        {
            materialEditor.ShaderProperty(prop, prop.displayName);
            return prop.floatValue;
        }
    }
}
```

6.3.3. *Audio Script*



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AudioScript : MonoBehaviour
{
    AudioSource myAudio;
    // Start is called before the first frame update
    void Start()
    {
        myAudio = GetComponent<AudioSource>();
        myAudio.clip = Resources.Load<AudioClip>("music");
        myAudio.Play();

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

6.4.Appendix D: *Graphical User Interface*

6.4.1. *Main Menu Script*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenuScript : MonoBehaviour
{
   public GameObject VideosPanel;


   public void OpenMCSMainRoadScene()
   {
      SceneManager.LoadScene("MCSMainRoad");
   }

   public void GoToSelectVideo()
   {
      VideosPanel.SetActive(true);

   }

   public void GoBackToMainMenu()
   {
      VideosPanel.SetActive(false);

   }

   // Start is called before the first frame update
   void Start()
   {

   }
  // Update is called once per frame
  void Update()
  {

  }
```

```
    public void OnApplicationQuit()
    {

        Application.Quit();
    }
}
```

MCS main road control

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;


public class MCSMainRoadUIControls : MonoBehaviour
{

    public void GoBackToMainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

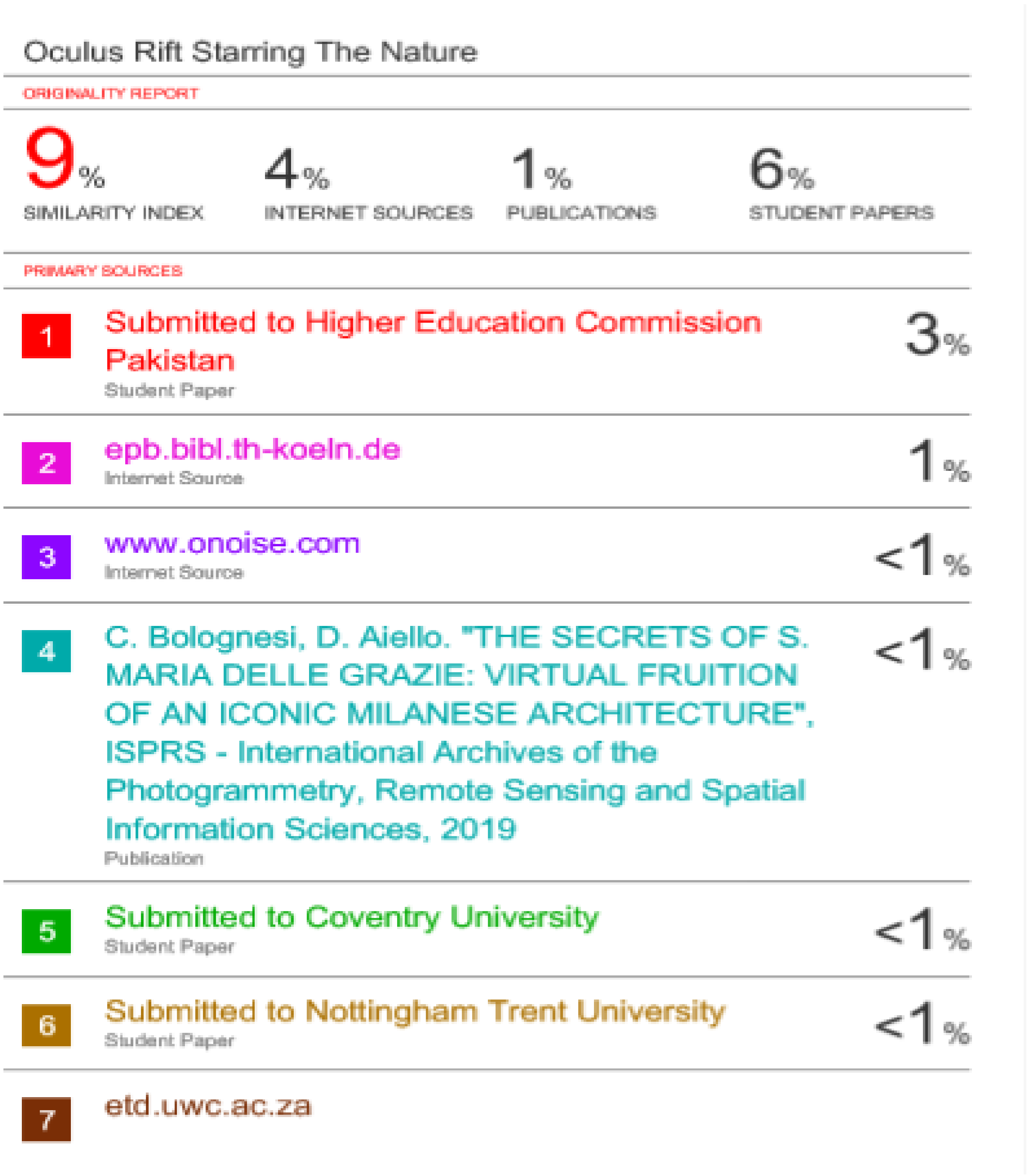
6.4.2. UI Controller

```
using Systems.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
```

```
public class MCSMainRoadUIControls;
MonoBehaviour
{
          Public void GoBackToMainMenu()
          {
            SceneManager.LoadScene("MainMenu");
          }
          //start is called before the first frame
update
          void start()
          {
          }
          //update is called once per frame
          void update()
          {
          }
}
```

7. BIBLIOGRAPHY

1. Ahola, S. (2019). DEVELOPING A VIRTUAL REALITY.
2. Aseeri, S. A. (2013). Virtual Reality Interaction Using Mobile Devices.
3. GawnYong Park, J. K. (2017). Development of virtual reality walking interface using.
4. Larsen, T. F. (2018). Virtual Reality games and gamified exercises in physioterapeutic treatment of non-specific low back pain patients with kinesiophobia.
5. Moreno, J. T. (2017). Virtual Reality Interaction Using Mobile Devices.
6. Subash, G. (2019). VIRTUAL REALITY ANALYSIS.
7. Wang, Y. (2020). AN OBJECT-ORIENTED SOFTWARE FRAMEWORK FOR IMMERSIVE VIRTUAL REALITY EXERGAMES.



Oculus Rift Starring The Nature

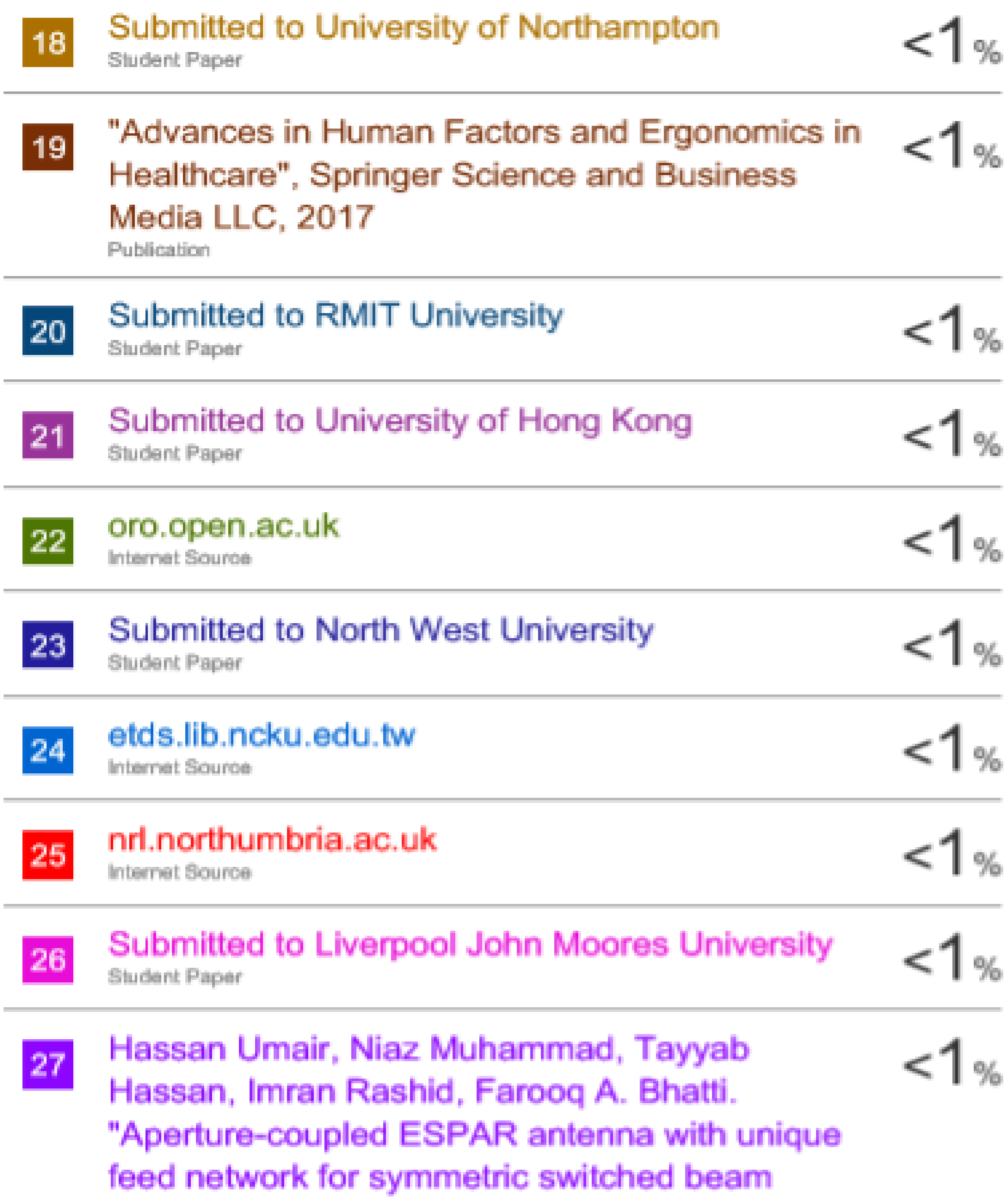
ORIGINALITY REPORT

9%	4%	1%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	3%
2	epb.bibl.th-koeln.de Internet Source	1%
3	www.onoise.com Internet Source	<1%
4	C. Bolognesi, D. Aiello. "THE SECRETS OF S. MARIA DELLE GRAZIE: VIRTUAL FRUITION OF AN ICONIC MILANESE ARCHITECTURE", ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2019 Publication	<1%
5	Submitted to Coventry University Student Paper	<1%
6	Submitted to Nottingham Trent University Student Paper	<1%
7	etd.uwc.ac.za	

	Internet Source	<1%
8	doaj.org Internet Source	<1%
9	Submitted to University of Huddersfield Student Paper	<1%
10	Submitted to University of Northumbria at Newcastle Student Paper	<1%
11	www.slideshare.net Internet Source	<1%
12	certs.lbl.gov Internet Source	<1%
13	Submitted to University of Queensland Student Paper	<1%
14	hdl.handle.net Internet Source	<1%
15	Entregado a University of Sheffield el 2012-09-05 Student Paper	<1%
16	wiki.hackpgh.org Internet Source	<1%
17	Submitted to University of Auckland Student Paper	<1%



18	Submitted to University of Northampton Student Paper	<1%
19	"Advances in Human Factors and Ergonomics in Healthcare", Springer Science and Business Media LLC, 2017 Publication	<1%
20	Submitted to RMIT University Student Paper	<1%
21	Submitted to University of Hong Kong Student Paper	<1%
22	oro.open.ac.uk Internet Source	<1%
23	Submitted to North West University Student Paper	<1%
24	etds.lib.ncku.edu.tw Internet Source	<1%
25	nrl.northumbria.ac.uk Internet Source	<1%
26	Submitted to Liverpool John Moores University Student Paper	<1%
27	Hassan Umair, Niaz Muhammad, Tayyab Hassan, Imran Rashid, Farooq A. Bhatti. "Aperture-coupled ESPAR antenna with unique feed network for symmetric switched beam	<1%

radiation patterns", International Journal of Microwave and Wireless Technologies, 2016
Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		