School of Electronic Engineering and Computer Science

# Final Report

**Programme of study:**

BSc FT Computer Science

## Project Title:

# Virtual Reality Audio: A Visual Experience

**Supervisor:** Dr Luk Arnaut

**Student Name**:

Mohammed Belal Rashid

Final Year
Undergraduate Project 2017/18

Queen Mary
University of London

Date: 23/04/2018

# Abstract

Music visualisation is the electronic generation of objects based on music. The visualisation adds new depth to the music by allowing you to “see” what is being played. Visualisation has been delved into thoroughly in the two-dimensional space that is the traditional desktop environment. Windows media player has a selection of appealing visualisations that have a pure entertainment value. This led me wondering about three-dimensional representations of music. Virtual reality is a medium that allows the user to engross themselves in a 3D world like the real world.

The goal was to acquire the skills to create a music visualizer with a user interface, in virtual reality, that boasts a high entertainment value. The project started off as a hobby project with personal goals that transformed into something more. The project is created in Unity for the Oculus Rift. The application fulfils the main goal. It provides the user with a functional music visualiser with a user interface in virtual space that can control music selection and visualize audio information from different sources.

The motivation behind this project stems from poor quality, free-to-acquire, visualizers not living up to the brief that this project is trying to achieve. Learning from the mistakes of these applications provides the project with insight on what to avoid. Furthermore, the project targets dedicated music listeners that want a seated experience.

The application runs on Windows for the Oculus. The project has concluded one development cycle and has provided me with research to further improve the visualisation concept. Testing concluded that psychedelic visualizations that aid a hallucinogenic experience should be a primary focus of virtual reality music visualizers.

# Contents

# Chapter 1 INTRODUCTION

Music is a universal enjoyment no matter what culture you are from. Watching a visualizer for a piece of music helps amplify a person's connection or feelings with that piece. I first came across music visualisation during the early days of windows media player. I was highly engrossed with the different variety of patterns you could make and how the visualisation always made the music feel better. This love and use of music made me continue to look for different mediums of visualization. I delved into the virtual reality space searching. It left me underwhelmed with the belief that there was more to explore. I decided to bring a music visualizer to the virtual reality space after delving into the current experiences that the virtual reality community provides.

The finalized idea for my project is a virtual reality music visualizer with a user interface. It should have a variety of different visualisations and cause the user little discomfort. The targeted users are people who enjoy listening to music while sitting down as their primary or only activity. The user interface should be intuitive and fully responsive.

The challenge that this project provides is development in the new technology that is virtual reality. Prior to this project, I had not developed for virtual reality. The project involves 3-dimensional development in Unity (game engine) with the programming language C Sharp. I had zero knowledge of both tools prior to starting the project. The visualisation aspect of the project requires knowledge of signal processing which I acquired for the project.

## 1.1 AIMS

The **primary aim** of this project is to develop a music visualizer in virtual reality. The music application should take a piece of music as input to update a virtual space of 3-dimensional objects that are graphically designed to be entertaining. Visualization can be considered achieved when the music's rhythm has a high degree of correlation with the graphic. The project should provide a variety of different graphical representations that can be navigated by a user interface in virtual reality. The idea behind providing variety is that it improves reusability and increases the entertainment value. The involvement of the user with the application is sitting down whilst listening to music.

The highest-level objectives of this project can be described into two components:

- Fully Supported Virtual Reality Music Visualization
- Optimised Virtual Reality User Interface

## 1.2 SCOPE

This project focuses on the Oculus virtual reality platform with the Oculus Rift. It targets development in Unity3D in Windows 10 with C Sharp. In general, the acquisition of paid-for applications or assets is beyond the scope of this project.

I identified popular applications in the Oculus Store, Steam and used word-of-mouth to find, at the time, the applications listed in my analysis of existing applications. However,

due to the sheer size of the Oculus and Steam stores I judged the most popular based on similarity to my objectives. Data was gathered from user reviews and my own experiences.

Visualisation, after processing, is highly dependent on the artistic capability of the creator. I am poor in this capacity so I have decided to generally use the standard Unity 3-dimensional models (primitives) to create my environment. Any paid for 3-dimensional or custom models are generally beyond the scope of this project.

## 1.3 Overview

This report assumes the reader has no knowledge of Virtual Reality or Unity but has a simple understanding of programming. It provides the reader with the background research necessary to understand what is trying to be achieved. It describes problematic visualisers and derives a requirement specification. It shows what was implemented with regards to workflow. Code and application screenshots are provided where necessary.

# Chapter 2 RESEARCH

This section provides a view into the research I carried out to achieve the goals I set out with for my project. It also discusses the virtual reality component/tool of the project as well as the game engine used to create the music visualization experience.

## 2.1 What is Virtual Reality?

Virtual reality is a fully immersive, computer simulated environment that tricks the user's brain into believing that they are inside the computer simulation in which the person can explore and interact. One of the most popular ways to experience virtual reality is through a headset. These devices use stereoscopic displays to make what you perceive three dimensional and provide depth to the image you see [1]. The headsets can track a user's movement, including head and eye movement, which allows the image in the headset to change with the user's perspective. Coupled with good sound and high levels of virtual interactivity brings about an alteration of the user's perception of reality to provide a realistic experience of a virtual world.

There are three main classifications in virtual reality based on experience. **Room-scale** experiences are designed with the intent of having access to an entire room as a play area. The user is capable of free movement in virtual reality that corresponds to the size of the room. These experiences are highly immersive but are difficult to design. **Standing** experiences are designed with the intent of creating a standing experience. Content is placed around the focal point that is the user. A painting application in virtual reality is an example of this type of experience. Applications that support this type of experience are also usually room-scale compatible. **Seated** experiences are designed with the intent to satisfy a smaller play area. The user in these experiences is sitting down with content placed in front of them. Content is placed in proximity to the user as the user is not expected to move very much. An example includes a theatre in virtual reality with users in seats watching a movie.

It was important to find the correct classification for my application. I have chosen to create a **seated** experience as I do not have access to a large play area and I believe users will want to enjoy the music, not move and enjoy the visualization.

## 2.2 Virtual Reality Platforms/Devices

To get into virtual reality you need a device that lets you view it. The market has many mobile-access variants of virtual reality that utilise your phone being plugged into a headset. Samsung Gear VR is a very popular platform for an affordable virtual reality experience. I have decided to develop for a personal computer, not a mobile platform, so the Samsung Gear VR is not a suitable choice for my needs. There are three suitable devices that follow my development choices. Samsung Odyssey/Windows Mixed Reality, HTC Vive and the Oculus Rift. These devices each have two screens, one for each eye, that work together to produce a stereoscopic display.

**Windows Mixed Reality** headsets are a series of virtual reality headsets produced for Microsoft by many different manufacturers. At the top of the pile is the **Samsung**

**Odyssey** which is a consideration. It consists of a headset and two wireless controllers. The Odyssey has two 1400 x 1600 AMOLED displays with a 110-degree field of view and built-in AKG headphones which is a benefit for music applications. These devices use a tracking method referred to as “inside out”. This tracking employs the multitude of cameras and sensors present on the headset to orientate the user in virtual space. This tracking method comes with obvious issues such as controllers losing visibility to the system if they are not in front of the camera on the headset. The Odyssey supports integrated graphics as a minimum requirement for a low-quality mode. It requires a Nvidia 1050 for the full system. The Windows Mixed Reality headsets use the OpenVR software development kit.

**HTC Vive** is a headset manufactured by HTC with the support of Valve. It consists of a headset, two wireless controllers and two “lighthouses”. The Vive has two OLED 1080 x 1200 displays with a 110-degree field of view and built-in headphones and mic. The system uses “lighthouses” to track the user that supports 15x15 feet with 2 sensors. These sensors support room-scale with a combination of pulses and sweeping actions with infra-red light. The Vive is built with room-scale experiences in mind. The placement of the sensors is most beneficial at opposite corners of the room. The controllers are heavy and very noticeable with touch-sensitive pads under the thumb and triggers buttons that act as primary selection. The controller acts like a single pointing finger in my opinion. The minimum requirement to run this headset is a Nvidia 970. HTC Vive uses the OpenVR software development kit.

**Oculus Rift** is a headset manufactured and developed by Oculus VR. It consists of a headset, two wireless controllers and two sensors. The Rift has two OLED 1080 x 1200 displays with a 110-degree field of view and built-in headphones and mic. Oculus supports 8 x 8 feet with 3 sensors. The system uses a “constellation” method to track with the infra-red LED sensors. The method consists of infra-red LED’s in the headset and the controllers that are picked up by the constant tracking of the cameras/sensors. The Rift is built with sitting or standing experiences in mind. Current support for room-scale can only be achieved with the addition of multiple sensors with 4 or 5 being optimal. The placement of sitting experiences is sensors in front of the user 2 metres apart. Room-scale experiences require sensors at each corner. The Oculus touch controllers have a joystick and button setup and allow for some simple gesture mapping based on how you’re holding the controller. The design allows the pair of controllers to function more like your real hands capitalizing on internal tracking sensors and haptic feedback which adds a greater sense of immersion. They are light and feel comfortable to hold. The Oculus implements “asynchronous spacewarp” technology. “The spacewarp system (which is built into the Oculus runtime) takes the two previous frames generated by software, analyses the difference, and calculates a spatial transformation that can generate a "synthetic frame" based on the current head translation and movement.” [10] This technology lowers the minimum requirement to run the Oculus to a Nvidia 960. Oculus Rift uses the Oculus software development kit.

I have decided to develop for the **Oculus Rift** primarily because my application is to be designed as a seated experience which is what the Rift is designed for. The Vive has the best tracking out of all three with the Rift coming second. The Rift's shortcoming is not relevant as I am not creating a room-scale application. The Oculus has the controllers out of all three which is an added benefit.

## 2.3 Virtual Reality Engine – What to Develop With?

Currently, there are two big engines that support virtual reality development, Unreal Engine 4 and Unity. This section compares the two.

**Unreal Engine 4** is a very powerful engine that is the core of many of the major game titles of recent years. It provides a very strong 3-dimensional development stage and only supports C++. The engine is available for free to students with 5% of any revenue generated going to Unreal. There is a large marketplace with many paid assets but limited free ones. The learning curve for the engine is quite steep with limited tutorials explaining the advanced controls. Overall, the community regards the engine with a high professional standard with professional developers.

**Unity** is a cross-platform game engine that supports 2-dimensional and 3-dimensional graphics with the ability to control asset behaviour with C# scripts. [2] It is a starting platform for many new VR developers. The Unity community consists of many hobbyist developers that provide a lot of support. There is a large asset store with many free resources. The community has a lot of beginner friendly tutorials that provide an easy to follow learning experience.

Considering the fact, I have basic knowledge of virtual reality development, I have chosen Unity as my development tool. The Unity engine is beginner friendly with easy to find development help.

## 2.4 User Interfaces In Virtual Reality With Unity

This section introduces the concept of Unity scenes, type of UI in Virtual Reality and how that UI should be optimised.

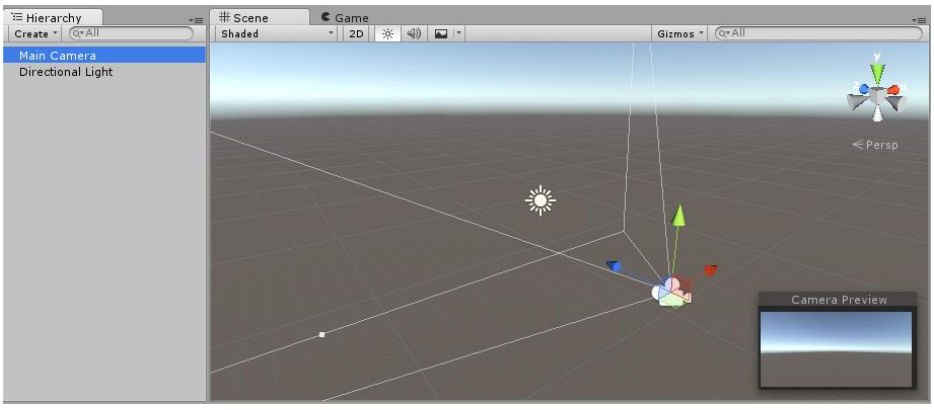



*Figure 1 - Empty Scene [13]*

Figure 1 is a scene in Unity as an empty world space. The world space is an empty environment for a creator to develop in. For a UI to exist in virtual reality, it needs to be included in this space.

The types of UI include diegetic, non-diegetic, spatial and meta [15]. In a non-VR application, a **non-diegetic** approach is usually used. This is when UI is overlaid on the top of the screen. An example includes showing the score on the heads-up-display. This type of UI does not exist in the world but makes sense for the player to view in the context of the application. Non-diegetic UI is usually represented in 2D. It cannot be used for a VR application as our eyes cannot focus on something that is so close. A **diegetic** UI exists in the world instead of being overlaid. An example includes pressing a button to make a character look at their watch to view the time.

A **meta** UI is like a non-diegetic is the sense that are not visualized spatially for the viewer. An example includes blood spatter displayed on the camera to indicate damage. A spatial UI presents elements in the 3D space with either diegetic or non-diegetic properties. Virtual Reality UI's are created spatially. Unity has a UI module with text, images, buttons, sliders, etc that need to be held on a canvas object. The canvas object is placed in the 3D environment creating a spatial UI.

**Spatial** UI's in VR come with problems that need to be addressed to create an optimised experience. VR headsets have a very high resolution so a UI element that is small in pixel matter will have major pixilation. To combat this, UI elements need to be designed with large fonts and scaled down. Element size and positioning play a key role in how the UI is perceived. UI elements should be in view of the camera at a comfortable reading distance. Clipping important UI elements to the camera ensures the UI is always in view.

Elements should be between two to four metres away from the camera view. The reason for this is due to the **vergence-accommodation conflict**. VR unfortunately does not simulate the real world, in the real world when you look at an object your eyes verge towards the object and your lenses change shape to bring the object into focus. In the real world these two processes focus on the same object. The problem in VR occurs in the simulation of depth, we will either verge to a faraway object or a close object depending on where the object is in virtual space but then we focus on a screen that is incredibly close to our face. Figure 2 is a graphical representation.

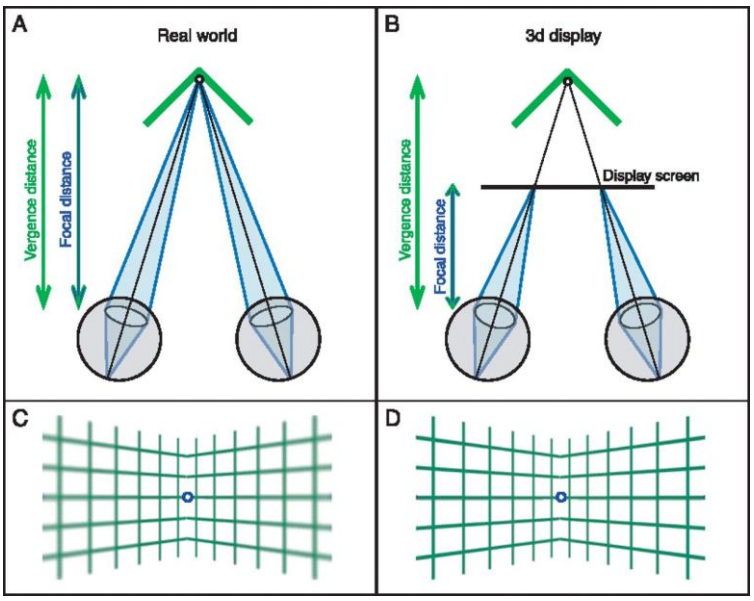



*Figure 2 - Vergence-accommodation [9]*

This causes problems like eye-strain, eye fatigue or motion sickness as you are verging at different depths and having to accommodate. One way to combat this, is to put elements at the accommodation point where we are focusing our eyes as we know roughly the distance between the user and the screens of the headset. Putting elements in this space combats eye fatigue as you verge and accommodate to the same place just like the real world. Designing UI elements that will hold the user focus for a long time should avoid the vergence-accommodation conflict. The accommodation point for oculus in general is between two to four metres depending on the headset. In Unity every unit is one metre.

## 2.5 What is Audio/Music Visualisation?

When a person talks about audio, they are often referring to a recording that they have listened to of sound. Audio is the acoustic, mechanical, or electrical frequencies corresponding to normally audible sound waves which are of frequencies approximately from 15 to 20,000 hertz [3]. 15 to 20,000 hertz is the safest range that humans can perceive of sound without loss or damage. Anything above the higher end of this range is known as ultrasound. Anything below this range is known as infrasound.

**Sound** is a vibration that manifests itself as an audible wave of pressure. The waves consist of multiple sinusoid and co-sinusoid waves at different frequencies. A sinusoid is a sine wave which is a continuous wave. For something to interpret these waves a translation tool is required. We, as humans, have ears and a brain whereas a computer has a sound card and a central processing unit. An audio file that is played on a computer is picked up by the sound card to be translated into sinusoid waves creating vibrations that is outputted by headphones or speakers for the user to enjoy. We can decipher the frequency (speed of the vibration that determines pitch) and amplitude

(size of the vibration that determines loudness) of the sound waves to gather information to create a visualisation. This is achieved with the use of the Fourier transform algorithm.

An early music visualizer is the Cubic Player (1994) [4]. This player uses a spectrum analyser that uses the **Fast Fourier transformation algorithm** to gather information on the audio spectrum of the music data. The Fast Fourier transform is a discrete version of the continuous Fourier Transform. It decomposes audio into the pure frequencies and amplitudes that make it up. Fourier transform breaks a signal into sine waves of different amplitudes and frequencies [5]. You can divide this spectrum into sub-categories:

Sub bass: 20 to 60Hz
Bass: 60 to 250Hz
Low midrange: 250 to 500Hz
Midrange: 500 to 2000Hz
Upper midrange: 2000 to 4000Hz
Presence: 4000Hz to 6000Hz
Brilliance: 6000Hz to 20000Hz [6]

Fourier Transform is a function. Fast Fourier Transform is an algorithm. When looking at real-world signals, you usually view them as a voltage changing over time. This is referred to as the time domain. A single Fast Fourier Transform is used when you want to understand the frequency spectrum of a signal from its time domain representation. The use of just the Fast Fourier Transform to produce a visualisation from a signal would be poor as it does not account for when the frequency components occurred in time. This is important as it is essential for rhythm to be visualised in real-time. To localise in time a short-time Fourier transform or “windowed” transform needs to be used. A short-time Fourier transform divides a longer time signal into shorter segments of equal length and then computes the Fourier transform separately on each segment. This reveals the Fourier spectrum on each shorter segment.

## 2.6 AudioSource.GetSpectrumData(samples, channel, window);

This section is very short but its inclusion states the importance of the function being explored. Without a music analyser, music visualization is not possible.

Unity can read and translate audio with GetSpectrumData(float[] samples, int channels, FFTWindow window) [11]. The function divides the data into an array of samples using Fourier Transform. The channel can either be 0 or 1 referring to left and right which can be combined to output stereo data. FFTWindow can be either Rectangular, Triangular, Hanning, Hamming, Blackman, or Blackman-Harris with each supporting a different a different effect suitable for different applications. A window weighting function is required to reduce spectral leakage. This phenomenon of leaking energy from peak value to other samples is called Spectral leakage.

## 2.7 WINDOWING IN UNITY

This section explores and explains the FFTWindow of GetSpectrumData from the previous section to find the best possible implementation for a visualisation.

If you pass a signal trough a linear system, the output will have not any frequency that was not already present at the input. However, if the system is not linear, the output will have new frequencies not present in the original input. These new frequencies are the spectral leakage. Music is often of the form where the start and end points are not equal. Windowing helps reduce leakage by multiplying the data in the time domain which brings the start and end points closer.

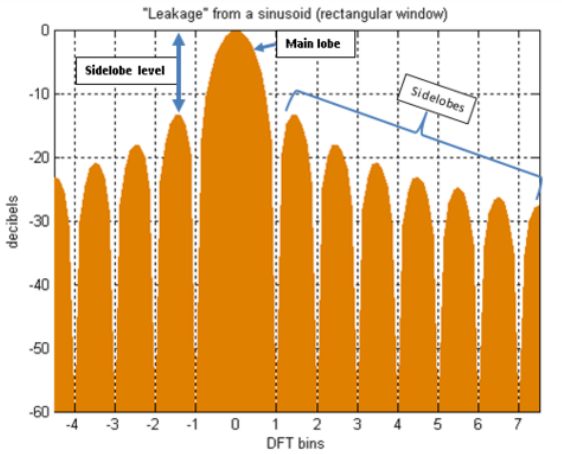



*Figure 3 – Leakage [14]*

Figure 3 represents a rectangular (no filter is applied) window of a sinusoid wave. Leakage is clearly seen in the bins that contain the sidelobes as these values are not in the original frequency. The bin that is relevant for measure is the main lobe.

Figure 4 is a comparative look into the supported windows of the GetSpectrumData function of Unity. Looking at each Fourier Transform window you can sort them into the following categories to filter leakage:

High filtered: Blackman and Blackman-Harris
Medium filtered: Hamming and Hann
Low filtered: Rectangular and Triangular

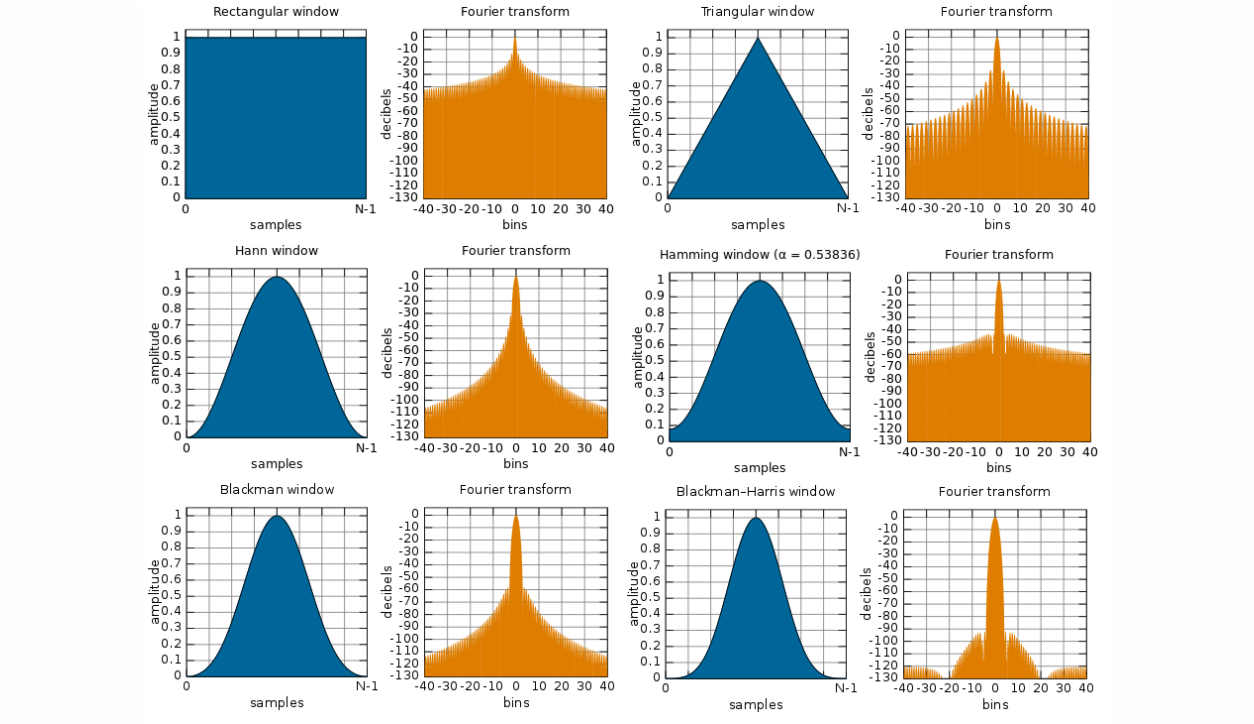



*Figure 4 – Windows [14]*

A smooth drop off where close frequencies are still present but anything else is highly filtered is best for a visualisation. The **Blackman-Harris window** is the best suited for my needs. The other windows each have their own purposes. An application that requires beat detection would best be suited with a Triangular window as it is good at detecting a spike in a single frequency.

## 2.8 SUMMARY

In this chapter I have explored what virtual reality is and the medium in which you can enter. I have decided on a development platform and researched spatial user interfaces in virtual reality. I have defined how we perceive audio and how we can use that audio data to make a visualization with the fast Fourier transform algorithm. I have explored an optimised route to make use of the algorithm by using Unity's get spectrum data function. I have researched the windows available to this function to decide on an optimal filter.

# Chapter 3 ANALYSIS AND REQUIREMENTS

This section delves into an analysis of existing virtual reality supported visualisers. It provides my own experience and provides insight into the decided features of my application by deriving requirements from the existing applications.

## 3.1 GROOVR

GrooVR is a music visualizer from Presence Labs that classes itself as “Music Driven Virtual Reality™” [7] that is available for Gear VR, Rift and Daydream. It promotes experiences with pre-set environments that react to music. Most of the environments, unfortunately, do not react to the music very well. They primarily work independently of the music and just play out with small visualizations that react to the music. It has great visuals but is not fully adaptive. It is like watching a 3-dimensional simulation play out that is the same every time. GrooVR has music support for SoundCloud and Spotify as well as local files.


*Figure 5 - GrooVR*

GooVr has a very intuitive and clean head-up display as shown in figure 4. The library is on the left and on the right, is the experiences with the current track in the middle.

## 3.2 INTONE

“Intone is an audio visual interactive VR experimental experience. The objects move in response to your voice and gaze input.” [8]

This experience is not determined by music but by user microphone input. The user gets complete control over the deep space environment they are flung into. Objects are brought into existence by the user’s voice dependant on the noises being made. The objects can be manipulated by the user’s gaze. Gaze-type input feels very natural. The application gives the user freedom to look at what they want, ignoring visualisations if they so choose. The application does not have a head-up display or any navigation. It is only one experience.

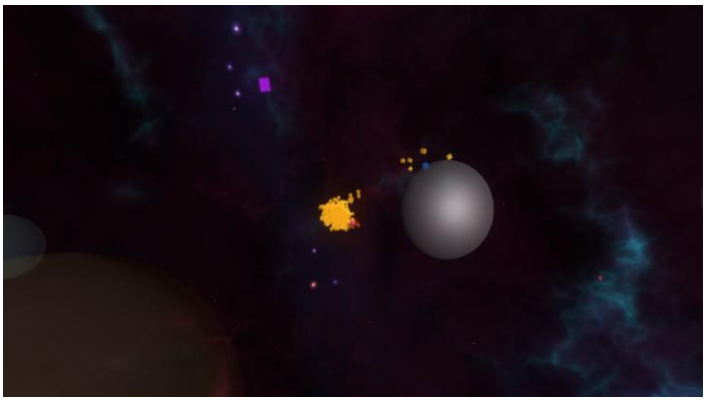


*Figure 6 – Intone*

The application does not have user interface and it can be odd but fun experience making weird noises to see what shapes you can conjure.

## 3.3 PLANE9

Plane9 is a 3D visualizer that has over 250 predefined scenes that can combine with transitions to form a view of the audio [12]. It supports Oculus and Vive but it does not have controller support or user interface support in virtual reality. It is sound-sensitive to whatever is currently being played to produce a visualisation. This is advantageous in the sense that the user does not have to provide a music file to be managed by the application. It allows fluidity in where the audio can be acquired from.

*Figure 7 - Plane9*

It is an adaptation of the desktop 2-dimensional visualisations that the program was originally created for so some of the visualisations are on a 2-dimensional space in 3-dimensions. Some of the visualisations also have no clear indication that the beat of the music is being followed with more focus on transitions.

## 3.4 VR AUDIO VISUALIZER

VR Audio Visualizer is a VR application available in the Oculus Store.


*Figure 8 - VR Audio Visualizer*

The application has six visualizations with six pre-selected songs. You cannot choose your own music. It utilizes keyboard controls to navigate scenes, it does not utilise VR interaction. There is no way to add your own music to the application. You cannot customise the visualisations in any way. The interface is very buggy with the keyboard controls.

## 3.5 Conclusion

From the experience of using all the above applications, I have derived the functional requirements, split into primary and secondary, and non-functional requirements derived from background research.

### 3.5.1 Primary Functional Requirements

These are the minimal requirements for the project to function and for the objective of the project to be achieved.

- Translate Music into the Virtual Environment: The application must take the user into a virtual reality environment and visualize music in a 3D space with 3D objects. The visualisation should correspond to the different frequencies of the music in a meaningful manner.

- User Interface in Virtual Reality: The application must have a User Interface in Virtual Reality that controls music management. A music player with play, pause, next, previous, shuffle, etc.

### 3.5.2 Secondary Functional Requirements

These are additional features that go beyond the minimal specification of the project but are important or beneficial for a good experience. These will be implemented after successful completion of primary goals.

- Customisable visualizations: The application should allow the user to tailor the visualization to personal preference. This could include colours, contrast, speed, rotation, physics, etc.

- Add and control music: The application should allow the user to add their own music through a user interface in virtual reality or load a playlist of music from a directory on disk. They should be able to control this music.

- Optimised User Interface: The application should include a user interface that is responsive, visually appealing, easily defined and not pixelated. The user should be provided with a form of feedback to show they are using an interface.

- Variety of Visualizations: The application should give the user the option to choose multiple visualizations from a virtual reality user interface.

- Microphone device Visualization: The application should allow visualization to use data from audio devices on the user's personal computer.

### 3.5.3 Non- functional Requirements

- Oculus support: The application must be able to run on the oculus platform with the highest supported resolution. It must not lag when music is being played. It must be fully functional with the minimum Nvidia 970 requirements of the Oculus

- Efficient speeds: The application must transition between visualisations at a fast pace without disrupting the user experience. The application must load music files from disk at a fast pace. The application must change between music files at a fast pace.

- Efficient Storage: The application must not overload memory while recording microphone audio data. The application must store music data for efficient retrieval.

- Optimised User Interface and Visualisations: The application must not cause motion sickness or considerable eye-fatigue.

## 3.6 SUMMARY

This chapter reviewed existing applications and identified the requirements for my application. Primary and secondary functional requirements were defined as well as non-functional requirements.

# Chapter 4 IMPLEMENTATION

This section explains my journey in creating my project from the beginning to the end with insights into my thought process.

## 4.1 INTRODUCTION

Unity 5.0 has in-built virtual reality support which is a necessity for virtual reality development. The Oculus requires OVRPlugin to function [20]. OVRPlugin is a part of the Oculus utilities asset that can be acquired from the Oculus developer's website or the Unity asset store. The asset contains the resources and scripts required for the Oculus to function. Unity has a Prefab asset type that allows you to store a GameObject object complete with components and properties. What is important for a developer like me to get started are the Prefabs provided by the Oculus utilities asset.

The most important selection here is the OVRCameraRig. An instance of one of these is necessary in every Scene. Scenes contain the environment of an individual "level". The OVRCameraRig houses the OVR Camera Rig and OVR Manager scripts. This prefab acts as the user's eyes for the Oculus. The main benefit of using this script is easy access to the OVR Manager script which is a singleton that exposes the Oculus SDK to Unity. The OVR Camera Rig script controls stereo rendering and head tracking.

I currently have four scenes in my application shown in figure 9. The skybox for these environments is completely black to make the individual elements stand out.

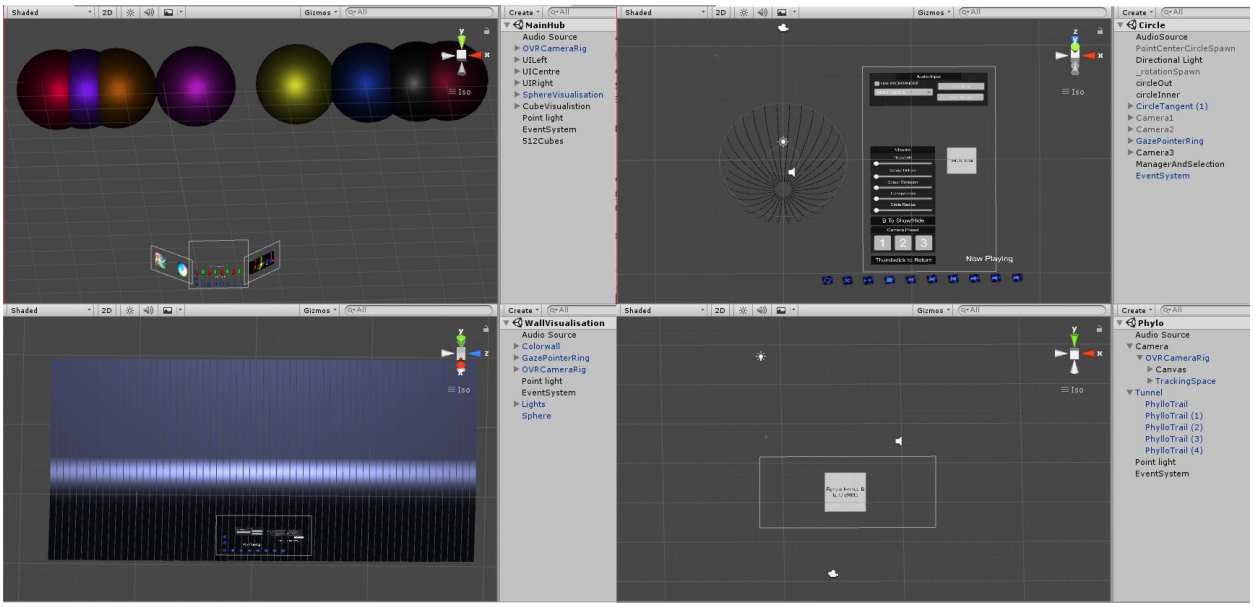



*Figure 9 - Scenes*

The first point of entry for the user is the main hub. This scene provides the user with easy to digest simple visualizations is the form of spheres and cubes. The scene sorrounds the user in a circle made of cubes that are visualized. Eight cubes are visualized by changing positions in front of the user, looking around the user sees 512 cubes in a circle visualized by changing positions and looking up the user sees eight

spheres visualised by deforming. In front of the user is instructions on how to use the interface and a music management system. The visualisation selections are around the user in the form of buttons that transport the user. Additons to visualizations will be added around the user in a circle. This scene acts as a focal point that leads to and from visualisations.

The wall visualisation is 64 bended line that visualise colour based on a gradient and spawn spheres that travel down the line. The user interface consists of sliders that customise the visualization. The user interface contains a music management system and microphone input selection. The ability to add a song with a file browser, load music files from the root directory, return to the main hub is included.

The circle visualisation consists of 64 bended lines around a circle in a cone that is instantiated on load. The visualisation rotates and changes colours based on a gradient. The UI consists of sliders that let you customise the visualisation. A music management system, microphone input, file management and camera positioning is included.

The phylo visualisation consists of a tunnel like visualisation with trail spirals that travel with the user. Currently the UI only includes a home button back to the main hub.

## 4.2 Standard Scene Stutructure With a Summary of Scripts

Unity is a scripting environment whereby everything runs alongside each other. Each scene in my project contains a set of common objects that are included as follows:

Audio source object that plays music and contains the Audio Visualization and Music Manager scripts

OVR camera rig that controls what the user sees attached with the OVR Camera Rig, OVR Manager, OVR Physics Raycaster and an audio listener.

UI Elements attached with control scripts and OVR Raycaster.

Visualizations attached with control scripts.

An event system attached with OVR Input Module.

Scripts are summarised as follows:

AudioVisualization: Main script that handles audio analysis, frequency band creation and microphone input.

CameraSelection: Handles enabling and disabling camera in a scene to change the visualization perspective.

ChangeScenes: Scene manager that facilitates changing visualizations.

ChangeValues, SliderValues: A control script that changes visualization values based on slider values.

CircleTangents, Cube, Cube512, colorwall, Phyllotaxis, PyyloTunnel, RotateThis, LightOnAudio, BlendOnAudio, DeformOnMusic: Manipulation scripts (transforms, lights, physics, etc) that are based on audio. Some include instantiating objects.

MicrophoneSelection: Handles enabling/disabling audio devices with the help of UI elements.

LoadFile: Handles loading files from disk or directory.

HideUnhide, HideShowMenu: Handle user input via controller to hide user interfaces so they can fully immerse in the visualisation

OVR Scripts: Standard Oculus utilities scripts

Important scripts and how they work is explored in the upcoming sections.

### 4.3 AUDIO VISUALISATION SCRIPT

Unity has an update function that is called once per frame. Figure 10 includes the methods called in this script. This section explores each method in update.

```
GetAudioData();
EightFrequencyBands();
MakeFrequencyBands64();
EightFrequencyBandsBuffer();
BandBuffer64();
CreateAudioBands();
CreateAudioBands64();
GetAmplitude();
MicrophoneInput();
```

*Figure 10 - AudioVisualization Update()*

The AudioVisualisation script generates the array of samples used for the visualisations. The vital part of this script is shown in figure 11.

```
 //FFT values
public float[] left = new float[512];
public float[] right = new float[512];

 void GetAudioData()
   {
      audioSource.GetSpectrumData(left, 0, FFTWindow.BlackmanHarris);
      audioSource.GetSpectrumData(right, 1, FFTWindow.BlackmanHarris);
   }
```

*Figure 11 - GetAudioData*

GetSpectrumData is a Unity method that was explored in the research section. An analysis of the left and right channels are stored in their respective float[] arrays. The window function is as follows: W[n] = 0.35875 - (0.48829 * COS(1.0 * n/N)) + (0.14128 * COS(2.0 * n/N)) - (0.01168 * COS(3.0 * n/N)).

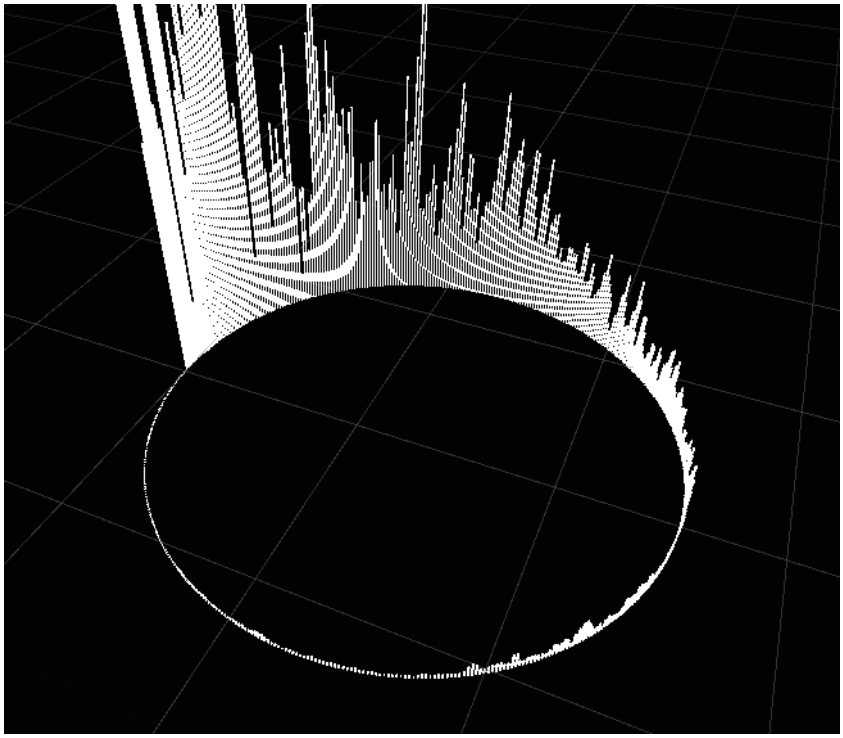


*Figure 12 - 512 Bins Visualised (Logarithmic Amplitude Samples)*

Figure 7 is a visualisation consisting of 512 cubes. The amplitude of the bins visualised is logarithmic. This shows the trend that the values in the higher frequencies are low while the values in the lower frequencies are high. The large amplitude difference is a problem for visualisations as the disparity is uneven. For a better result you can use a logarithmic scale that forces the lower and higher frequencies to have the same maximum amplitude. Using the input of 512 individual samples would be very taxing to deal with, so taking inspiration from the seven frequency bands I have decided to create a geometric sequence of eight bands.

The seven frequency bands are as follows:

Sub bass: 20 to 60Hz
Bass: 60 to 250Hz  everything runs at the same time
Low midrange: 250 to 500Hz
Midrange: 500 to 2000Hz
Upper midrange: 2000 to 4000Hz
Presence: 4000Hz to 6000Hz
Brilliance: 6000Hz to 20000Hz

The spectrum consists of 22050 hertz. Dividing this value by 512 yields 43 hertz per sample. Splitting samples using the sequence 2, 4, 8, 16, 32, 64, 128, 256 + 2 uses the entirety of the 512 samples.

| Frequency Band | Sample Count | Hertz Range |
|---|---|---|
| 0 | 2 | 0 to 86 |
| 1 | 4 | 87 to 258 |
| 2 | 8 | 259 to 602 |
| 3 | 16 | 603 to 1290 |
| 4 | 32 | 1291 to 2666 |
| 5 | 64 | 2667 to 5418 |
| 6 | 128 | 5419 to 10922 |
| 7 | 256 + 2 | 10923 to 22050 |

*Figure 13 - Eight Bands Hertz Range*

The band ranges in figure 13 can be created by finding the average amplitude of all the samples in the frequency band. Figure 9 accomplishes this.

```
int count = 0;
for (int i = 0; i < 8; i++)
{
    float average = 0;
    int sampleCount = (int)Mathf.Pow(2, i) * 2;
    if (i == 7)
    {
        sampleCount += 2;
    }
    for (int j = 0; j < sampleCount; j++)
    {
        if (channel == _channel.Stereo)
        {
            average += (left[count] + right[count]) * (count + 1);
        }
        …..
        count++;
    }
    average /= count;
    freqencyBand[i] = average * 10;
}
```

*Figure 14 - Make Eight Frequency Bands*

The definition of 8 bands translates very well into a visualisation by giving homage to the sub-categories of the spectrum. A stereo visualisation is produced by adding the values from the left and right channels together. The visualisation produced from this point is adeqaute but very jittery. The music values change rapidly from high to low producing a visualisation that is not smooth. This is a serious issue as if you tie in a color spectrum to the visualisation values, it would go from the highest light color to the lowest dark color rapidly causing an uncomfortable experience. To overcome this problem a buffered value needs to be introduced that slowly transitions two values together.

```
for (int g = 0; g < 8; ++g)
{
    if (freqencyBand[g] > bandBuffer[g])
    {
        bandBuffer[g] = freqencyBand[g];
        bufferDecrease[g] = 0.005f;
    }
    if ((freqencyBand[g] < bandBuffer[g]) && (freqencyBand[g] > 0))
    {
        bandBuffer[g] -= bufferDecrease[g];
        bufferDecrease[g] *= 1.2f;
    }
}
```

*Figure 15 - Eight Frequency Bands Buffer*

Figure 15 creates a buffer for the freqency band values. If the value in question is lower than the previous value (buffered), the value will be lowered at a steady pace to the current value. If the value is higher than the previous value, the buffered value becomes the initial value. A buffer is essential in an artistic capability. Allowing for a smooth transition in rotation, positional, scale, etc values is an additional tool.

```
for (int i = 0; i < 8; i++)
{
    if (freqencyBand[i] > freqencyBandHighest[i])
    {
        freqencyBandHighest[i] = freqencyBand[i];
    }
    audioBand[i] = Mathf.Clamp((freqencyBand[i] / freqencyBandHighest[i]), 0, 1);
    audioBandBuffer[i] = Mathf.Clamp((bandBuffer[i] / freqencyBandHighest[i]), 0, 1);
}
```

*Figure 16 – 0 to 1 Audio Bands*

To easily use these spectrum values in everything that a visualisation should offer in Unity, you can create a range of values that scale from 0 to 1, where 1 is the highest

amplitude of the current audio data. A value of 0 to 1 allows ease of use with transforms, lights, shader values, blend shapes, physics, animations, etc. To get this value you should divide the current value with the highest value that has been played, as shown in figure 16, and store it separately.

A problem that arrises at this point is the beginning of a visualisation. There is severe instability in the first few seconds of a visualisation as the values trip up because there is not a profile of highest audio values. A few seconds later the visualisation smooths out but, depending on the audio data, some bands will have very small values causing an inaccurate visualisation.This issue can be fixed by assigning a predefined highest frequency band that is assigned to all bands as shown in figure 17.

```
void AudioProfile(float audioProfile)
{
    for (int i = 0; i < 8; i++)
    {
        freqencyBandHighest[i] = audioProfile;
    }
}
```

*Figure 17 - Set Highest Frequency Band*

Another tool that can be used in a visualisation is averge amplitude. This can be achieved by adding all of the bands together and dividing by the number of bands. The last tool to mention is microphone input.

Microphone input is achieved by recording the audio that is being played through the device and then played through the audio source. This creates an echo as the input from the microphone device is being served to the audio source which is something that is not desired. To combat this issue you can use an audio mixer that mutes the output of the microphone but still feeds values to the audio source to be visualised. An operating system may have many different devices that produce audio so it is good practice to allow the user to pick a device. Stereo Mixer is a device that is included in Windows 10 that parses all audio that is being played on the host desktop. Choosing this device to record from widens the possibility of audio that can be visualised. For example, playing a spotify playlist in a broswer or a youtube music video. A portion of the code is included in figure 18.

```
    if (useMic)
        {
            count++;
            MusicManager.Stop();
            audioSource.outputAudioMixerGroup = mixerGroupMic;
            if (Microphone.IsRecording(selectedDevice))
            {
                ramFlushTimer += Time.fixedDeltaTime;
                RamFlush();
            }

            if (!Microphone.IsRecording(selectedDevice))
            {
                audioSource.clip = Microphone.Start(selectedDevice, true, 10,44100);//Starts
recording
                while (!(Microphone.GetPosition(selectedDevice) > 0))
                {
                } // Wait until the recording has started
                audioSource.Play(); // Play the audio source!
            }
        }
```

*Figure 18 - Microphone Input*

Increasing the ammount of bands from 8 to 64 increases the range of values to play with for a visualisation. The division of samples is shown in figure 19.

| Freqency Band | Samples Each Band | Total Number of Samples |
|---|---|---|
| 0-15 | 1 | 16 |
| 16-31 | 2 | 32 |
| 32-39 | 4 | 32 |
| 40-47 | 6 | 48 |
| 48-55 | 16 | 128 |
| 56-63 | 32 | 256 |

*Figure 19 - 64 Bands Samples*

Using these tools you can create a multitude of visualisations only limited by your imagination and 3-dimensional objects.

## 4.4 VISUALISATIONS

At this stage visualisations are only limited by the visual objects. A lot of effort needs to be placed into designing and creating objects to manipulate. Since 3D modelling is not the focus of this report I have decided to primarily use Unity 3D primitives to give meaning to the scenes. Meaning can be derived from the following classifications.

Transform includes a change in rotation, position or scale. A position transform in the x, y or z axis brings the scene alive by moving at a certain beat. A rotation transform shows a new perspective to the object. A scaling transform brings an object into view when compared to a lower scaled object.

Colours introduce a wide variety of options for a visualisation. Changing saturation, contrast, hue, etc gives the scene definition. Within Unity colour changes are made possible with a shader that acts as a material. A change in a colours brightness sets a mood. A lerp between different colours based on audio provides a new spectrum to view from.

Physics controlled by audio produces outcomes based on the applying force. An example includes pushing a sphere towards a viewer at a speed that is determined by a specific audio beat.

Using symmetry in a visualisation completely engrosses a viewer. The symmetrical pattern is easily identifiable with focal points for viewers to concentrate on. Symmetry can make complex visualisations look simple and easy to digest for a viewer.

### 4.4.1 Eight Cubes

A simple visualisation is the eight cubes in figure 20 that corresponds to a single band of the eight frequency bands from 0 to 7.

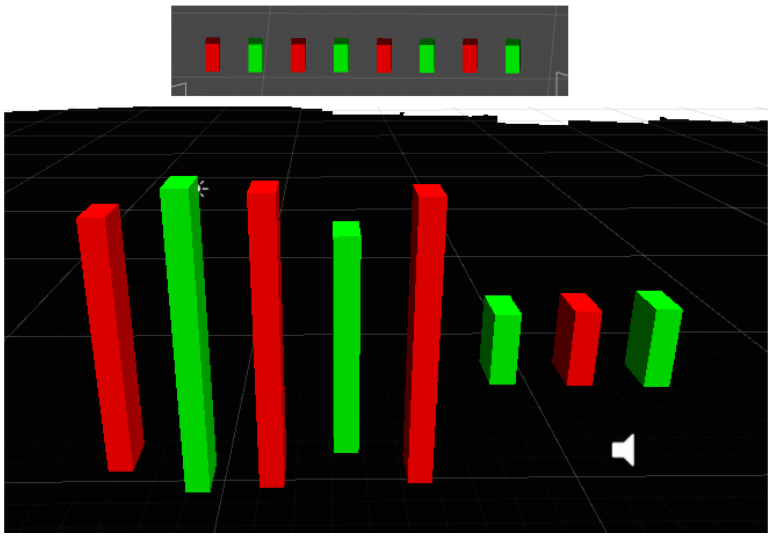


*Figure 20 - 8 Cubes*

They are transformed with the code in figure 21.

```
this.transform.localScale = new Vector3(transform.localScale.x,
(AudioVisualization.audioBandBuffer[category] * scaleMultiplier) + startScale,
transform.localScale.z);
```

*Figure 21 - 8 Cubes Code*

### 4.4.2 Eight Spheres

Figure 22 includes a set of spheres that correspond to a single band of the eight freqency bands.

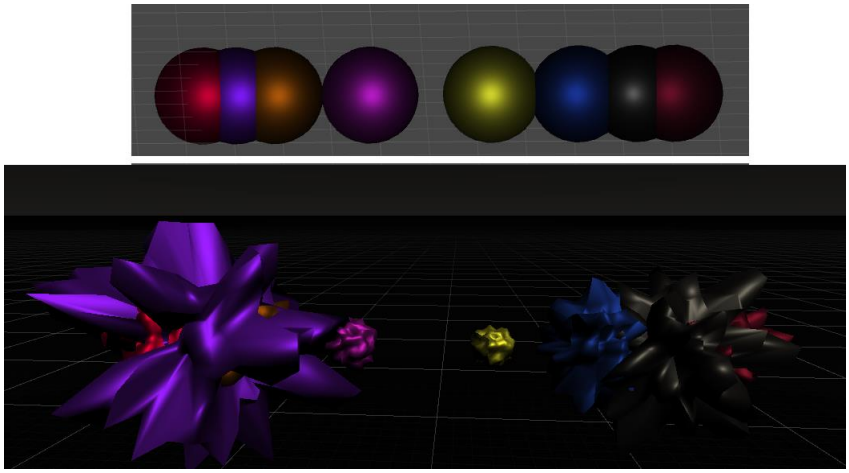


*Figure 22 - 8 Spheres*

The spheres deform according to the beat. This is achieved through a mesh deformation plugin [16] that moves the vertices of the triangles to an offset based on noise.

```
noiseVelocity = new Vector3 (AudioVisualization.audioBandBuffer [audioBand] *
noiseVelocityOnMusicMultiplier.x, AudioVisualization.audioBandBuffer [audioBand] *
noiseVelocityOnMusicMultiplier.y, AudioVisualization.audioBandBuffer [audioBand] *
noiseVelocityOnMusicMultiplier.z);

transform.localScale = new Vector3 (startScale + ((AudioVisualization.audioBandBuffer
[audioBand] * musicScaleMultiplier) * ScaleOnMusicMultiplier.x), startScale +
((AudioVisualization.audioBandBuffer [audioBand] * musicScaleMultiplier) *
ScaleOnMusicMultiplier.y), startScale + ((AudioVisualization.audioBandBuffer [audioBand] *
musicScaleMultiplier) * ScaleOnMusicMultiplier.z));

deformAmount = AudioVisualization.audioBandBuffer[audioBand] * deformMultiplier;
```

*Figure 23 - 8 Spheres Transform*

The transform code included in figure 23 calculates the noise veolicty to create the deform effect. After calculation is concluded the mesh is rebuilt with the parameters of the value. The code also scales the sphere in size depending on audio.

### 4.4.3 512 Cubes

Figure 24 shows 512 cubes instantiated on load corresponding to a single value of the 512 float[] array.

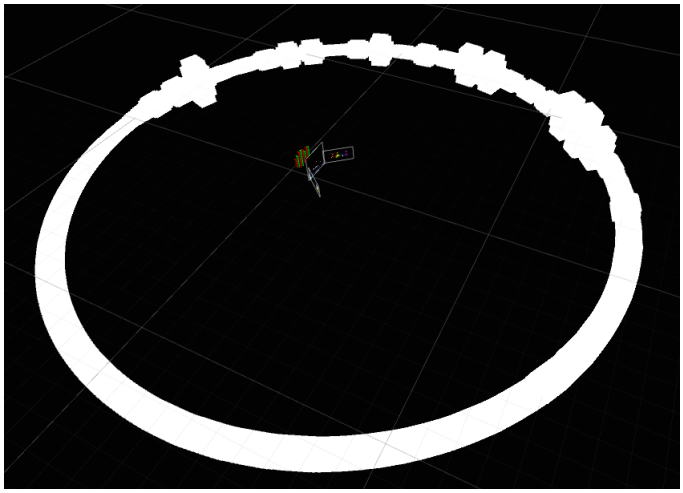


*Figure 24 - 512 Cubes*

The instantiating code shown in figure 25 saves time by negating the need to add in a single cube at a time.

```
for (int i = 0; i < 512; i++)
      {
          GameObject instanceSampleCube =
Instantiate(sampleCubePrefab);
          instanceSampleCube.transform.position = this.transform.position;
          instanceSampleCube.transform.parent = this.transform;
          instanceSampleCube.name = "SampleCube" + i;
          this.transform.eulerAngles = new Vector3(0, -0.70312f * i, 0);
          instanceSampleCube.transform.position = Vector3.forward * 100;
          sampleCube[i] = instanceSampleCube;
      }
```

*Figure 25 - 512 Cubes Instantiate*

The visualisation transforms the same way as the 8 cubes but has the added benefit of surrounding the user.

### 4.4.2 A Wall of Colour

Figure 26 shows a collection of 64 bended lines that try to form the structure of a wall.

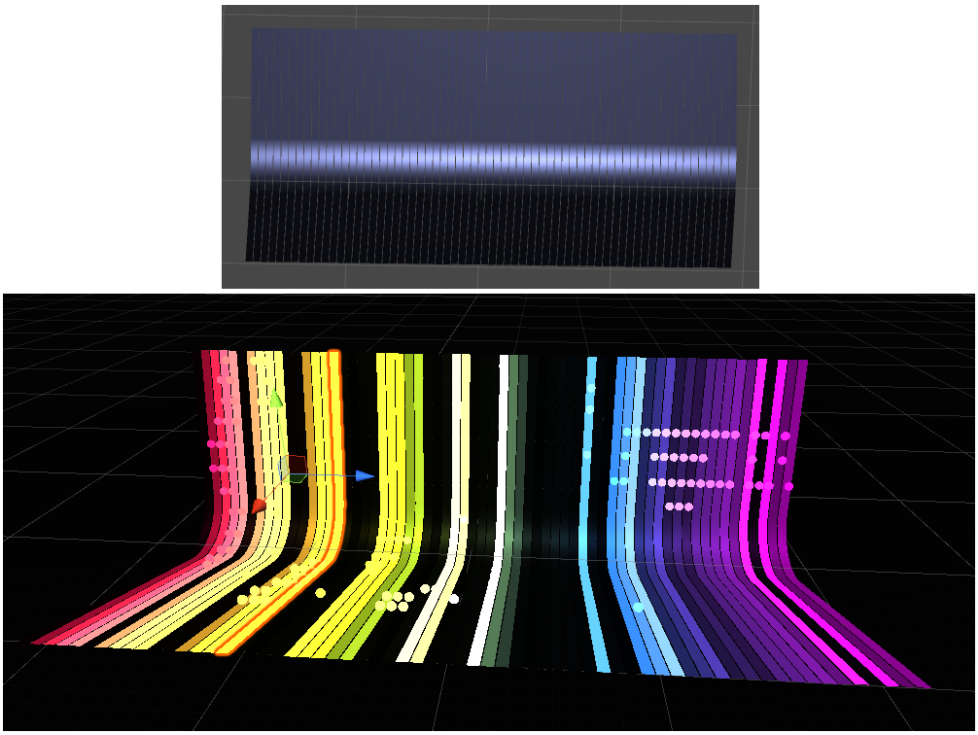
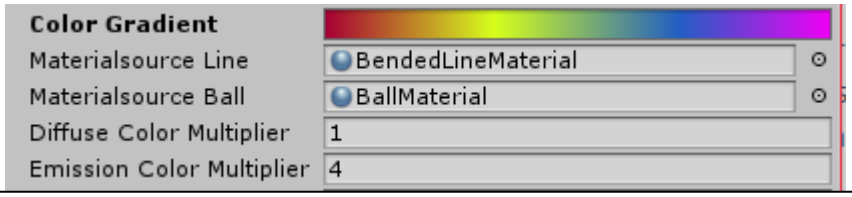


*Figure 26 - Wall of Colour*

This visualisation uses 64 freqency bands. The colour is determined by a colour gradient calculated by audio shown in figure 27.




```
Color DiffuseColor = new Color ((_color [i].r * audioVisualization.audioBandBuffer64 [i]) * _diffuseColorMultiplier, (_color [i].g * audioVisualization.audioBandBuffer64 [i]) * _diffuseColorMultiplier, (_color [i].b * audioVisualization.audioBandBuffer64 [i]) * _diffuseColorMultiplier, 1); // the diffuse color is always updating, to the bandbuffers, and multiplied by the amount specified on the slider
_materialLine [i].SetColor ("_Color", DiffuseColor); //apply the color above
```

*Figure 27 - Colour Gradient*

```
if ((audioVisualization.audioBand64 [i] > _thresholdBallSpawn) && (_intervalBall [i] <= 0)) { // for each audioband that is between 0-1 we check if the amplitude is higher than the threshold, and then spawn. The interval makes sure we have control over how often balls spawn
//spawn
for (int g = 0; g < _ballsPool.Count; g++) {
if (!_ballsPool [g].activeInHierarchy) { //checking our object pool for an inactive object
_ballsPool [g].transform.position = new Vector3 (_spawnPosBalls.position.x, _spawnPosBalls.position.y, _lineTransform [i].position.z); // set it's position to the spawnballs transform, and the Z axis of the corresponding line
_ballMeshRenderer [g].material = _materialBall [i]; // set it's material to the correct color
_ballsPool [g].SetActive (true); //set this object to true
_ballRigidbody[g].AddForce (0, -5000, 0); // adding some extra force down, so the ball has some higher magnitude rolling
break; // we want this script to stop running after we instantiated a ball
}
}
_intervalBall [i] = _ballIntervalTime; //reset the interval time at this line
}
```

*Figure 28 - Wall Spheres*

Physics is also included in the form of instantiated spheres. A pool of spheres is created on load and dropped with force down the bended lines as shown in figure 28.

The visualisation can be customised with the settings shown in figure 29 with varying effects.

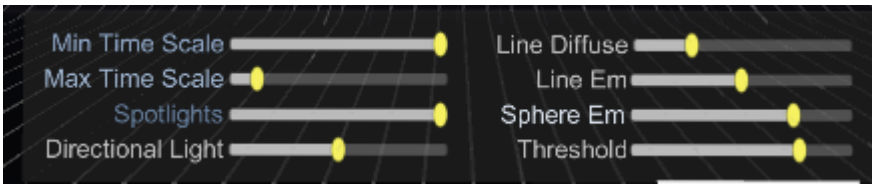



*Figure 29 - Wall of Colour Options*

### 4.4.3 A Virtual Eye

This visualization uses 64 freqency bands. It consists of sphere lines with a dome that instantiate on load and form a circle as shown in figure 30.

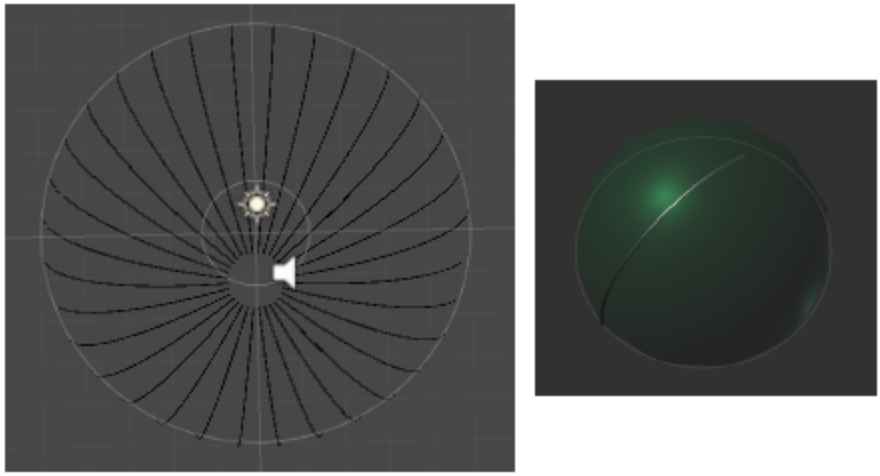


*Figure 30 - Virtual Eye*

It has the same colour visualizations as the previous but it also includes a rotation element, shown in figure 31, attached to a game object.

```
this.transform.Rotate (_rotateAmount.x * AudioVisualization.amplitudeBuffer * Time.deltaTime, _rotateAmount.y * AudioVisualization.amplitudeBuffer * Time.deltaTime, _rotateAmount.z * AudioVisualization.amplitudeBuffer * Time.deltaTime);
```

*Figure 31 - Rotate*

The visualisation can be customised with the options shown in figure 31 with varying effects. Changing the circle radius drastically changes how the visualisation looks as shown in figure 32.

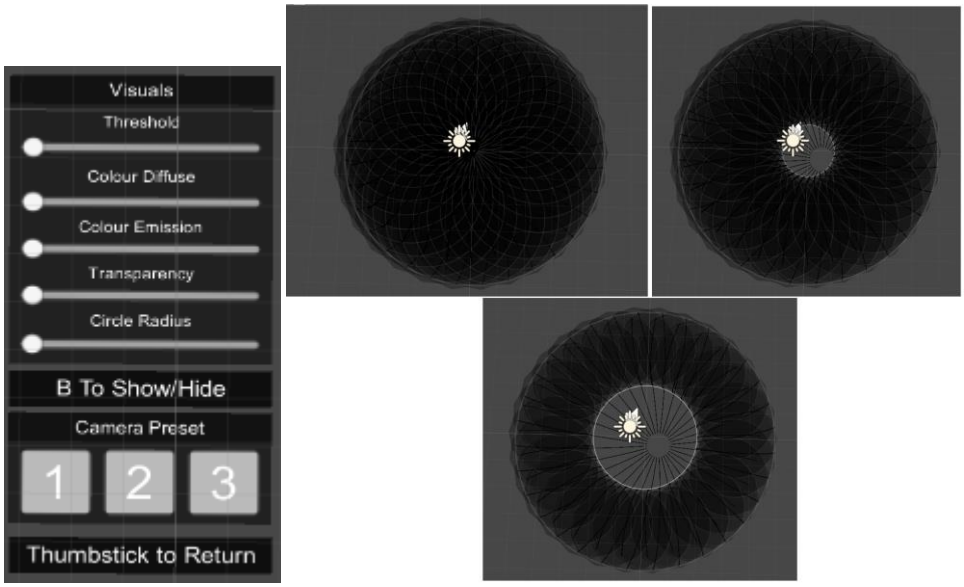



*Figure 32 - Virtual Eye Options*

This visualization has three different camera views that change the perpective of the user. The perpectives are shown in figure 33.

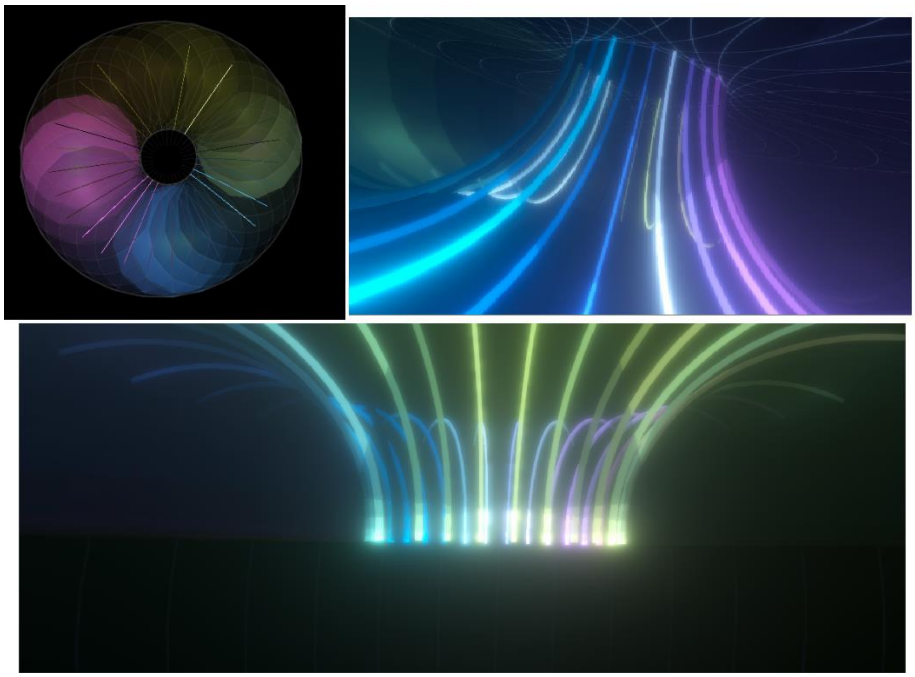


*Figure 33 - Virtual Eye Perspectives*

### 4.4.4 Phyloxtaxis

This vizualisation uses 8 frequency bands with Unity trial renderers to form a 3D spiral-like trail based on audio using Vogel's formula of phylotaxis shown in figure 34. Phyloxtaxis is the arrangement of leaves on a plant stem. Vogel's formula is a planar model of phylotaxis to describe the patterns of florets or seeds of in a sunflower head. [17]

*Figure 34 - Phyllotaxis*

The formula is $\varphi = n * 137.5^{\circ}$, $r = c\sqrt{n}$ where n is the ordering number of a seed counting outwards from the centre, φ is the angle between a reference direction and the position vector of the nth seed in a polar coordinate system originating at the centre, r is the distance between the centre and the centre of the nth seed and c is a constant scaling parameter.

This formula is a polar coordinate system so we need to convert it to a cartesian co-ordinate system to make use of it. X = r * cos(angle), Y = r * sin(angle) accomplishes this with the implementation shown in figure 35.

```
private Vector2 CalculatePhyllotaxis(float degree, float scale, int number)
   {
      double angle = number * (degree * Mathf.Deg2Rad);
      float r = scale * Mathf.Sqrt(number);
      float x = r * (float)System.Math.Cos(angle);
      float y = r * (float)System.Math.Sin(angle);
      Vector2 vec2 = new Vector2(x, y);
      return vec2;
   }
```

*Figure 35 - Phyllotaxis Formula*

The trails scale, lerp position and change colour based on audio. Changing the degree variable greatly changes how the visualization acts. A code snippet is shown in figure 36.

```
switch (audioColorSetting)
      {
         case _audioColorSetting.Band:
            audioColorFloat = audioVisualization.audioBand[colorBand];
            break;
         case _audioColorSetting.BandBuffer:
            audioColorFloat = audioVisualization.audioBandBuffer[colorBand];
            break;
         case _audioColorSetting.Amplitude:
            audioColorFloat = audioVisualization.amplitude;
            break;
         case _audioColorSetting.AmplitudeBuffer:
            audioColorFloat = audioVisualization.amplitudeBuffer;
            break;
         default:
            audioColorFloat = audioVisualization.amplitude;
            break;
      }
```

*Figure 36 - Phyllotaxis Colour Settings*

The trails move forward in 3D so the user needs to follow along. This is accomplished with a movement script shown in figure 37.

```
tunnel.position = new Vector3(tunnel.position.x, tunnel.position.y, tunnel.position.z + (audioVisualisation.amplitudeBuffer * tunnelSpeed));

this.transform.position = new Vector3(this.transform.position.x, this.transform.position.y, tunnel.position.z + cameraDistance);
```

*Figure 37 - Follow Phyllotaxis*

## 4.5 UI

This section explores how I designed a user interface in virtual reality and their components.

Using Unity's user interface module, you can create a virtual reality interface on a canvas. Placing a canvas in the environment and setting it to exist in the world space allows you to create a spatial UI. To interface with created elements I have decided to use gaze input as shown in figure 38.

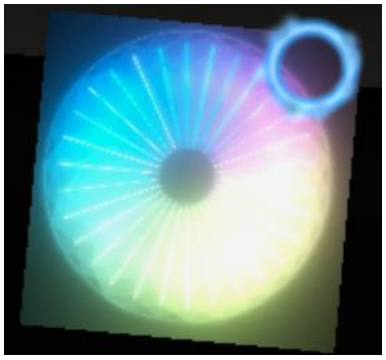


*Figure 38 - Gaze Input*

This works by activating a gaze circle, which serves functionality similar to a mouse, when the user looks at the UI element. Pressing the A button on the controller activates the element. This is accomplished by OVR Raycaster, OVR Physics Raycaster, OVR Gaze Pointer and the OVR Input Module scripts working together, acquired from the Oculus utilities package.

The UI is designed to be easily noticable upon load and at a comfortable distance from the user. The UI for the mainhub, wall of colour, and eye scenes is shown in figure 39.

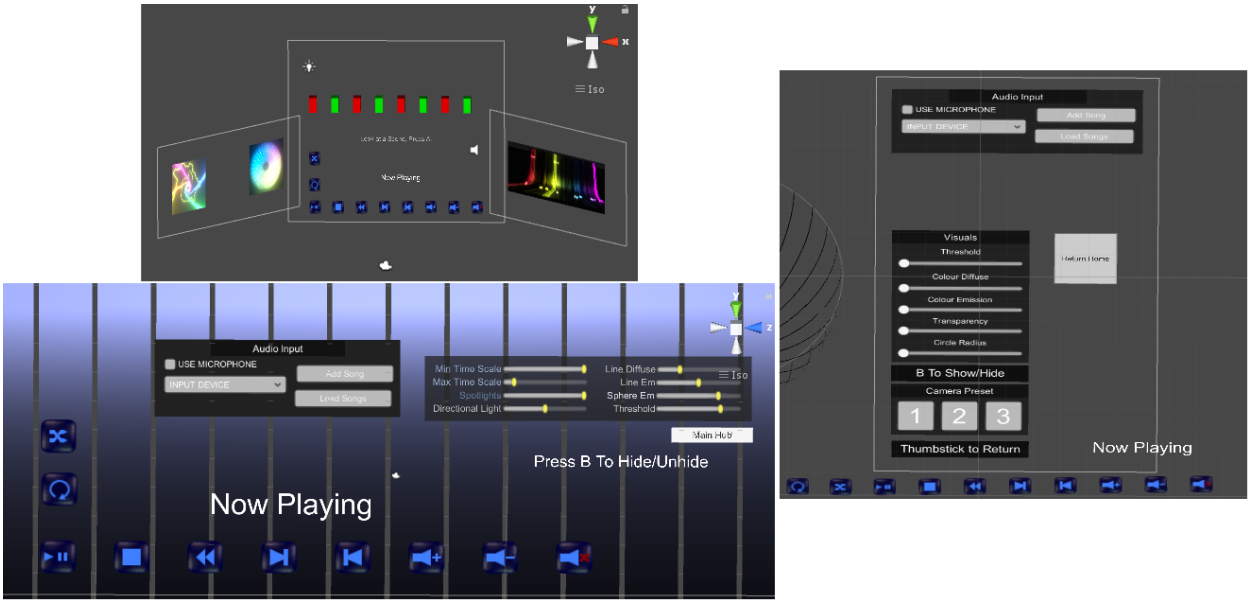



*Figure 39 - User Interfaces*

Features include customisable options tailored to the visualization, a music manager (adapted from a Unity asset [18]), add a song (adated from a file browser Unity asset [19]), load songs, microphone device selection dropdown, instructions to hide the UI, the path of the current song being played with time as shown in figure 40 and a return button to the main hub.


*Figure 40 - Music Label*

The music manager handles track control with standard music player functionality such as play, pause, next, previous, shuffle, repeat, etc.


*Figure 41 - Audio Input*

Figure 40 contains a checkbox when activated prompts the user to select a device from the dropdown menu. Add a song prompts a file manager, shown in figure 42, to display where the user can select a supported (.ogg, .wav) file format.

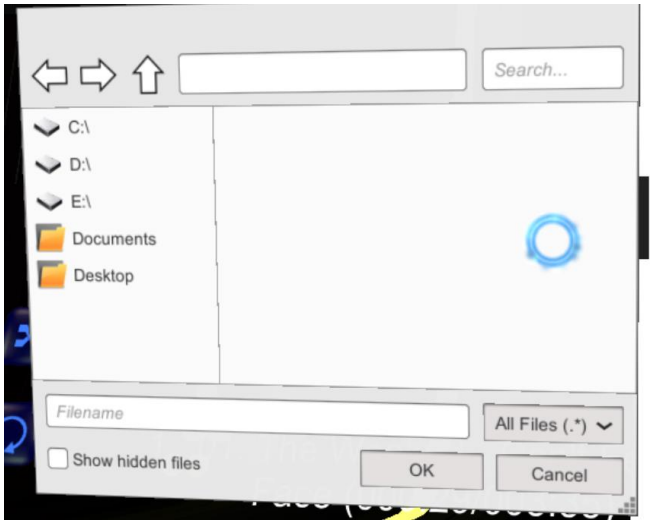



*Figure 42 - File Manager*

Load Songs removes all the current songs in the playlist and loads all the supported music files from the root directory of where the program is installed. The Unity WWW class is used to accomplish this as shown in figure 43. Figure 44 is called upon selecting a file with the file browser. Figure 45 is called upon selecting the load songs button in figure 41.

```
IEnumerator LoadSong(string url)
{
    WWW www = new WWW("file:///" + url);
    AudioClip loadClip = www.GetAudioClip(false, false);
    while (!loadClip.isReadyToPlay)
    {
        yield return www;
    }
    loadClip.name = url;
    MusicManager.AddToPlayList(loadClip);
}
```

*Figure 43 - Load Song*

```
string url = FileBrowser.Result;
        if (IsValidFileType(url))
        {
            MusicManager.Stop();
            StartCoroutine(LoadSong(url));
            MusicManager.Previous();
        }
```

*Figure 44 - Add Song*

```
MusicManager.Stop();
    MusicManager.ClearPlayList();
    // get all valid files
    var info = new DirectoryInfo(absolutePath);
    soundFiles = info.GetFiles()
        .Where(f => IsValidFileType(f.Name))
        .ToArray();

    // and load them
    foreach (var s in soundFiles)
        StartCoroutine(LoadSong(s.FullName));
```

*Figure 45 - Load Songs*

The UI components distract from the experience of the visualisation by existing in the world. For this reason, figure 46 includes code that hides the UI upon pressing the B button on the controller.

```
if (OVRInput.GetDown(OVRInput.Button.Two))
    {
        bool stateOptions = _options.activeSelf;
        _options.SetActive(!stateOptions);
    }
```

*Figure 46 - Hide/Unhide*

## 4.6 SUMMARY

In this chapter I went through the key components of my implementation with screenshots and code snippets where appropriate. I explained how my scenes link together and how the audio visualisation of the project functions. I explained the features of the visualisations included in the project. I discussed UI features of the project.

# Chapter 5 TESTING

This section explores the testing cycle of the project after implementation. A testing of functional requirements was conducted with test cases. An online user survey with a demo video was created that resulted in 25 responses was done to determine if the project deserves further work and is heading in the right direction. 5 testers were interviewed and analysed with recorded heart rate while they used the application.

## 5.1 REQUIREMENTS TESTING

This section describes the requirements testing that was done to determine if requirements were met.

| Test Case | Feature | Functional requirement | Performance | Added comments |
|---|---|---|---|---|
| 1 | Visualise Audio | Translate Music into the Virtual Environment | Very Good | Audio Visualization is obvious with transforms to objects in the environment |
| 2 | User Interface with gaze input | User Interface in Virtual Reality, Optimised User Interface | Good | The interface works well but using head tracking to navigate has caused some neck-ache. The UI visuals could be better. UI is not pixelated. |
| 3 | Change Visualisation Parameters with sliders via User Interface | Customisable visualizations | Good | Placement of the options could be better. |
| 4 | Add Song via file manager | Add Own music | Good | Displayed Error messages are not included if the user picks a wrong file. Only allowing .ogg and .wav files as per Unity functionality is limiting. |
| 5 | Load Songs | Add and control music | Very Good | Loading from the root directory works well but the limiting factor is still .ogg and .wav files. |

| | | | | |
|---|---|---|---|---|
| 6 | Different Types of visualizations navigated by a user interface | Variety of Visualizations, Efficient speeds | Good | There is a variety of scenes with different forms of visualisations to satisfy the requirement. |
| 7 | Microphone input | Microphone device Visualization, Efficient Storage | Good | Device management works well with the added benefit of using stereo mixer to let the user visualize playing audio |
| 8 | Music manager components (play, pause, etc) | Add and control music | Good | All features work |
| 9 | Display current track | Optimised User Interface | OK | There is a bug regarding microphone input whereby the timing of the track is displayed incorrectly. |
| 10 | Wall visualisation options | Variety of Visualizations, Customisable visualizations | Good | Working as intended |
| 11 | Eye visualization options | Variety of Visualizations, Customisable visualizations | Good | Working as intended |

## 5.2 User Acceptance Testing

I conducted a survey asking the question, after seeing the demo, would you want to use the VR music visualizer regularly? The possible responses are as follows:

1. no
2. no but it looks interesting
3. no but I dedicate time listening to music
4. yes, I dedicate time listening to music
5. yes, I don’t dedicate time listening to music
6. I would only use this with drugs
7. Yes, but not regularly

I received 25 responses visualized in figure 47.

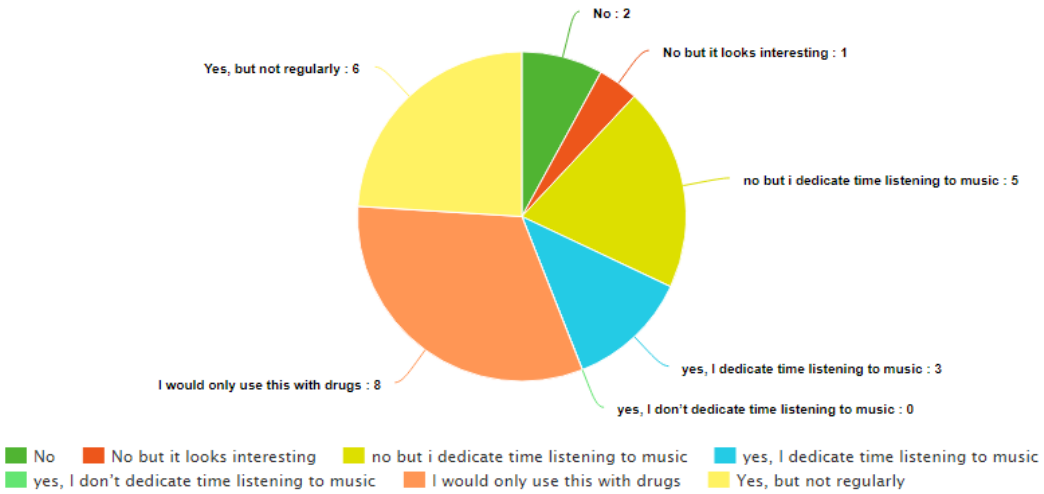



*Figure 47 - User Survey Pie Chart*

The data shows a majority selection in option 6. A greater trend in this selection shows an aversion for users to use the virtual reality application as a psychedelic experience. Visualisations encouraging drug use is not a healthy sign but the data shows popularity for this application is going to be based on this type of user. Making visualisations tailored to increase the psychedelic experience would be the way forward.

The number of users answering no is 8 to 17 saying yes. Disregarding option 6 the amount of yes responses falls to 9. The data shows different users have different use cases. The data shows dedicating time to music is not an accurate gauge to determine if the user is interested in the application. The majority of "no" selections where users that dedicate time to listening to music which could indicate that the application is not interesting at all.

## 5.2 User Application Testing

I gave 5 testers the option to choose a song, showed them a visualization and recorded their heart rate at 1-minute intervals for 3 minutes after the start. The songs chosen by testers in order were: Adele - Chasing Pavements, Coldplay – Spies, Daft Punk - Harder, Better, Faster, Stronger, A$AP Rocky - Long Live a$AP and Kanye West – Famous. Figure 48 shows the value acquired.


*Figure 48 - Heart Rate Info*

The data shows that, in general, the testers heart rate decreased after the first minute. Tester 3 shows a steady increase that goes against the norm of fluctuating rates shown by tester 1, 2 and 3. Tester 5 showed a steady decrease. In general, an upwards trend was seen at the second minute. Third minute values vary and do not show a consistent trend. The data shows a trend of the user either fluctuating, increasing or decreasing their heart rate with only one user staying close to their base heart rate. This can be attributed to the varying music choices of the user but it shows consistently that the application does affect the user.

Testers were shown the features of the application and asked the following questions:

1. Did you experience any moments of bad performance?
2. Did you feel motion sick?
3. Did you enjoy it?
4. Did it look good?
5. What did you like most?
6. Would you recommend the visualizer to others?
7. Would you like to use the visualizer again?
8. How much you would pay for the visualizer?

A surface level analysis of the responses shows that the overall response of questions 1 and 2 is no with testers having no complaints regarding motion sickness or bad performance. Questions 3 and 4's overall response was yes with testers commenting that they had not used anything like it before. A notable response of question 5 was testers enjoying the Virtual Reality experience. Responses to questions 6 and 7 were positive. The response to question 8 was varied from £5 to £20.

## 5.3 SUMMARY

This chapter showed the testing measures completed for this project. The measures were analysed and data points was introduced.

# Chapter 6 CONCLUSION

This section explores what I have learnt and achieved, the challenges I have faced with positives and negatives. It discusses further work that can be done to improve the quality of the project. It provides my own critical analysis of the application.

## 6.1 What I learned and achieved

I learned to use the Unity3D environment to create a virtual reality music visualizer. I mastered C Sharp to provide functionality for my application. I educated myself in the basics behind audio, audio analysis and optimised audio analysis features with native Unity functionality for my application. I taught myself how to use oculus (OVR) functionality to create a virtual reality application. I evaluated best practices for virtual reality user interfaces to provide my user with an experience that does not cause motion sickness or sever eye-strain. I learned to manipulate the different 3D models in Unity with meshes and materials to combine and create an audio visualization with audio analysis data. I learned how to customise the Unity editor to suit my needs. I achieved including a variety of different visualisation with different features in virtual reality. I designed a UI and implemented it into virtual reality with a variety of functionality. I implemented file management features for the application.

## 6.2 Challenges that I Faced

When I started the project, I had zero experience with Unity and C sharp. I had minimal experience with a 3D environment and 3D models. I had no experience whatsoever in developing for virtual reality. I had to familiarize myself with the environment. It was tricky designing a virtual reality space and it took a lot of time and effort. Understanding virtual reality components and 3D object manipulation was a challenge. Making visualisations look good and function properly took longer than initially theorised. Understanding and implementing a UI in virtual reality had to be implemented efficiently.

## 6.4 What would I add or do differently

There are not many things I would do differently. I think the way I approached the project in Unity was a valid implementation path. There are many things that can be added to improve the project. Improving the visuals is a big aspect that can be added. Adding environments to visualisations instead of a black skybox. Adding more visualisation options. Improving the visuals of the user interface. Mapping music management controls to the entirety of the oculus controller. Adding a singleton music manager that persists throughout every scene. Researching and implementing different algorithms to use as visualizations. Adding a story-like experience that visualizes and entire environment that he user can walk through and is not limited to a seated experience. Add in more visualization customisation options. Add in a user profiles that save visualisation options. Add in multiple user support that lets multiple users experience the same environment with each other in virtual space.

## 6.5 CRITICAL ANALYSIS

The application achieves the main functional requirements but there is room for improvement in the secondary requirements. The current form of the application is in its first iteration. Small bugs still exist that need to be fixed. The visuals used could be better. The visualisation included look good but they need an environment around them to really be appreciated. The UI needs to be tested by users to iron out necessary features. The testing undertaken has a very small sample size and needs to be improved.

I believe what I set out to do was achieved but to make what I imagined at the beginning of the project to life I need to put a lot of work into 3D modelling and environment design.

## 6.6 SUMMARY

This chapter reflected my own experiences and what I have learnt from undertaking this project. It summarises my own opinion on what I have created and any further work that could be undertaken is I choose to continue. Virtual reality is a platform that needs to be improved to allow creators to create immersive experiences.

# Chapter 7 REFERENCES

[1] SHERRI L. SMITH & MICHAEL ANDRONICO (Mar 28, 2016) ***What is the Oculus Rift?*** Available at: https://www.tomsguide.com/us/what-is-oculus-rift,news-18026.html

[2] Ian Zamojc (17 May 2012) ***Introduction to Unity3D*** Available at: https://code.tutsplus.com/tutorials/introduction-to-unity3d--mobile-10752

[3] Merrian-Webster (22 Apr 2018) **audio** Available at: https://www.merriam-webster.com/dictionary/audio

[4] OPENCP **OPENCP** Available at: http://www.cubic.org/player/doc/

[5] National Instruments **Understanding FFTs and Windowing** Available at: http://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf

[6] Teach Me Audio **Audio Spectrum Explained** Available at: https://www.teachmeaudio.com/mixing/techniques/audio-spectrum/

[7] GrooVR **GrooVR** Available at: http://groovr.com/

[8] wearevr **Intone** Available at: https://www.wearvr.com/apps/intone

[9] medium (Nov 8, 2015) **Vergence-accommodation conflict** Available at: https://medium.com/vrinflux-dot-com/vergence-accommodation-conflict-is-a-bitch-here-s-how-to-design-around-it-87dab1a7d9ba

[10] KYLE ORLAND (10/6/2016) **arstechnica** Available at: https://arstechnica.com/gaming/2016/10/oculus-lowers-minimum-rift-specs-using-asynchronous-spacewarp-tech/

[11] Unity **Unity** Available at: https://docs.unity3d.com/ScriptReference/AudioSource.GetSpectrumData.html

[12] Plane9 **Plane9** Available at: https://www.plane9.com/

[13] Unity3D **NewEmptyScene** Available at: https://docs.unity3d.com/uploads/Main/NewEmptyScene.png

[14] Available at: https://en.wikipedia.org/wiki/Window_function

[15] gamasutra User interface design in video games Available at: http://www.gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games.php

[16] Available at: https://github.com/keijiro/Kvant

[17] algorithmicbotany **Phyllotaxis** Available at: http://algorithmicbotany.org/papers/abop/abop-ch4.pdf

[18] assetstore unity **Flexible Music Manager** Available at: https://assetstore.unity.com/packages/audio/music/flexible-music-manager-73866

[19] assetstore unity **Runtime File Browser** Available at: https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006

[20] assetstore unity **Oculus Integration** Available at: https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022