An Investigation into Speaker and Headphone-Based Immersive Audio for VR and Digital Gaming Applications

A thesis submitted in fulfilment of the

requirements for the degree of

MASTER OF SCIENCE

at

RHODES UNIVERSITY

by

KYLE DONALD MARAIS

ORCID ID

0000-0002-3785-0098

February 2022

ABSTRACT

There is a need for audio spatialization in gaming and Virtual Reality applications. If the sounds of a virtual environment correspond to their visual sources, the user will feel more engrossed or immersed in the environment and will more likely suspend their disbelief. Certain software, such as Audio Middleware, works in cooperation with game engine software to facilitate the audio requirements for virtual environment and game developers. This thesis provides a comparison of speaker and headphone-based immersive audio systems in the context of Virtual Reality, with the aim of determining and analyzing the differences between both systems. Accordingly, the thesis describes the implementation of an enhanced speaker-based Audio Middleware sound capability for Unity game engine developers. The capability leverages the functionality of a popular Audio Middleware software suite, FMOD, and a speaker-based spatialization system, ImmerGo. The headphone spatialization system that was contrasted with the bespoke speaker-based system was the Oculus Audio Spatializer. A sample VR application that employs both spatialization systems was developed, and served as the testing framework for the experimentation. Ultimately, the application provides users with 3D sound control in real time. Thus, via the use of the Leap Motion controller and Oculus Rift Development Kit 2 Head-Mounted Display (HMD), users can select an audio source with a gesture, and then subsequently pan the audio around them in three-dimensional space. The findings of the research suggest that the speaker-based system facilitated greater immersion than the headphone-based system, despite the headphone-based system providing more accurate audio localization and a higher 'perceived audio quality'.

ACM COMPUTING CLASSIFICATION SYSTEM CLASSIFICATION

Thesis classification under the ACM Computing Classification System (2012 version valid through 2022)

- **Human centered computing → Virtual reality**
- **Human centered computing → Gestural input**
- **Applied computing → Sound and music computing**
- **Software and its engineering → Virtual Worlds software**
- **Software and its engineering → Middleware**
- Software and its engineering → Coroutines
- Software and its engineering → Interactive games
- **Computer methodologies → Motion Capture**
- **Computing methodologies → Virtual reality**
- *Computing methodologies → Motion Processing*
- **Computer graphics → Image-based rendering**
- *Computer graphics → Collision detection*
- Networks → Network servers
- *Computer systems organization → Embedded and cyber-physical systems*

ACKNOWLEDGEMENTS

This thesis would not have been possible without the unwavering support of my parents, Mike and Sue Marais. Their continued belief and encouragement have been invaluable. They have been incredible over the past years, and I cannot stress how much I love and respect them. I should also mention the incredible insight and support that has been afforded me by my supervisor, Prof. Richard Foss. Despite numerous issues and setbacks, his determination and confidence in my abilities have had and will continue to have a profound impact on the Computer Scientist I am and will become. It has not been clear sailing, but I believe that, through his guidance, this work is an important contribution to the field of immersive audio in VR and will undoubtedly influence my career trajectory. I have already used this skillset in a highly successful immersive experience entitled the Subterranean Imprint Archive (SIA), and I am excited to see what else is in store for me in the future. I would also like to thank my significant other, Rachel Strachan, who without a doubt has been a rock throughout the writing stages of this dissertation. She is an extraordinary person, and her wit, humor, and caring nature have enabled the completion of this project. I am so grateful for her role in helping me, and for her love and companionship. I would also like to thank MiniDSP for funding the project and providing the host of hardware that I required. Finally, and not least, I would like to dedicate this work to my best four-legged friend, who unfortunately passed away some time back. Baldrick, you will forever be in my heart, and I miss you every day.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODE BLOCKS

LIST OF EQUATIONS

ABBREVIATIONS

VR – Virtual Reality

VE—Virtual Environment

AR—Augmented Reality

MR—Mixed Reality

HMD— Head-Mounted Display

DK2—Development Kit 2

6DOF—Six Degrees of Freedom

DSP— Digital Signal Processing

DAC—Digital to Analogue Converter

LMC— Leap Motion Controller

HCI—Human Computer Interaction

OBA— Object Based Audio

CBA— Channel Based Audio

DBAP—Distance Based Amplitude Panning

VBAP—Vector Base Amplitude Panning

HRTF—Head Related Transfer Functions

SDK—Software Development Kit

API—Application Programming Interface

IDE—Integrated Development Environment

RPG—Role-Playing Game

FOV—Field of View

FPS—First Person Shooter

FPS[1]—Frames Per Second

3D—Three-Dimensional

IPC—Interprocess Communication

[1] Context will determine which abbreviation applies.

1 INTRODUCTION

"[T]o be forever looking beyond is to remain blind to what is here."

— Alan W. Watts

Virtual Reality rendered through Head-Mounted Display (HMD) technology has been readily available to consumers since early 2016, as the Oculus Rift Consumer Version One was released on March 28 of that year (Oculus, 2016). Since then, Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) application development has burgeoned, and there is no indication that this development will slow down (Schütze & Irwin-Schütze, 2018). While most proponents of the technology predict a massive influence on the entertainment industry, VR and AR development is also being applied in a host of other domains. For example, it is used for the simulation of environments and scenarios for educational and training purposes (Freina & Ott, 2015). In 2016, the global market value of AR and VR was estimated to be 6.1 billion dollars. In 2017, the global market value increased to 9.4 billion dollars, and estimates indicate that it will grow to around 70 billion USD by 2028 (Statistica, 2018; GrandViewResearch, 2021). Apart from the VR and AR industry, gaming has become the biggest and most lucrative component within the entertainment industry (Zackariasson & Wilson, 2012). Indeed, the recent enthusiasm shown by large technological corporations in the creation of a 'Metaverse' indicates that VR technology is set to advance rapidly over the next few years (Lee, et al., 2021).

In the current context of VR technology, and parallel with the development of VR/AR/MR applications and high-fidelity digital games, more and more emphasis is being placed on realistic audio (chapter two discusses VR technology and its applications). Users and developers have recognized that a truly immersive experience requires immersive audio as much as it requires immersive visuals (Schütze & Irwin-Schütze, 2018). This thesis aims to underline the critical role that audio, in particular, plays in the rendering of such an experience, and details how an immersive audio capability was developed to operate in tandem with ImmerGo, an immersive audio speaker[2] spatialization system (Devonport & Foss, 2018). The capability was developed from within the Unity game engine framework and utilized the FMOD Audio Middleware software and Application Programming Interface (API). Once the capability had been implemented, an associated VR application was created, firstly, to showcase the capability and its specific workflow, and, secondly and more importantly, provide the context needed to determine the critical aims and goals of this thesis (see Aims and Goals of the Project below). In short, the primary goal of the research was to determine whether a speaker or headphone-based approach best applies to an immersive interactive VR application/experience. Therefore, the results of the experiments sought to provide insight into this question, whilst also providing usability ratings of the system, and key points of discussion for future research in this area.

This chapter looks to provide a clear reference point for each of the following chapters, and how they combine to enable the deductions presented in the conclusions of this thesis. Consequently, the first

[2] Throughout this body of work, the term "speaker" will be synonymous with loudspeaker, and vice versa.

section of this chapter will provide a more comprehensive discussion of the primary and secondary goals of the research. The second section will indicate the relevance of this work, by briefly analyzing the core field of research that applies to it. After this examination of relevant literature, the hypotheses of the project will be presented. In the third section of this chapter, an overview of the capability, and how it operates will be provided. Subsequently, the need and usefulness of this capability will be analyzed. The chapter will conclude by providing an overview of each of the subsequent chapters and their objectives.

1.1 AIMS AND GOALS OF THE PROJECT

To better reinforce the underlying reasons why this research was conducted, and to provide more clarity as to the questions it has answered, the following two sections investigate the primary and secondary goals of the research.

1.1.1 The Primary Goal

The primary goal of this project was **to analyze and determine the differences between immersive spatial audio implemented using speakers or headphones in the context of VR applications**. In order to do so, the perceived audio quality and spatial localization accuracy of each implementation was determined and contrasted. The systems were therefore both quantitatively and qualitatively analyzed, and the results from this analysis served to prove the hypotheses of the research which are detailed in section 1.2.2. These hypotheses relate to the degree of immersion rendered by the two implementations and are provided in the context of a VR application that was developed for the purpose of comparison.

1.1.2 The Secondary Goals

The secondary goals of the research enable the realization of the primary goal and are outlined in the following paragraphs.

First, one of the goals was to **develop an immersive audio capability that integrated the Unity game engine and the Object Based ImmerGo spatialization system frameworks**. In addition, this capability needed to adhere to the workflow of the Audio Middleware solution, FMOD. Audio middleware is discussed in chapter four, but essentially this type of software facilitates critical audio functionality that the native Unity audio engine does not enable.

Secondly, an additional goal of the project was to **develop a system that utilizes the implementation of the immersive audio capability for an immersive VR application.** This entailed the development of a VR application with high fidelity audio and visual components. Intuitive interaction is also encompassed in this goal by providing the functionality of hand-tracking and gesture recognition within the testing framework. The Oculus Rift Development Kit 2 and Leap Motion (now Ultraleap) controller and their accompanying SDKs were used to facilitate the development of this immersive experience, entitled "Immersive Planets". The developed application served three purposes:

- To provide a testing framework by which the capability could be quantitatively and qualitatively compared against an existing headphone based HRTF spatial audio function provided by the Oculus Spatializer[3] integrated into the FMOD Audio Middleware solution.

[3] https://developer.oculus.com/documentation/unity/audio-spatializer-features/

- To serve as a use-case for the capability that has been developed.

- And, finally, to provide a workflow for the future development of speaker-based Immersive VR experiences using the immersive audio capability.

The success of the system that was developed to utilize the capability was determined via usability ratings. In addition, questions were posed to the participants involved in qualitatively analyzing the system not only to help identify some of the issues experienced in each spatial audio approach, but also to determine their opinions on which implementation best suits VR application use-cases.

1.2 PROJECT HYPOTHESES

Following the fulfilment of the above goals, two hypotheses were proposed. Before discussing these further, it is appropriate to briefly introduce the core research that informed the hypotheses. This research and the associated concepts are examined more closely throughout the thesis.

1.2.1 Core Concepts and the Domain of the Research

To begin, it is important to recognize the importance of audio in its ability to complement and help render immersive experiences. Indeed, the next chapter provides the framework in which immersion is conceived of in this research and emphasizes the importance of multimodal layers in VR applications. It has been posited, however, that audio plays a critical role in provoking illusory qualities in new age media (Garner, 2017), and the importance of immersive sound in game and VR development cannot be overstated. Certain research also suggests that sound plays a crucial role in enabling users to be affected by whatever it is they are experiencing (Nacke & Grimshaw, 2011). According to Schütze, then, "For the new realities (VR/AR/MR), the quality of the audio is not simply beneficial to your end result, or 'nice to have,' it is absolutely critical" (Schütze & Irwin-Schütze, 2018).

This critical aspect of audio can be attributed to a few reasons, the first of which stems from the fact that audio has a unique ability to represent the entirety of a space, and not just what can be seen. The field of view (FOV) of a user in a VR experience generally only represents 90-120 degrees of what can be seen in a 360-degree environment. It is therefore the responsibility of audio to provide the user with information about that which they cannot see in their virtual environment (Sinclair, 2020; Schütze & Irwin-Schütze, 2018). Moreover, another more scientific reason behind the need for audio in immersive experiences corresponds to how the human auditory system can process numerous frequencies and amplitudes from audio that it is perceiving at any one time. In contrast, the visual implementation in VR experiences encompasses set colors for each bit in the image that is being presented to the user. This therefore indicates that reproduced (virtual) audio can provide a depth to perception that is just not possible with visual reproduction (Schütze & Irwin-Schütze, 2018). This then vindicates the need for research to develop immersive audio solutions that can be implemented in new age media. As this research is located firmly in this category, it can be seen as part of a critical movement to emphasize audio in VR application development.

In the context of audio being considered vital in new age media, it is worth discussing its role in an accompanying field, that of the digital game industry. Indeed, the overlap of audio requirements in digital game development and VR development is essentially the same. Of course, the type of digital game or VR application does play a role in the importance of its audio, and the genre of First-Person

Shooters (FPSs) is the genre that corresponds to VR experiences the most. The following discussion on digital games can be seen to apply to VR applications as well. Sinclair and colleagues provide a well-crafted argument for the role of audio in digital games. They suggest that the purpose of audio in digital games revolves around three factors, namely, to inform, to entertain, and to immerse the player (Sinclair, 2020). Informing a player of the virtual environment they are in not only encompasses enabling spatial awareness of objects in the virtual world, but also enhancing game mechanics and feedback. Entertaining the player denotes the ability of the audio to elicit emotional responses through the soundtrack, the music, and the mixing of the audio. The auditory immersion of a player in a virtual world can be attributed to the success of the first two factors (Sinclair, 2020).

From the above, it should be clear that the accuracy and precision of spatial audio in digital games mirrors the requirements that VR applications have. The majority of digital game audio is implemented for headphones (Garner, 2017). This can be attributed to a few reasons. Firstly, the widespread development and adoption of technology for headphones enables headphone implementations of spatial audio in digital games to reach more consumers. Secondly, due to the first reason, the digital game industry has prioritized immersive audio headphone implementations, which have resulted in highly accurate representations of spatial audio in this industry for headphone-enabled spatial audio (Garner, 2017; Sinclair, 2020). Finally, and perhaps the most important reason, is that the average player does not have the financial or practical means to facilitate the use of large speaker configurations for their entertainment; headphones cost a fraction of the price and require little to no overheads for configuration.

However, if one looks at other immersive media, such as film, the benefits of speaker set-ups are easily identified. The ability to enable multiple viewers to experience the audio of the film at one time is one factor, but it can be argued that speakers provide a fuller experience, both in terms of acoustic quality and realism (Rajguru, et al., 2020). Another less documented issue with headphones in media, is that they can induce claustrophobic and anxious responses in certain users. Literature corroborating this phenomenon is scarce; however, the results of this research indicate that this is an issue that speaker-based systems can help alleviate. Indeed, the primary benefits of immersive speaker-based systems can be attributed to their ability to actualize virtual sound fields (Rajguru, et al., 2020). This then denotes the ability of the system to better model reality and how we perceive audio “space” and virtual sound fields.

Of course, it is important to point out that VR as a medium exists neither as film nor as digital games (see Figure 1), and that the benefits of differing approaches to implementing immersive audio in VR will undoubtedly not be consistent across both film and digital games and media. This therefore further reinforces the value of research such as this, which attempts to analyze and determine what type of spatial audio best fits VR application development. Of course, deductions of this nature should recognize the use-case of audio in the application, and the goals of the application being developed, since VR applications can range from uses within the medical industry to the realization of artistic pursuits. They therefore encompass wholly different use-cases from those associated with film and digital games (see chapter two).

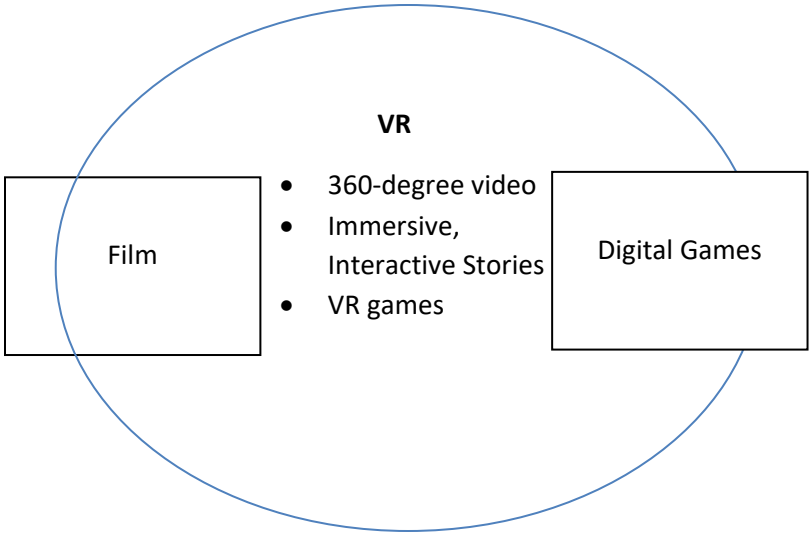

Figure 1: A diagram indicating the domain of VR relative to film and digital games

Certain research has suggested that headphones elicit greater immersion in listeners when compared to speakers for several reasons. Firstly, this conclusion can be attributed to headphones being able to "isolate" the subject from their surrounds, so that the perceived "distance" from the sound source to the listener is reduced (Kallinen & Ravaja, 2007). In this context, "distance" does not necessarily refer to how far the sound is from the listener, but rather the intimacy of the audio, i.e., how the sound source better embodies normal human-human communication (social distance). In addition, the findings of this research suggest that a listener's personality and their individual characteristics play a significant role in their preferred audio system (Kallinen & Ravaja, 2007). In another investigation into whether speakers or headphones better enable perception of auditory messages, it was also found that headphones enabled a higher degree of presence than speakers (Lieberman, et al., 2016).

It is important to note that neither of these instances declared the speaker configuration they employed, and so it can be assumed that the setups they utilized were stereophonic systems, and not 3D/immersive ones. Consequently, this research is useful, as it provides a more contemporary comparison of the current modes in which audio is delivered through both output systems. However, a shortcoming of speaker-based localization is attributed to the propensity of speakers to elicit "crosstalk", and how this phenomenon contributes to substantial audio signal degradation and therefore reduces the localization capabilities of the system. However, mitigating factors are the provision of a "sweet spot" to listeners, which forces listeners to occupy a specific location in the speaker array (Davis, et al., 2005). Indeed, certain rendering algorithms can circumnavigate the requirement of a "sweet spot" and this adds to their desirability in VR application (Lossius, et al., 2009).

To the best of the author's knowledge, there is limited information and research into whether headphones or speaker-based spatial systems better render immersion in the context of VR applications. This then indicates the potential usefulness of such an inquiry. The findings presented in the research above, however, provide a useful framework for the research conducted in this project, and the hypotheses outlined in the following section.

The need for spatial audio in both film and digital games indicates an associated need for spatial audio in VR applications. In addition, the need for spatial audio in VR applies to numerous different applications of the technology. The research conducted in this project attempts to better identify which type of spatial audio (rendered by speakers or headphones) best contributes the VR application developed as part of this project. This application is intended to serve as a template for the different types of VR applications that exist. It is neither a game, nor a film, and can be best described as an immersive interactive experience. Although the findings from this project apply to this specific type of VR application, they can contribute to further research into more specific audio use-cases for VR technology and its applications.

1.2.2 The Hypotheses

Having indicated the fields of research in which this project exists, it is now appropriate to introduce some of the hypotheses. Whilst spatial audio for headphones is desirable and easily accessible to most, the primary hypothesis of this thesis is that **speaker-based spatialization systems facilitate more natural and intuitive experiences when utilized in immersive VR applications**. Whilst the accuracy and precision of headphone-based alternative solutions might be more suitable for certain VR applications (such as fast-paced VR games), it is hypothesized that speaker-based systems are better candidates for narrative driven immersive worlds (stories), as well as VR art exhibitions and film. As indicated previously, a speaker-based VR application has been built as a vehicle to test this hypothesis. The limitations and elements of the implemented system and experimental methodology will be analyzed to negate any bias they might cause.

A secondary hypothesis is that **multimodal VR applications, i.e., applications that enable 3D video, spatial audio, and interactivity, enhance immersion**. In addition, this hypothesis entails determining whether the developed system can be categorized as “highly” immersive. In other words, the usability of the system, together with the visual and audio fidelity of the implementation, needs to be determined to validate whether the system fits this classification.

1.3 THE IMMERSIVE AUDIO CAPABILITY

The capability exists as three components that communicate and function in parallel. Figure 2 provides a schematic representation of the overall system and how the components communicate with one another. The black arrows indicate communication between the components, the numbered squares indicate the three underlying components of the system, and the surrounding circles indicate the frameworks in which the components exist.

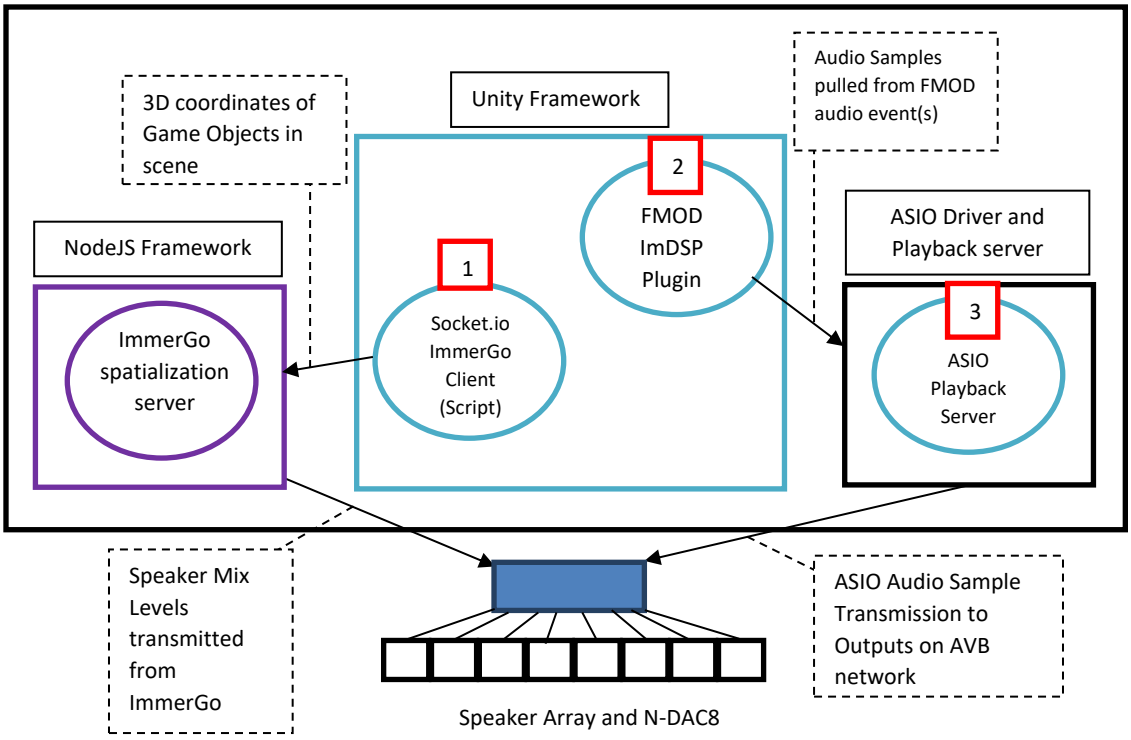

Figure 2: A schematic diagram indicating the three critical components of the capability, and how they operate. The three components are numerically indicated in the diagram.

The first component of the system operates by relaying the three-dimensional (3D) coordinates of game objects, which have an audio source attached, to the ImmerGo server. As mentioned earlier, the system was developed for the Unity Game Engine. Unity is a community driven game engine that has been used to develop many games and VR applications (Unity, 2021), and in this implementation, the personal edition of Unity has been used (version 2017.1.0f3). In Unity, developers may implement their own logic and functions using imported libraries (known as “plugins”) and C# scripts. The first component of the system exists as a script that keeps track of an array of game objects’ positions by updating vector variables (that is, variables suited to holding coordinate values) for each FMOD audio source. In addition, the script contains a function that regularly repeats (polls) and checks the objects’ previous positions. Should the position have changed for any of the audio sources, the script sends socket messages to the ImmerGo server that convey these changes. Moreover, each of the FMOD audio events/sources are attached to game objects within the Unity scene. Therefore, the 3D coordinates of the game object apply to the audio source.

Another important aspect of the first component is the ability to communicate with the ImmerGo spatialization system. The ImmerGo system is a speaker-based audio spatialization system, which currently employs various audio rendering techniques: Distance-Based Amplitude Panning (DBAP), Vector-Based Amplitude Panning (VBAP), and Ambisonics. ImmerGo functions according to the Object-Based Audio (OBA) paradigm. The details pertaining to these rendering techniques and the OBA paradigm, as well as a description of how the ImmerGo spatialization system operates, are provided in chapter three. Typically, the ImmerGo system uses a Digital Audio Workstation (DAW) for its audio sources, and a web-based client for the 3D positioning of these sources. This project required the modification of this system to suit the needs of the implementation. As a result, the first component can

be seen as a “modified” client that performs the functionality of the traditional ImmerGo web client. Figure 3 indicates how this modified ImmerGo Client passes 3D coordinates to the ImmerGo server for rendering, and the production of mix levels. These mix levels are passed on to devices on the AVB network.

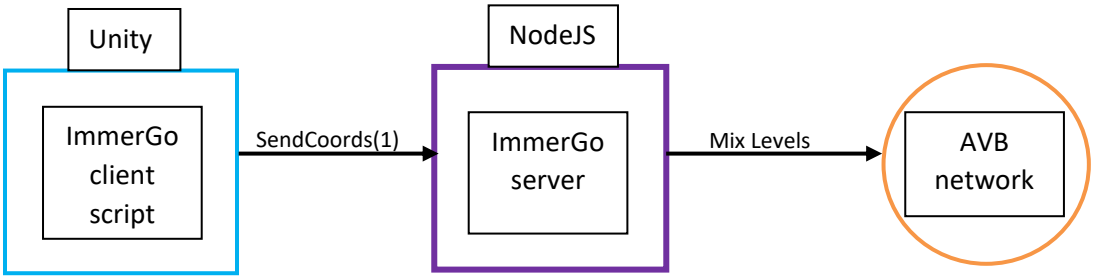

Figure 3: A diagram indicating how the embedded ImmerGo client transmits coordinates to the ImmerGo server that, in turn, calculates the requisite mix levels for speakers on an Ethernet AVB network according to the rendering algorithm being utilized.

The second component of the system is an FMOD DSP plugin, entitled “ImDSP” (see label “2” in Figure 2). FMOD is an Audio Middleware software solution and is described in chapter four. The developed plugin adheres to the requirements of an FMOD Effect plugin and includes socket communication functionality via the Windows Sockets API (Winsock). Developers can bind an instance of the “ImDSP” plugin to their FMOD audio event alongside traditional Digital Signal Processing (DSP) effect plugins (see Figure 4). The plugin functions by populating an audio sample array (with a size specific to the buffer size to which the FMOD DSP is initialized) with samples that are pulled from the audio signal (the FMOD audio event samples). It then transmits these samples across to an audio playback server (which forms the third component of the system) via an IP socket connection. To retain the processing functionality of other FMOD DSP plugins, the “ImDSP” plugin should be placed at the end of the signal processing chain.

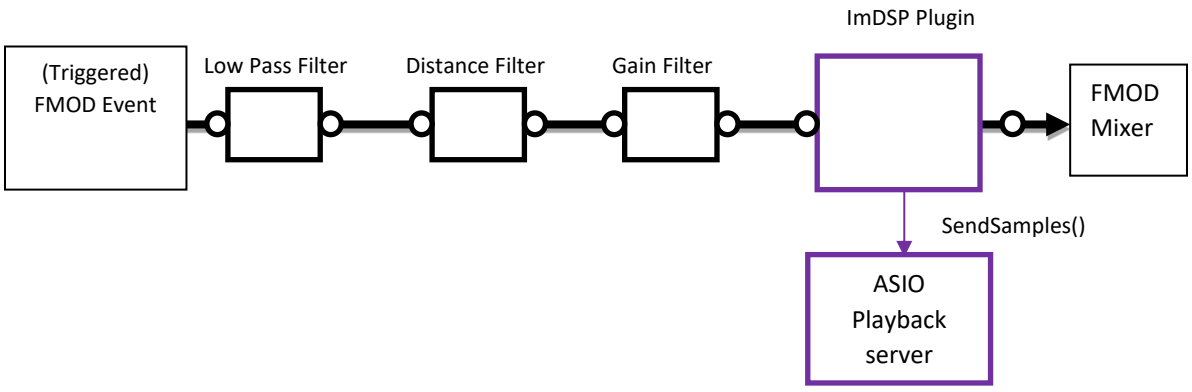

Figure 4: A diagram indicating the ImDSP plugin's location on the DSP effect chain of an FMOD audio event. In this example, the FMOD audio event has a Low Pass Filter, a Distance Filter, and a Gain Filter applied to it before the ImDSP plugin.

The third and final component of the system facilitated the playback of audio samples across an Ethernet AVB network. This ASIO playback server receives audio samples from the “ImDSP” plugins via socket connections (see label “3” in Figure 2), and uses a Winsock implementation of sockets, together with the Audio Streaming Input Output (ASIO) Software Development Kit (SDK) for audio playback. Figure 5 below provides an overview of how the ASIO playback server operates.

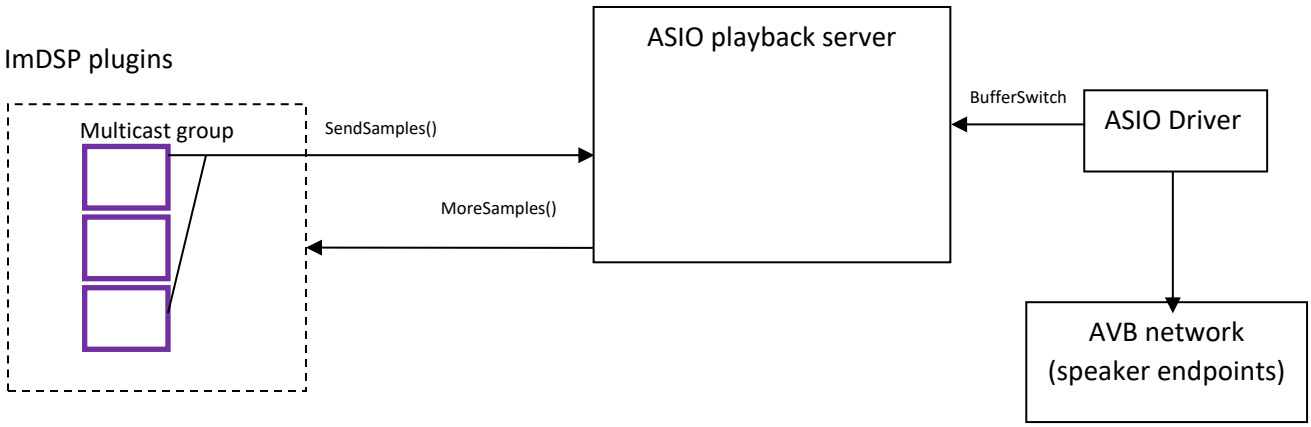

Figure 5: A schematic diagram of the ASIO playback server and how it operates with the ImDSP plugins to receive samples, and then stream them out onto the AVB network.

Chapter five provides a more comprehensive understanding of each of the components that have been introduced and how they function individually, and in tandem, thereby providing an immersive audio capability.

Having described the implemented capability, it would be beneficial to briefly explain the operation of the sample VR application that employs the capability. Essentially, the user is placed at the center of a virtual representation of our solar system. Orbiting around them, are eight of the nine planets. To facilitate interaction, the Leap Motion Controller (LMC) is used to provide input to the system. The LMC is an optical tracking system that renders a digital representation of a user's hands (Guna, et al., 2014), and the SDK that accompanies the controller enables the use of a three-dimensional cursor. Consequently, a user of the developed VR application can select a planet with this three-dimensional cursor and pan it around them in three dimensions. Attached to each of the eight planets, are unique FMOD audio events. Once the experience has been launched, the audio begins to playback on every source, thereby rendering an immersive soundscape. The system incorporates virtual scene navigation, spatialized audio, and free hand interactivity, and thus serves as a viable template for VR applications, and a useful example to test the thesis hypotheses. The particulars of how this application was used in the experiments conducted to ratify the aims of the project are detailed in chapters six and seven.

1.4 The Need For and Usefulness of the Capability

The immersive audio capabilities that the Unity engine provides by default are unsatisfactory when utilizing a large array of speakers in heterogenous speaker configurations. Audio Middleware provides certain means to overcome this issue. However, most middleware solutions require additional third-party support to enable immersive audio within projects. Some of these solutions are detailed in chapter four. This lack of native speaker-based immersive audio support underpins an important reason for this project implementation. In addition, the scalability, modularity, and breadth of rendering algorithms that the ImmerGo spatialization system supports, made it an ideal candidate for the implementation of an immersive audio capability from within the Unity and FMOD development frameworks.

The capability and the system that has been produced provide a pointer to a way in which immersive audio can be utilized for VR applications that wish to utilize the frameworks that have been mentioned in their development. Although certain research suggests that headphone-based audio provides a more

immersive experience, the use-cases of this research need to be analyzed in conjunction with the findings they provide. Moreover, the results of the current research indicate that there are definite benefits to using speaker-based spatialization systems as opposed to headphone alternatives. As such, this capability should be regarded, first and foremost, as a means to implement immersive audio into immersive interactive experiences, therefore lending itself to use-cases in the realm of VR art, film, and performance. Of course, VR games are also supported by the capability; however, it is unlikely that most users will have access to large speaker arrays or personalized immersive audio systems like ImmerGo. Immersive VR experiences that are designed to be exhibited in art galleries and VR film festivals are therefore the most likely candidates to use the immersive audio capability.

When considering the research domain that this project belongs to, there are not many comparisons of headphone and speaker-based solutions to immersive audio within the context of VR. There is a growing need for immersive audio in the new digital media paradigm, and there is plenty of research that pertains to one or the other immersive audio approach, but comparisons of the two are scarce. Indeed, the inception of commercial speaker-based systems like Dolby Atmos and the contemporary interest in Ambisonics indicate that there is a corporate awareness of how speaker-based systems benefit immersion. Ambisonics is detailed in chapter three and is an effective way of reproducing a sound field from a recording. This therefore indicates that the contribution of the research detailed in this project (to its research domain) is an important foray into determining which immersive VR application use-cases benefit most from either type of audio reproduction or spatialization solutions.

1.5 AN INTRODUCTION TO THE FOLLOWING CHAPTERS

Having provided a wholistic view of the project, whilst also supplying context to the research area which it inhabits, this section will now briefly outline the structure of the next seven chapters. The following three chapters provide a context for the critical areas that this project encompasses. To begin, chapter two presents a detailed examination of VR technologies (with a specific emphasis on those relevant to this project), and includes sections on definitions, a historic perspective, the new technological paradigm, and current applications of VR. Subsequently, immersion will be defined in the context of this project, and its importance to VR will also be highlighted, together with the crucial role that audio plays in enabling immersion. The third chapter will dissect audio technologies, with a particular emphasis on 3D and immersive audio. Thus speaker-based rendering algorithms and Head-Related Transfer Functions (HRTFs) for headphone spatial audio will form the core component of this chapter. In addition, Dolby Atmos will be introduced, and the ImmerGo spatialization system will be examined. The fourth chapter will investigate current perspectives on digital game development and VR application development with an emphasis on Audio Middleware solutions. The FMOD Audio Middleware software suite and the Unity game engine will be explicitly studied.

Chapters five and six will provide in-depth reasoning and detailing of the processes and procedures adopted in the implementation of both the capability and the accompanying VR application that have been developed. These chapters aim to clarify how the immersive audio capability operates from within the context of an immersive VR experience (application). Chapter seven will present the methodology and experimental design used to realize the aims and goals of this research, after which the results from these experiments will be presented.

Finally, chapter eight will provide a discussion as to how successful the project has been in realizing its primary and secondary aims, whilst also indicating areas of the implementation that could be improved. This chapter will attempt to vindicate decisions taken both in the implementation and experimental processes. The results and conclusions of the project will then be contextualized according to current research, and modifications of and future work for the project will be proposed.

2 VIRTUAL REALITY TECHNOLOGIES

"To every man is given the key to the gates of heaven. The same key opens the gates of hell.

And so it is with science."

— Richard Feynman

This chapter examines technologies and relevant concepts that relate to the research that has been conducted. Throughout the chapter, the value of immersive audio in these technologies will be emphasized. Initially, the 'New Technological Paradigm' will be examined, and Virtual Reality (VR) will be defined. The second section deals with the history of VR, and this is followed, in section three, by a discussion of Head-Mounted Display (HMD) technology, together with a brief description of the HMD used in this research, the Oculus Rift Development Kit 2 and the system Interaction device, the Leap Motion Controller (LMC). To conclude this section, an analysis of current interaction modalities of the technology will also be provided. The fourth section of this chapter outlines some of the current applications of VR technology, and the need for immersion within the technologies will also be explored and defined. Finally, the chapter concludes with a discussion of the future of VR and some of the ethical issues it raises. The information in this chapter should provide a clear context for VR technologies and their applications.

2.1 THE NEW TECHNOLOGICAL PARADIGM

Virtual Reality technologies of today provide novel and exciting ways in which people and machines interact. The paradigm referred to in the heading of this subsection pertains to how technology has rapidly evolved over the past few decades, and how new and compelling innovations have enabled the contemporary modalities through which we interact with computers, VR being one of these new modalities (Mütterlein, 2018). The research presented in this thesis explores this new paradigm from the perspective of VR technology, and particularly how audio contributes to this technology. VR enables users to explore digital environments, with the ultimate goal that these environments can be perceived as 'real' (Biocca & Delaney, 1995). As such, the visual, auditory, and interactive qualities of a particular environment should enable the user to 'suspend their disbelief'. This suspension of disbelief is a key metric in determining how immersive a virtual reality application truly is. In this context, suspension of disbelief refers to a user's acceptance of a Virtual Environment (VE), as opposed to truly believing the realism of the digital world. And, of course, each of the primary qualities needs to be at a requisite fidelity for total immersion to occur. These concepts are examined in the final section of the chapter (see section 2.4). However, before discussing the complexities and subtleties of VR technology, it would be worthwhile to provide definitions of core ideas and concepts in this field.

Virtual Reality encompasses a variety of definitions, as it is applied in numerous different fields. Additionally, the term has existed for some time now, as seminal work in VR was conducted from the 1960s onwards. In the context of computing, the term 'Virtual Reality' was originally used by Jaron

Lanier in the late 1980s. He was the first person to formally tie the term to “interactive computer-generated three-dimensional immersive displays”, and to produce a fully immersive system that has subsequently become synonymous with VR technologies of today (Schroeder, 1993). As there are many differing opinions on how to define VR, certain attempts have been made to unify said definitions to better categorize and understand how research fits within various frameworks (Kardong-Edgren, et al., 2019). VR needs to be defined according to the technological system (both in terms of hardware and software) that creates the Virtual Reality Environment (VE) that is to be experienced, and may be defined as ‘‘A wide variety of computer-based applications commonly associated with immersive, highly visual, 3D characteristics that allow the participant to look about and navigate within a seemingly real or physical world. It is generally defined based on the type of technology being used, such as head-mounted displays, stereoscopic capability, input devices, and the number of sensory systems stimulated’’ (Lopreiato, et al., 2016).

By the same token, a suitable definition of a VR system is a “a medium composed of interactive computer simulations that sense the participant’s position and actions and replace or augment the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation (a virtual world)” (Sherman & Craig, 2002). There are similarities between both definitions, but it can be argued that the term Virtual Reality also encompasses the software systems being used to render the virtual experience. Another definition for Virtual Reality that is applicable to this specific project, however, is “the sum of the hardware and software systems that seek to perfect an all-inclusive, immersive, sensory illusion of being present in another environment” (Biocca & Delaney, 1995). Before moving on, it is also important to recognize that VR exists on a Reality-Virtuality continuum. If one observes this continuum in Figure 6, VR exists on the opposite side of the continuum from the ‘Real Environment’, and Mixed Reality (MR or XR) is located in-between both VR and the ‘Real Environment’. What is important about this continuum is that it indicates that the concepts of VR and AR sit alongside one another, and that they share common aspects and principles, albeit that they are still discrete fields of technology and research (Milgram, et al., 1995).

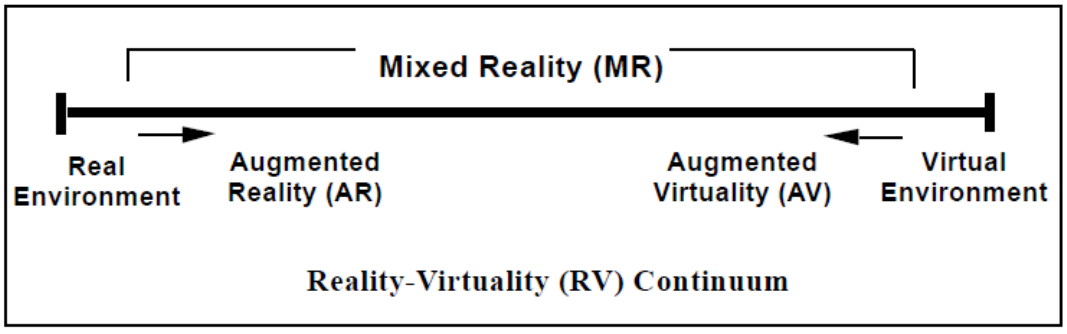

Figure 6: The Reality-Virtuality Continuum (Milgram, et al., 1995).

Having defined VR, it would be beneficial to discuss factors of VR that contribute to the successful experience of a reality. Immersion is indeed one of these factors, but it will be investigated and defined in a later section and can be seen as a sum of the following—perception, telepresence, and interactivity (see Figure 7).

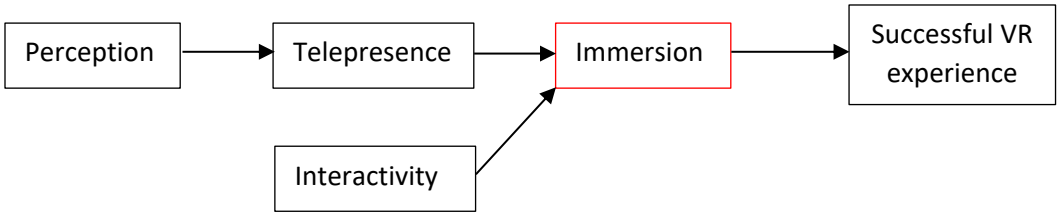

Figure 7: A diagram indicating the factors that influence immersion.

Perception, derived from the Latin "perceptio", which means "gathering or receiving", entails the "organization, identification, and interpretation of sensory information in order to represent and understand the presented information or environment" (Schacter, et al., 2011). This is important for a successful Virtual Environment, as it needs to allow the user to perceive accurately and intuitively whatever it is conveying.

Telepresence can be defined as "A perceptual process where the media user somehow looks past or overlooks the technology to experience the medium". In other words, they can feel 'present' somewhere other than their real, physical location (Steuer, 1992).

Interactivity refers to how a user cooperates with an environment. This cooperation is a very useful way of viewing what interaction encompasses, as the notion of interaction implies scenarios of cause and effect. In other words, a user is provided with a means of performing some or other action, and once performed, this action results in a change or shift (however subtle) in their environment. In more technical terms, the interaction within the environment should be viewed as input to a system – input which is then computed by the system and translated into some or other output. A proposed definition from the perspective of Virtual Reality systems is that interactivity is the ability of a system's "Virtual objects and characters within the virtual world to react to users' actions and interactively communicate with them" (Nalbant & Bostan, 2006).

Before briefly discussing the history of VR and its relevance to contemporary culture, it is worthwhile to cement the conceptual understanding of VR. It is useful to understand *perfect VR* to be a system so immersive that users would be unable to tell whether they are in the virtual or real world. Of course, this situation would only be possible if we address all the human senses we possess. Crucially, the primary senses of vision, sound and touch would need to be activated at a level of unprecedented fidelity. Additionally, the system would have to encompass a depth of interaction that is also foreign right now. This utopian view of VR has been well understood by contemporary culture, as well as by scientists and researchers for some time now. The benefits of a perfect VR system are easily conceivable, as the ability to mirror reality via a virtual experience would entirely disrupt our current ways of being and enable truly novel interactions with technology and ourselves (Gerschütz, et al., 2019). The following chapter will provide general context for VR and its technologies relevant to the research conducted in this body of work. In addition, the proposition that audio and indeed VR sound are almost as crucial to VR as the visual acuity of an environment, will be considered in tandem to the discussions raised throughout the chapter.

2.2 History of Virtual Reality

The history and context of this technology falls into three somewhat discrete eras (see Figure 8). When reviewing the requisite literature, most researchers divide the history of VR into two specific time periods. However, the author of this research argues that the technology is better understood contextually if one reviews its history as three time periods. This argument is hinged on the fact that VR

technology has entered the consumer sphere three times with varying degrees of success (Heim, 2017). The first era encompasses the initial advent of the technology and subsequent strands of research and adoption. The second era denotes the period in which the technology developed into a multi-million-dollar industry and resulted in the second wave of consumer VR systems (Schroeder, 1993). The final and contemporary era is on-going, and constitutes a third wave of VR systems that have resulted not only in a billion-dollar industry but also consumer systems with very real use-cases (as discussed in section 2.4).

This third era of VR once again entering the mainstream and resulting in consumer access to VR technology can be said to have started in 2014. The rapid endorsement and development of consumer VR backed by multinational corporations provides a stark contrast to immersive technology that immediately preceded the third era. In 2012, a variety of immersive devices were heavily critiqued by users and reviewers alike. An example of this was the HMZ-T1 Personal 3D Viewer, which is a 3D display that was developed by Sony and marketed as 'the world's most advanced headset'. This headset was unfortunately far too heavy for most people who tried to use it, and its display capability was criticized for a low field of view resulting in a 'binocular' effect (Harley, 2020). Meanwhile, a young VR and gaming enthusiast by the name of Palmer Luckey was unhappy at the current status quo of the technology in industry and consumer markets and, in April 2012, created the Oculus VR company after having successfully prototyped six iterations of a VR headset. In addition, he initiated a Kickstarter campaign to fund a project he entitled the Oculus Rift.

As the success of the Rift started to gain more momentum, large technological corporations also began to take note. In 2014, Luckey's company, Oculus, was acquired by Facebook for an estimated 2 billion dollars. Later in the same year, the Oculus Rift Developer Kit 2 (DK2) was released and was produced and distributed to VR enthusiasts alongside tech companies and industry contemporaries. The DK2 was an instant success, and until the final consumer version of the Oculus Rift was released in 2016, it remained at the forefront of VR technology. The release of this prototype HMD (Head-Mounted Display) indicated the third era in VR technology had truly begun.

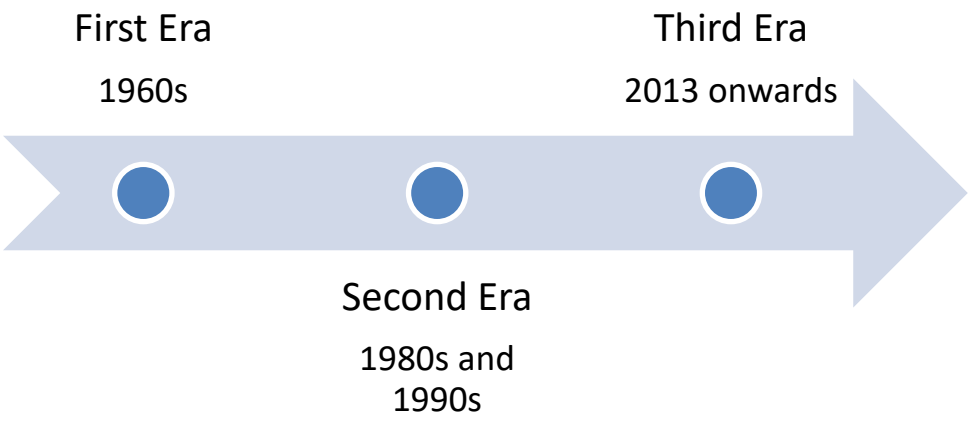

Figure 8: A diagram showing the three eras of VR

2.3 THE ADVENT OF THE HMD

An HMD is a computer display (or screen) that is mounted on a user's head to provide them with a unique perspective into a virtual three-dimensional environment. The first Virtual Reality Head-Mounted Display was pioneered by Ivan Sutherland, whose conceptual understanding of perspective and the use of stereoscopic images to provide a person with a view into a three-dimensional environment was formulated in his 1968 paper (Sutherland, 1968). In this paper, he argues that the use of stereoscopic images (an imaging technique that provides an illusion of depth by taking two photographs of the same subject from two distinct viewpoints whose distance is approximately the average distance between a human's eyes) presented by two screens in front of each eye of a user would enable a three-dimensional perception of the subject being shown.

Stereoscopy is based on the concept of visual parallax, which describes how the positional difference between our eyes causes each eye to perceive a different image of the same scene. Any object that we focus on is in fact perceived from two distinct viewpoints (the left and right eye positions). Our brain is then able to translate these images and perceive depth (Garner, 2017). In HMD technology, if the illusion of depth is to be maintained, Sutherland suggested that the three-dimensional display must exactly update its images according to how the observer moves their head. In other words, the display should exactly model how the perspective of an object would shift in the physical world when a person moves their head. Furthermore, Sutherland suggested that input to the environment in the form of tracking systems would realize an even greater ability of the system to *immerse* users in that environment (Sutherland, 1968). One crucial issue raised at the time of Sutherland's prototypes was the expense entailed in utilizing computing power to render detailed three-dimensional environments and, more importantly, the cost of accruing such a system.

The cost of computing power has decreased in accordance with Moore's law, and the dawn of the information age and subsequent reliance on computing systems across all sectors of society has signaled an age in which computational power is no longer at a premium and exclusionary to many (Shalf, 2020). In addition, the technology that is employed in HMDs has made significant advances. Lightweight mechanical and optical technology is synonymous with the 21st century, as cathode ray tube (CRT) displays have long given way to Light-Emitting Diode (LED) displays.

2.3.1 Head-Mounted Display Technology

The research conducted in this thesis, and the associated implementation of a Virtual Reality front end for an immersive audio system, was created from a modular perspective, and the system that was created for the assessment of the immersive audio capability (see chapter six) was developed using the Unity Game Engine (see chapter three). The HMD that was utilized in the development of the implementation was the Oculus Rift Development Kit 2 (released in 2014). While this HMD has aged significantly in the interim, the Unity application that was created has subsequently been ported to more contemporary HMDs such as the Oculus Quest 2 (released in 2020). It is important to recognize that the core functionality of the DK2 rivals that of more recent HMD released, albeit that this HMD has a significantly lower refresh rate, and a lower resolution display when compared to the Valve Index[4] or latest desktop Oculus HMD. [5]

[4] https://www.valvesoftware.com/en/index
[5] https://www.oculus.com/quest-2/

Traditional two-dimensional (2D) displays provide users with a first-person perspective by presenting them with a window or image into the virtual world. A three-dimensional (3D) display requires stereoscopic imaging across two or more displays to enable a sense of depth in the virtual environment. For a HMD to accurately depict a virtual environment, the stereoscopic displays must be constantly updated to ensure that the illusion of depth is maintained. Additionally, the headset needs to accurately translate a user's head movements, and therefore perspective from the physical world into the virtual one (Peek, et al., 2013). It is therefore important to understand how HMDs function both theoretically and practically. To enable a HMD to accurately interpret a users' head movements in three dimensions, the device supplies rotational data about three axes to a processor (either onboard the HMD or via the system to which the HMD is connected) which performs requisite calculations to align coordinate frames between the physical and virtual worlds (LaValle, et al., 2014). These axes are called the Normal, or yaw axis, the Transverse, or pitch axis, and the Longitudinal, or roll axis (see Figure 9). An HMD can determine this rotational information using an accelerometer, a gyroscope and a magnetometer. All three sensors collect information that is assimilated in a process known as *sensor fusion*. The accelerometer provides the critical rotational information, while the other sensory components are used to adjust for possible drift along any of the axes. This drift can be attributed to the fact that the gyroscope is unable to account for the original position of the headset (Goradia, et al., 2014).

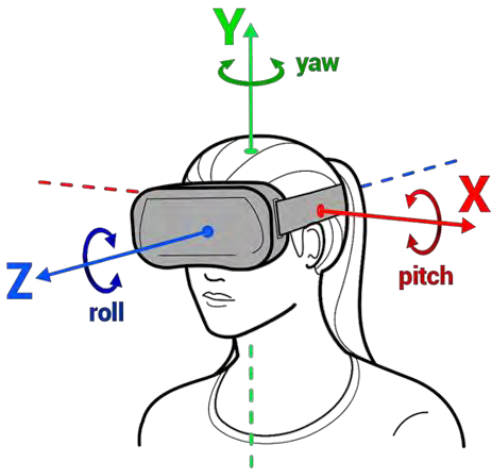

Figure 9: A diagram indicating the three rotational aces that are employed to translate physical head movements into the changes perceived by a user in a virtual environment.

The refresh rate of the displays employed within the HMD will determine how regularly this set of rotational information needs to be assimilated into accurate translational information. Once this information has been translated, the resultant information needs to be interpreted by the processor and applied to images that the stereoscopic displays convey to the user. From a more formal perspective, this interpretation conforms to Euler's Rotation Theorem, which holds that "any 3D orientation can be produced by a single rotation about one particular axis through the origin" (LaValle, et al., 2014). In the contemporary context of VR, a system that utilizes these three rotational axes to accurately adjust a

users' perspective is a system that employs Three Degrees of Freedom (3DOF), which denote the ability of the system to perform orientational tracking. [6]

HMDs expand upon 3DOF principles and, for the most part, are Six Degrees of Freedom (6DOF) systems, which are not only able to perform orientational tracking, but also to enable positional tracking (see Figure 10), and thereby enhance the ability of the system to translate a user's movements accurately and precisely from the physical world into the virtual one (Peterson, n.d.). The aforementioned movements encompass up/down, left/right, and forward/backward changes. To achieve positional tracking, HMDs utilize a variety of sensors and cameras to virtualize the physical space a user inhabits. Should a user move within this space, the information recorded by the sensors will pick up this movement and, by using image processing, determine how the user has moved. This movement can then be translated into the virtual world in such a way that the movement inside the virtual world exactly mirrors that in the real world.

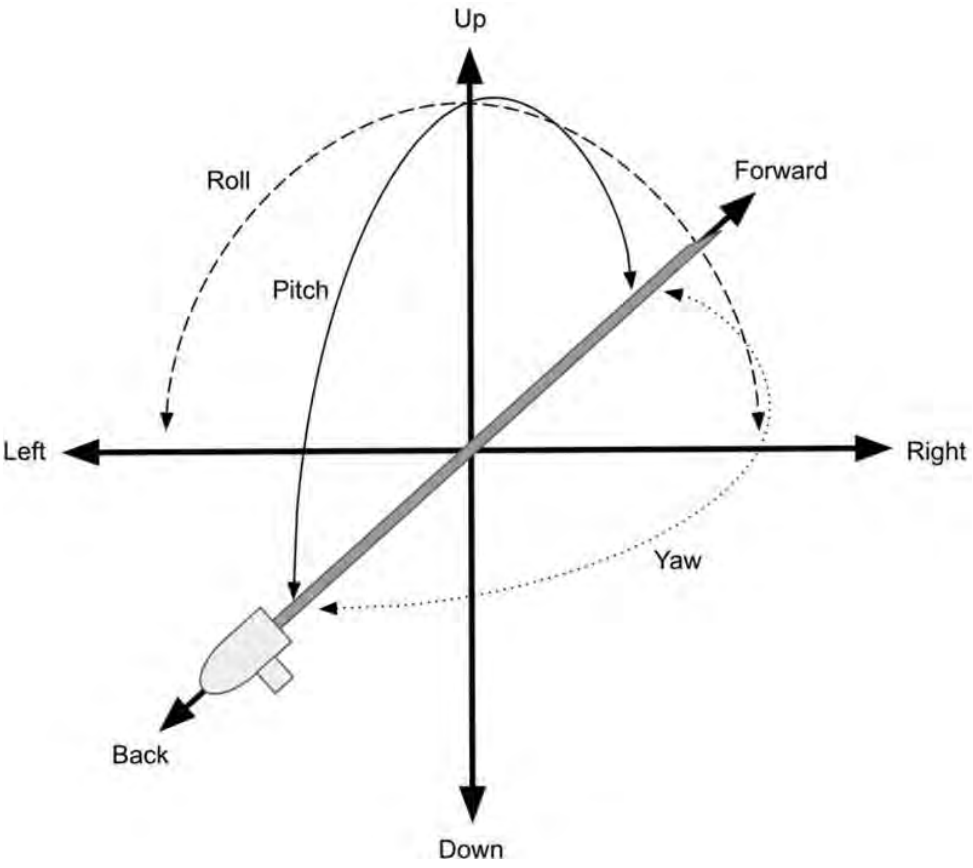

Figure 10: A diagram indicating the six degrees of freedom exhibited by contemporary VR HMDs (Park, et al., 2020).

Having briefly discussed the theoretical conceptualization of HMD technology, and subsequently discussing how head-tracking technology is employed within HMD systems, the final portion of this section will look at the HMD that was used for the research conducted in this thesis.

[6] https://creator.oculus.com/learn/vr-glossary/

2.3.1.1 Oculus Rift Development Kit 2

The Oculus Rift DK2 began being distributed to developers and research spheres from July of 2014.[7] The headset is the prototype that best represents subsequent consumer versions of the Oculus Rift line of HMDs. The fundamental orientational tracking technology of the DK2 remains the same for more contemporary Rift devices. However, to enable 6DOF and accurate positional tracking, an external positional tracker was used to help the DK2 better recognize its surroundings (see Figure 11).

Figure 11: An image of the Oculus Rift Development Kit 2 alongside its external positional tracker8

An aspect of the DK2 and, indeed, all subsequent HMDs that has enabled their success is the ability of the devices to reduce input lag. This reduction in latency can be primarily attributed to the ability of the DK2 to support sensory sampling rates of up to 1000Hz, thereby enabling the HMD to receive, forward, and process the sensory information 1000 times a second (Goradia, et al., 2014). This translates to approximately an interval of 2 milliseconds between a user's head moving and the processor receiving this information (Feng, et al., 2019). A high latency between the sensory information being transmitted for processing results in a greater chance of simulator sickness, as changes in head movements translating into the virtual environment will be delayed and inconsistent, thereby breaking any sense of immersion and negatively affecting a user's vestibular system (Stauffert, et al., 2018). Another aspect affecting user comfort and their perceived immersion would be the refresh rate of the HMD's displays. As with traditional 2D displays, the higher the refresh rate, the more seamless the experience. A high refresh rate generally corresponds with a reduction in perceived blur of moving objects in a 3D environment. Of course, a prerequisite of a high refresh rate is that the video stream (whether it be supplied by film or from a virtual environment) needs to produce images at this same rate (Didyk, et al., 2010).

Technically the DK2 represented one of the first truly 6DOF HMDs, as it enables both orientational and positional tracking. The low persistence displays also resulted in more crisp and clear imagery (see Figure 12). Persistence in display technology pertains to the time interval between a frame (or image) being displayed and the next frame that replaces it. Longer persistence intervals create motion blur, which is unfavorable for VR applications.[9]

---

[7] https://web.archive.org/web/20160203172109/https://www.oculus.com/en-us/blog/announcing-the-oculus-rift-development-kit-2-dk2/

[8] https://www.researchgate.net/figure/Oculus-Rift-Development-Kit-2-Fig-1-Oculus-external-positional-tracking-camera-with_fig1_310142225

[9] https://developers.google.com/vr/discover/fundamentals

Figure 12: An image that shows the internal display and screens used for the Oculus Rift Development Kit 2 (Goradia, et al., 2014).

2.3.2 VR System Interaction

One of the fundamental issues that VR technology has faced and still does, is that traditional means of interaction with computers are not desirable for this new modality. The precision, robustness, and accuracy of traditional 2D display interaction has been established for a long time. Indeed, in the realm of computer gaming, the ability of professional eSport athletes (gamers) to master this form of interaction exhibits its success. The dexterity and precision that eSport athletes exhibit through the computer interface mirrors that of other athletes who are experts with their equipment in traditional sports (Watson, et al., 2021). Of course, VR demands a totally different way of interacting with the computer, as the user's focus determines the perspective of a virtual environment that is seen. Crucially, VR interaction needs to recognize that the new medium of Human Computer Interaction that it enables demands a totally new set of use-cases and therefore applications.

There are three primary means of interacting in contemporary VR. The first encompasses eye-tracking to navigate through an application, or a user's eyes being monitored and their sustained focus on an element dictating its selection. This form of interaction is common in portable VR such as Google cardboard or Oculus Go applications. The second means of interaction is achieved through the use of hand-held controllers, much like the controllers employed by popular gaming consoles. Finally, the third means of achieving VR interaction is achieved using hand-tracking technology. This third kind of interaction technology can be further divided into two important systems. The first is hands-body-free tracking, a system which does not require the user to wear a device (Guna, et al., 2014). The second system is hand-tracking, which involves utilizing wearable devices that then relay information to a processor. In the remainder of this section, however, the term hand-tracking will refer to the former system (hands-body-free).

2.3.2.1 Hand-Tracking

The desirability of hand-tracking is easily understandable. Traditional methods of HCI encompass the use of command-line and Windows/Icons/Menus/Pointers (WIMP) solutions. However, the need for more natural forms of computer interaction is becoming increasingly important for specific computer system usage (Van Dam, 1997). Indeed, ideal interaction in VR would encompass better and more natural ways in which people interact with the environment (Krueger, 1991), and a truly natural interaction would be one that adheres to how humans interact with the physical world (Cipresso, et al., 2018). As a result, hand-tracking potentially provides one such more natural way to interact with a virtual environment. The appeal of hand-tracking technology indicates how this form of HCI could enhance the immersion and presence a user might feel in a VR application, and this concept of adding to a user's perceived immersion is an underlying reason for the research that is detailed in this thesis. Moreover, just as hand-tracking is a desirable and more natural form of interaction in VR, so too is immersive sound a more natural and organic way in which the audio of an experience might influence the immersion of the experience.

The origin of hand-tracking as input for VR systems dates back to the 1970s. VPL Research created the first commercial hand-tracking glove, called 'the data glove', which was able to somewhat accurately detect the bending of a wearer's fingers (Yang, et al., 2019). Of course, significant advances in this technology and the current computing paradigm have yielded a new era of devices. At the forefront of consumer-level hand-tracking technology are the Leap Motion Controller (see following section) and the on-board hand-tracking offered by the Oculus Quest and Quest 2. The current technology was, however, preceded by the Microsoft Kinect,[10] a device which offers full body tracking but does not provide the capability of precise finger and hand tracking (Guna, et al., 2014). The principles used by the Kinect are employed by both the Quest HMDs and the Leap Motion Controller (LMC). This technology leverages principles of Image processing and the use of infrared cameras and sensors, but exactly how the devices exactly achieve hand-tracking is protected by patents and intellectual property. However, a brief discussion on the LMC's technology will be presented in the following section.

2.3.2.1.1 Leap Motion Controller

The LMC was released in 2013 and, at the time, represented a big step forward in providing an accurate and precise hand-tracking solution. In addition, the controller greatly improved upon competing gesture-detection technology. Since then, Leap Motion (now Ultra Leap, the company that engineered the device) has not released a later iteration of the device. However, their software solutions that accompany the LMC have greatly improved the tracking capability and use cases of the controller. In addition, the device was initially used for hand-tracking with general use-cases. However, in 2016, the company provided a suite of software that included specific VR support for their device.[11] In 2021, the company released a new Software Development Kit (SDK), which greatly improves upon numerous issues that their previous SDK did not address (more on these developments is detailed in the following chapter).

[10] https://developer.microsoft.com/en-us/windows/kinect/
[11] https://developer.leapmotion.com/orion

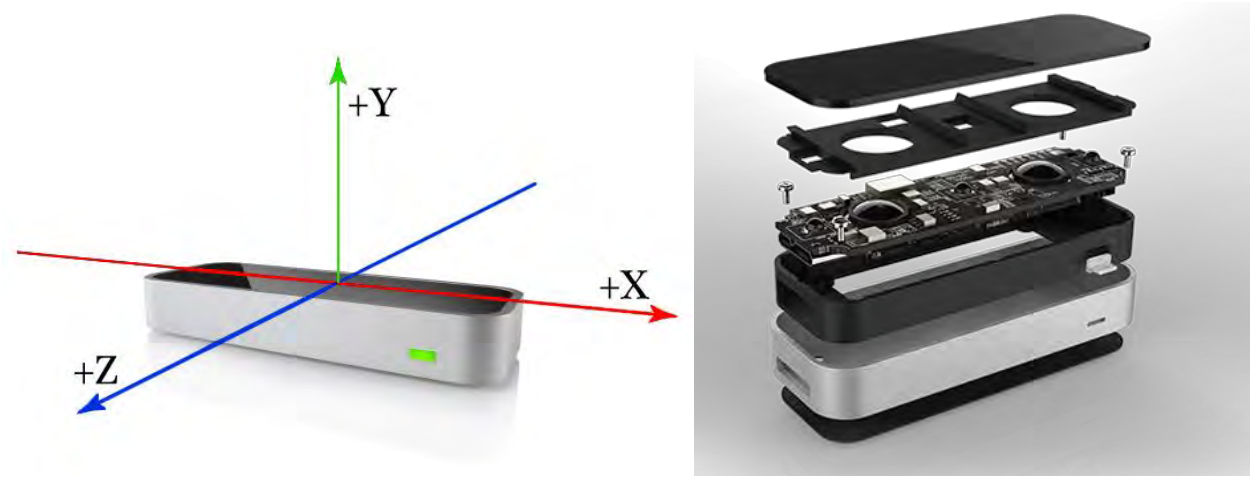

Figure 13: The left picture of the leap motion controller indicates the right-handed coordinate system it employs.[12] The right picture shows the different layers of the device. [13]

By utilizing infrared imaging, multiple cameras can accurately detect objects in real time within the field of view (FOV) of the controller. The controller has three infrared LED emitters, and two infrared cameras (see Figure 14). It can be assumed that the device achieves accurate hand-tracking by employing the stereo vision principle to determine the three-dimensional position of objects relative to the controller, and can therefore be classified as an "optical tracking system" (Guna, et al., 2014). In practice, this principal functions by interpreting the two separate images produced by the cameras to approximate a user's hand position (Weichert, et al., 2013). An image recorded by one of the cameras, can detect a hand by comparing this image against a classified image that contains a hand (or set of images), thereby using concepts of object detection in image processing (Viola & Jones, 2001). Once both images have correctly classified whether a hand is present or not, the three-dimensional location of the hand can be deduced by translating the stereoscopic pair into three dimensions.

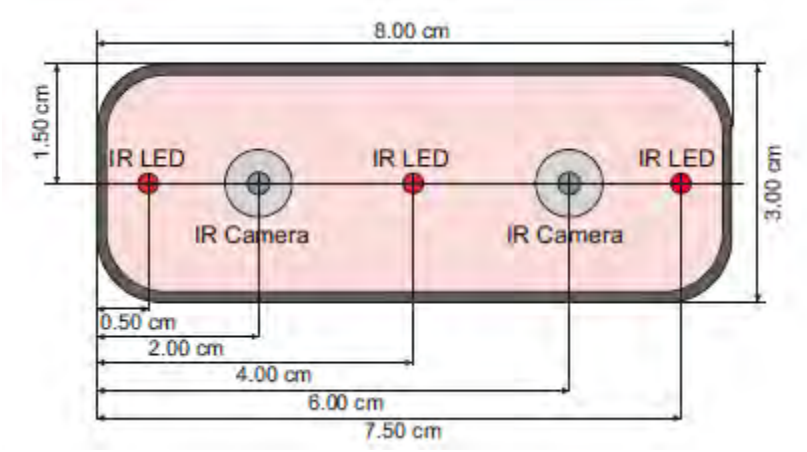

Figure 14: A schematic view of the Leap Motion Controller that shows the positions of the infrared LEDs and cameras that are contained by the device (Weichert, et al., 2013)

[12] https://developer-archive.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html
[13] https://blog.leapmotion.com/leapuvc/

If one assumes the perspective of the software that utilizes the sensory information provided by the device, the developer documentation indicates how the implementation uses a 'Frame' object at the root of its data model. This object contains any tracking information of whatever hands are present at the specific moment in time. If a hand is present in the frame, it is recognized according to a 'hands' class that contains information regarding the orientation of the hand. Each finger of the hand is also represented by a class that identifies the type of finger, and the positions of the four bones of the finger (see Figure 15). Each finger also contains a *TipPosition* and *Direction* vector which are used to determine the direction in which a specific finger is pointing. The SDK provides even further class abstractions, but they are outside of the scope of this section. This information serves as an indication of how a developer can utilize the SDK in detecting specific gestures (if specific finger position and direction vectors can be compared with a threshold value). Indeed, the information also indicates how the controller accurately detects hands and fingers at a single moment in time and facilitates the use of hand-tracking for developers in their applications.

The field of view of the controller is an inverted pyramid. It has been suggested that the device can comfortably track objects up to 60 centimeters above the device's surface (Guna, et al., 2014). By virtue of the Leap Motion software, and the cartesian coordinate system that is used (see Figure 13), the controller can accurately detect a user's hands and fingers, and then translate this information into a virtual space (i.e., a VR environment).

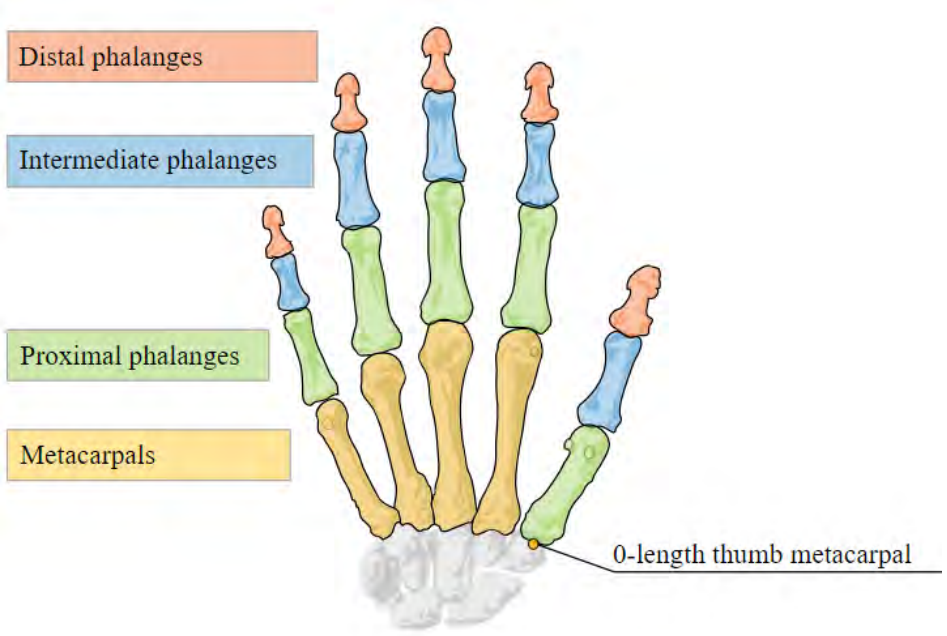

Figure 15: A diagram of the left hand of a human. This diagram indicates the four bones of each of the five fingers.

When analyzing the performance (with regards to tracking reliability) of the LMC, Guna and their colleagues determined that the device struggled to accurately track the dynamic movement of objects when the objects were more than 25 centimeters above the surface of the controller (Guna, et al., 2014). In addition, their research showed that the LMC was better at tracking objects at the center of the device's FOV. The final issue they identified was that the sampling rate of the controller was erratic and could pose a problem if the controller were to be used in real-time systems. They postulate that

these issues may be related to the API (Application Programming Interface) or the sensory system of the controller. As the Leap Motion software has developed significantly since this research was conducted, a repetition of the experiments might reveal whether the issues they identified are indeed software related. Despite these issues, the research suggests that the LMC represents a huge step forward for affordable tracking systems (Guna, et al., 2014). For example, Weichert and his contemporaries indicate, in their analysis of the accuracy and precision of the LMC, that the device performed admirably, and outperformed competing devices like the Microsoft Kinect (Weichert, et al., 2013).

It is important to recognize that most of the reviewed studies analyzing the LMC apply to specific use-cases for the controller. Additionally, none of the studies analyzed the controller being used as an interaction modality for VR applications beyond simple grab interactions (Masurovsky, et al., 2020). As a result, the LMC was desirable for enabling a more natural form of interaction for this project's implementation and research aims.

2.3.2.1.2 VR Suites

The software solutions for VR development are numerous due to the variety and abundance of different VR HMDs. Chapter five (Constructs to Enable FMOD, Unity and ImmerGo Collaboration) will detail some of the specific solutions and SDKs that were utilized in this project. However, it is worth noting some alternate solutions that are available and some useful innovations in the following paragraphs.

As the research domain of this project lies within the Unity game engine domain, the focus of this section will look at specific VR tools that are of importance and exist within the framework for this game engine. The Unity game engine is a multi-platform production engine, as it can be used for the development of games, films, and VR applications for all the major gaming platforms (PlayStation, Xbox, pc gaming, and the mobile platforms). As such, it can support VR for each of the platforms that support VR development. At the time of this project's development, a standard for cross-platform development was in its infancy in the form of OpenXR.[14] Unfortunately, most of the major proponents of VR were using their own separate software suites. In accordance, Oculus provides their developers with the Oculus Integration package, which enables Unity developers to access Application Programming Interfaces (APIs) for the Oculus HMDs. Similarly, Valve released their SteamVR plugin for Unity that interfaces with their SteamVR runtime and enables VR development from within the Unity game engine. Of course, Sony followed suite, but requires that developers are in fact licensed PlayStation developers to access their VR development capabilities for the Unity engine. Indeed, the same trend applies when developing VR applications for Google's VR products, and an associated SDK and plugin is needed for VR development from within Unity.

This disparate and splintered ecosystem for general VR development has been a major reason why VR applications are generally not developed to be cross-platform applications. Additionally, this has meant that developers need to become accustomed to and learn each of the various frameworks before being able to develop their respective applications. Not surprisingly, this has been a major issue for the average consumer as well, as the fragmentation of the market and ecosystem has resulted in a plethora of VR titles that are not accessible if a consumer does not possess a specific HMD. However, this makes sense from a corporate perspective, as the VR market still has so much financial potential and, at this point, there is still no clear market leader. Therefore, if a leader is established, then their software

[14] https://www.khronos.org/api/index_2017/openxr

framework could potentially become the general standard for VR development. Fortunately, this current paradigm is shifting dramatically as OpenXR is paving a way for an industry standard by developing an "open standard that provides high-performance access to Augmented Reality (AR) and Virtual Reality (VR) platforms and devices". In July 2021, Oculus announced that they would be depreciating their proprietary Oculus APIs and moving across to the OpenXR framework for future development.[15] This is a seismic shift for VR developers, as Valve and therefore SteamVR also announced their departure from proprietary APIs in favor of OpenXR in late 2020. This change is significant, as it indicates that the entire VR market will, in some ways, unify. The hardware market will undoubtedly remain competitive, but it can only be beneficial for the software side of the ecosystem to exhibit congruency and cross-platform functionality – benefits which will not only assist the end consumer the most, but also allow for future VR applications to learn from and profit from one another.

2.4 Current Applications of Virtual Reality

The following section will provide a brief analysis of how VR technology is currently being utilized by society. Accordingly, sectors affected by the technology will be broadly examined. An additional document detailing specific case studies for this section is provided in the Project Resources file as online subsidiary information. Of course, the ultimate usefulness of VR is still to be determined, as the technology does represent a new paradigm by which humans interact with computers (Cipresso, et al., 2018). Therefore, the following sections broadly indicate the current applications, but one should realize that VR potentially offers nearly limitless uses. Before introducing some of contemporary applications for VR, it would be useful to briefly discuss the recent global interest in a Metaverse.[16] This term was briefly introduced in the opening chapter of this thesis, but it is worthwhile to provide a more robust definition of what this 'Metaverse' could and will be. Indeed, Facebook's recent rebranding and restructuring of their company under the banner of Meta, indicates a concerted effort to bring a Metaverse to life. As such, the applications of VR will begin to unify with the inception of this concept. According to Mystakidis, the Metaverse can be defined as a "post-reality universe, a perpetual and persistent multiuser environment merging physical reality with digital virtuality." (Mystakidis, 2022) Accordingly, it can be argued that all the applications of VR that are detailed in the subsequent paragraphs would eventually form part of this unified ecosystem.

2.4.1 Art

> *"[In the context of digital art] … virtual reality may become a second nature that profoundly challenges the basis of our concepts of perception and the dualism of 'flesh' and 'spirit'."*
>
> — Christiane Paul (Paul, 2003)

The digital revolution has impacted most spheres of humanity, not least the realm of art. Historically, the mediums that artists employ have advanced alongside society's advances. The digital age is no exception, and digital technology, including VR, can be considered a new medium by which artists can evoke feelings and create new works (Schiavoni & Gonçalves, 2017). Indeed, VR, if successful in its

[15] https://uploadvr.com/facebook-deprecates-oculus/
[16] https://time.com/6116826/what-is-the-metaverse/

ability to suspend disbelief, could become a medium that sits alongside the novel, cinema, and television (Bates, 1992). While this is indeed a possibility, Bates argues that, if one excludes the technological issues with VR, this new medium will have to carefully consider how it handles both the content presented and the style of said content. His argument contextually exists within The Second Era (see section 2.2), but if one looks at a more contemporary context, then it can be claimed that VR art will have to overcome the passivity of traditional mediums. Therefore, the interactivity that VR enables will play a significant role in how successful it becomes as a medium for digital art in the future. Figure 16 indicates where the domain of VR art lies in relation to other contemporary artistic domains.

Conceptually, art has always had a clear understanding of immersion. Out-of-body experiences can generally be understood as a definite objective of artists in any medium (Grau, 2003). Of course, VR makes these notions of disembodiment much more literal, as perfect VR (see section 2.5) denotes the ability of a VR system to totally immerse the user, such that they cannot differentiate between what is a virtual reality and what is not. For digital art, and in particular VR-based art, the barrier for entry to the technology is sizeable, however (Nebeling & Speicher, 2018). Indeed, an artist will require some sort of programming literacy to utilize VR development tools, until either the tools become less exclusionary, or artists become more comfortable with programming, VR art for the most part will have to be produced using software developers, and/or production companies.

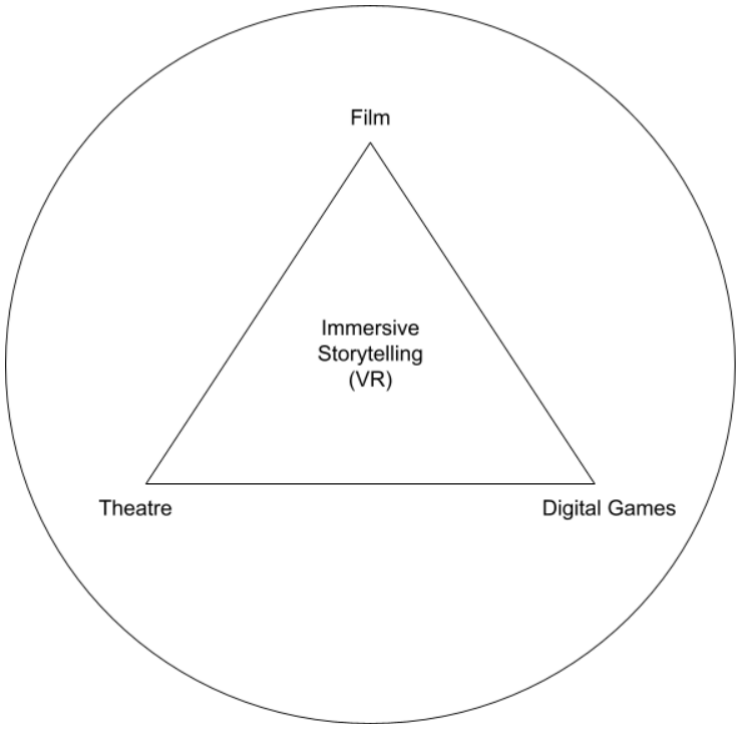

Figure 16: A diagram that contextualizes immersive storytelling as an artistic medium in comparison to other mediums. The surrounding sphere should be interpreted as 'Art' as a whole.

One of the major use cases for VR has been within the sphere of film festivals and in art exhibition spaces. Indeed, both are currently one of the more accessible ways in which the public can experience

VR technology. The Subterranean Imprint Archive (SIA) is an immersive experience that the author of this thesis helped create as the sole software developer of the experience. A substantial amount of its development hinged on the skillset that the researcher accrued over the course of this project. A description of the immersive experience is beyond the scope of this research, but it has premiered at the International Film Festival Rotterdam,[17] as well as the MUTEK 2021, and the Akademie der Künste.[18]

2.4.2 VR in Journalism

It is crucial to recognize that consumer VR technology has generally targeted the entertainment sector, i.e., through gaming applications. While this is the case, it can be argued that due to the nature of the technology, acting as a unique modality in which humans can interact with computers, its potential uses stretch far beyond the entertainment industry. Numerous sectors of society have slowly or more dramatically adopted and been influenced by VR technology. In particular, the creation of 360-degree videos, which can be showcased using VR HMDs, has blossomed since the first major VR headsets were released. These videos and experiences can fall into numerous categories, but here they will be referred to as immersive stories, and news organizations are becoming a major driving force for the generation of such stories and 360-degree content. For example, major groups, such as the *Guardian,* the *Wall Street Journal*, and the *New York Times*, have all created and distributed 360-degree media to their readers (Mabrook & Singer, 2019). The desirability of the use of immersive stories in journalism and film is obvious. It has been documented that people respond to VR media with a higher degree of empathy than alternate forms of media (Schutte & Stilinović, 2017). This, therefore, underlines a clear and important aspect of the technology when used in both film and 360 videos. Indeed, because of the empathy-inducing capacity of VR technology, recent research has proposed that VR should be used to modify attitudes of the public to support some of the major causes of our time, such as climate change awareness and protest (Markowitz & Bailenson, 2021).

2.4.3 Entertainment, Exergaming, Social VR, and Gaming

One of the biggest pitfalls in early VR development has been that projects have been modelled on games that exist for traditional two-dimensional (2D) mediums/displays. Of course, this has its merits, but also pitfalls. As has been detailed, the digital game industry has had a profound impact on the development of VR technologies. However, the current context of digital games denotes the development of games that demand a certain level of reflex, decision-making, and precision of control that contemporary VR interaction does not allow for in similar use-case scenarios. Of course, there are exceptions to this precedent, but in general this holds true. The design principles manifest in the digital game industry are a product of three decades worth of research, investment, and innovation in the digital game industry, compounded by the development of extremely precise and accurate gaming peripheral devices (i.e., gaming mouses and keyboards).

The more successful VR gaming application releases (of which there are few) nevertheless exhibit a unique and astute understanding of the platform and, indeed, medium that they are employing. For example, the developers of *Half-Life Alyx*,[19] recognized immediately that the real value of their project was not just compelling gameplay but, above all else, environmental interactions and the telling of a highly immersive story in a beautifully crafted and interactable world. In other words, like Murray, they recognize that a compelling tale can be as immersive as reality (Murray, 1999).

[17] https://iffr.com/en/iffr/2021/films/the-subterranean-imprint-archive
[18] https://www.adk.de/en/projects/2021/gedaechtnis/individual_projects/what-stays_archiving-care.htm
[19] https://www.half-life.com/en/alyx/

One of the more interesting innovations in the VR entertainment sector is what has recently been coined as 'exergaming'. The term has existed for some time and has been defined by Oh and Yang as "playing exergames or any other video games to promote physical activity" (Oh & Yang, 2010). The realization of VR technology has renewed interest in the genre and has given it a new medium in which to develop. At the helm of this new sub-genre, is the VR experience *Beat Saber*, which is a relatively unique application in that it has a very high skill cap and relies on swift reflexes and a user's ability to make quick decisions in real time. These developers realized that their focus was not going to be narrative, but rather using a highly intuitive means to interact with an environment via current interaction mechanisms.

VRChat is a fascinating case study of what the technology might begin to incorporate into its development, namely, the intersection of social media and multiplayer experiences within the medium of VR. This type of composite application has recently been defined as 'Social VR'. 'Social VR' can be best understood as a combination of social media and multiplayer gaming, housed within the context of a Virtual Reality environment. It is contended that 'Social VR' interactions are more like face-to-face communication than either social media or multiplayer gaming communication. Indeed, 'Social VR' has been argued to "afford[] a broader spectrum of communication modes including both verbal and non-verbal interaction such as voice, gestures, proxemics, gaze, and facial expression" (Maloney, et al., 2020).

In the entertainment industry, VR application developers should not consider their applications the same way they would consider digital gaming applications. Ideally, developers should analyze how their application can best leverage the differences that a VR experience provides as opposed to that of a digital game. It is not difficult to recognize that applications or experiences that better model reality and how we interact with our physical world are preferable to a 'gamified' scenario. A suggestion, then, is that developers should examine *how a real scenario can be gamified*, as opposed to *creating a game, and subsequently applying the realism that VR allows*. Although digital games are an important component of this research, the scope of the research does not include analysis of certain pioneering and popular contemporary digital games. However, a game such as Fortnite (created by EPIC games) is a useful benchmark to determine the precursors and design decisions that are taken when implementing and developing a digital game, and how these might and should differ from the processes taken when creating a VR experience.[20]

Figure 17 provides a useful perspective on how VR differs from digital games according to three different domains. The y-axis represents the form which Virtual Environments might take – conceptually, concretely, and in three dimensions. These distinctions indicate that VR environments have the potential to provide a higher perception of three dimensions, which makes sense considering that digital games utilize two-dimensional screens to exhibit three dimensions, while VR systems use HMDs to accurately model how we perceive the physical world (i.e., in three dimensions). The x-axis refers to the Ludic status which, according to Garner, "refers to the extent to which something embodies traditional gameplay properties" (Garner, 2017). On this axis, it is easy to see that VR can exist with interaction types that are reminiscent of digital games, but additionally it can include forms of interaction that are much more natural, such as using hand-tracking or eye focus as interaction

[20] https://www.epicgames.com/fortnite/en-US/home

modalities. These transcend traditional gameplay properties. Finally, the z-axis is the Reality-continuum, and, self-evidently, it encompasses how 'real' an experience might be. Overall, then, VR once again supersedes digital games, in that it can provide a much more compelling representation of virtual worlds and physical reality.

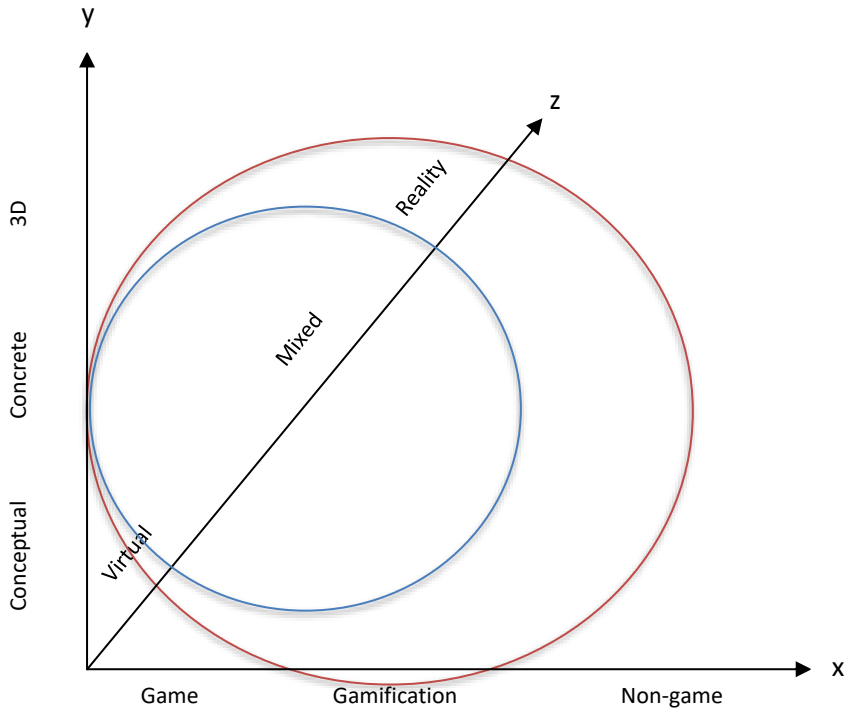

Figure 17: A diagram that illustrates how VR compares to digital games in three domains (Garner, 2017). The x-axis represents the Ludic status, the y-axis represents the form of the virtual environment, and the z-axis represents the reality continuum.

2.4.4 Medicine

Mazurek and his colleagues propose that the first use of VR in medicine occurred in the 1990s, when it was used to help visualize medical data (Mazurek, et al., 2019). Since then, the adoption of VR into the healthcare sector has, in part, been facilitated by the International Society for Virtual Rehabilitation, which was established in 2009. This group has been influential in encouraging the introduction of new technologies into motor, psychological, cognitive, and social rehabilitation (Mazurek, et al., 2019). Through a statistical analysis of scientific research pertaining to VR in medicine, Yeung and associates have also revealed that the term gained significant traction and growth in the medical research fields during the 2010s. Significantly, their research suggests that in the clinically related research fields, the term has been used more, on average, than in both the areas of Computer Science and Engineering (Yeung, et al., 2021). This allows for the key deduction that VR in medicine can no longer be considered niche. On the contrary, it is a burgeoning area of medical research that continues to grow off the backdrop of the Third Wave of VR and the continually increasing availability of computing power globally.

2.4.5 Education

It is worthwhile recognizing that education as a field encompasses various different levels and domains, such as higher education, K12 education, job training, and vocational training. Nevertheless, they all share similar themes and tenets. Kavanagh and contemporaries argue that VR technology has been utilized within education for five decades, but that its widespread implementation in education remains elusive. They attempt to provide answers for this, and their conclusions mirror the reasons why VR did not successfully infiltrate the commercial sectors until the third era of VR. Most notably, they suggest that the costs and logistics required to deploy VR systems have remained prohibitively expensive up until now (Kavanagh, et al., 2017). When considering existing implementations of VR in education, their research postulates that most focus primarily on increasing the 'intrinsic motivation' a student might have for their work, i.e., they do not necessarily seek to help learners achieve learning outcomes.

Nevertheless, there is a considerable degree of enthusiasm about how VR technology might be employed more effectively within the education sectors. In a systematic review of virtual reality applications in higher education, a few core deductions about the technology and its use in education were made. The research suggests that one of the most significant problems associated with the technology's lack of adoption in classrooms and curricula is that analysis of the systems and applications that might be used in education generally seek to evaluate their usability, as opposed to determining the success of learning outcomes facilitated by the systems. This then engenders a scenario where the true function of VR in education has yet to be properly documented, researched, or understood (Radianti, et al., 2020). In addition, there is a suggestion from the findings in this review that most VR applications in education are used experimentally and for developmental work, as opposed to being a core component in actual teaching.

2.4.6 Industry

VR has influenced industry across numerous different disciplines and fields. There is research that suggests that it is beginning to play a major role in the areas of mining, tourism, aviation, product design, manufacturing, and engineering. For example, it has been suggested that up to 25% of large companies globally are using VR/AR technology in some capacity (Bos, et al., 2021). This 'early' adoption of the technology will result in it disrupting and influencing more traditional industrial methods. In addition, it has also been proposed that medium and smaller businesses and companies will start to implement the technology in their operations as the cost price and accessibility of the technology improves (Bos, et al., 2021). In the sectors of product design and manufacturing, it has been proposed that VR and AR might have a dramatic effect in allowing companies to rapidly prototype products, and to streamline design and production costs, while maintaining product quality (Liagkou, et al., 2019; Turner, et al., 2016). Moreover, the possibility of operating real manufacturing processes remotely via employees using VR technologies is becoming an increasing reality. And, finally, another offshoot of the impact VR might have on industry lies in training, since manufacturing and product design employees can train virtually before they encounter risky environments or situations in their workplace (Liagkou, et al., 2019).

2.5 The Need for Immersion in 'The New Age of Media'

When VR was defined in the opening section of this chapter, immersion was explained as the sum of a few important factors, and the following discussion will attempt to explain how these factors operate. The first section will review relevant literature to help provide a definition of immersion that pertains to the research conducted in this body of work. In so doing, the section will investigate the factors that render an immersive environment, namely, perception, (tele)presence, and interactivity (see Figure 7). Following this, the second section will shed light on the concept of immersion by explaining it using a layered approach, and a scenario of this approach will be provided to help understand the reasoning behind it. This layered approach is of particular importance, as it mirrors the developmental process of a VR application. Indeed, a game engine is composed of multiple libraries (or layers), and these all combine to create an interactive and compelling experience. The final section will look to provide a taxonomical approach to classifying immersive systems, and how one might apply this classification. The predominant aim of this section is to introduce immersion as a multifaceted concept and, in so doing, provide a key reference point for readers moving forward through this body of work.

2.5.1 Defining Immersion and its Components

When reviewing research that pertains to the phenomenon of immersion, there are numerous ways in which the concept is defined. However, there is generally an agreement on which factors are present in an immersive experience. Nevertheless, it is worthwhile to briefly introduce the concept historically. From this historical perspective, immersion is not a concept that only pertains to immersive media in the 21st century. In fact, as Murray argues, any medium (literature, film, etc.) with a compelling narrative can constitute a 'virtual reality'. While this sentiment does not expressly adhere to the definition proposed for VR at the start of the chapter, it serves the purpose of better explaining immersion outside of the contemporary technological paradigm. Indeed, Murray's suggestion is that immersion is the feeling one gets when one is transported to this virtual place (Murray, 1999). A correlation for immersion in the physical world is the experience of being submerged in water. Murray contends that "we seek the same feeling from a psychologically immersive experience that we do from a plunge in the ocean or swimming pool: the sensation of being surrounded by a completely other reality, as different as water is from air" (Murray, 1999). This comparison provides a powerful analogy for what it is like to experience a fully immersive VR experience. The following paragraphs will more accurately define immersion in the context of the research detailed in this thesis.

Immersion in VR can be defined as "the subjective experience of feeling totally involved in and absorbed by the activities conducted in a place or environment, even when one is physically situated in another" (Mütterlein, 2018). Another similar definition for immersion is "a perception of being physically present in a non-physical world. The perception is created by surrounding the user of the VR system with images, sound, or other stimuli that provide a very absorbing environment" (Freina & Ott, 2015). In reading about the various factors that tie into definitions of Virtual Reality, there is often confusion as to how immersion and telepresence differ, but this conceptual understanding of immersion draws a clear distinction between the latter and telepresence – namely that telepresence pertains to the feeling of being transported to another place, while immersion pertains to the subjective feeling someone experiences when performing activities within this 'other' space (Mütterlein, 2018). This difference between the two concepts thereby presupposes that for a user to be totally immersed within a VR experience, they must first feel as though they are present within this location (the virtual space). In other words, immersion is not possible until telepresence is experienced by a user. On this assumption, it is reasonable to postulate that telepresence itself is also only possible if a user can appropriately

perceive whatever environment it is that they encounter. To provide more clarity, telepresence depends on perception, and it is only possible for a user to be immersed in an environment if they are feeling 'present' within this environment.

If a user's experienced immersion in an environment requires them to perform activities, then it stands to reason that how they perform these activities is critical, which ties into the final factor that was mentioned earlier, namely interaction. It has been suggested that interactable environments generally contribute to feeling much more present in the environment than passive ones – environments where a user does not interact with anything and is merely an 'observer' (Steuer, 1992). This then encompasses an important dependency of immersion on interactivity and telepresence, that is in turn dependent on perception (see Figure 7). It is useful to observe what Slater perceives to be the real power of VR: that, not unlike illusion, the technology can induce wonderful responses in individuals upon their perception of the virtual, albeit that they know what they are experiencing is ostensibly an illusion, and not real (Slater, 2018).

If one wishes to define the aspect of perception in immersion differently, then Wang has proposed a definition that encompasses perception in a more relatable way. Their research revolves around social VR, but certain principles that are detailed in this research apply to all forms of VR. They argue that the three factors affecting VR technology are indeed Imagination, Immersion, and Interaction (see Figure 18). In this model, which can be called "the triangle of virtual reality technology", it is suggested that immersion consists of visual, acoustic, and tactile components (Wang, 2020). This further justifies the usefulness of providing a layered understanding of immersion (see following section). The interaction factor includes naturality and real-time components that model the qualities of normal social functions. Finally, they indicate that imagination refers to the ability of users to identify how to navigate through a VR environment based on their judgement, reasoning, and information gathered from the virtual context – in other words, how they 'perceive' their environment (Wang, 2020).

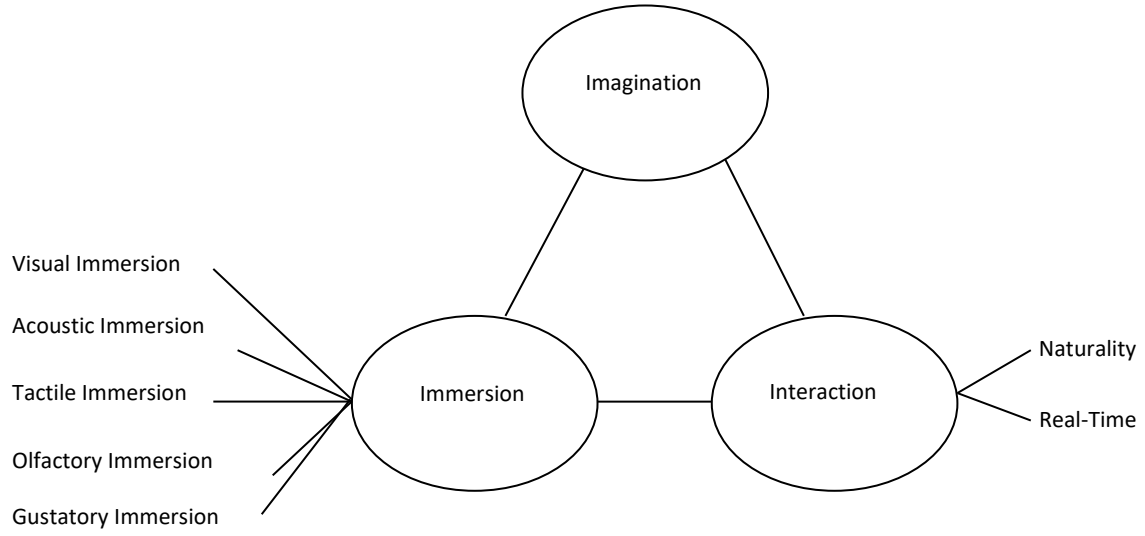

Figure 18: A diagram adapted from Wang's study into social VR. The Olfactory and Gustatory immersion influencers were included in this adaptation (Wang, 2020).

2.5.2 The Layers of Immersion

Having provided definitions for immersion and the factors of perception, interactivity and telepresence above, the following section will demonstrate how the concept of immersion can be further analyzed in order to describe how it is affected by different types of sensory information. Indeed, a VR system able to stimulate each of the senses could result in perfect VR (defined earlier as a system so immersive that users would be unable to tell whether they are in the virtual or real world). This layered understanding is of particular use to the research performed in this thesis, as it denotes the importance of each of the senses in rendering a truly immersive environment. The following figure indicates the multimodal layers used in rendering immersion. It is apparent that the primary layers used in contemporary VR systems encompass the visual, auditory, and haptic senses. If used correctly, these layers can all contribute to the requirements of a highly immersive system (see following section for a grading of immersive systems). The diagram also indicates that if every sense were to be incorporated into a VR system with a high enough fidelity, then it would be feasible to experience perfect VR.

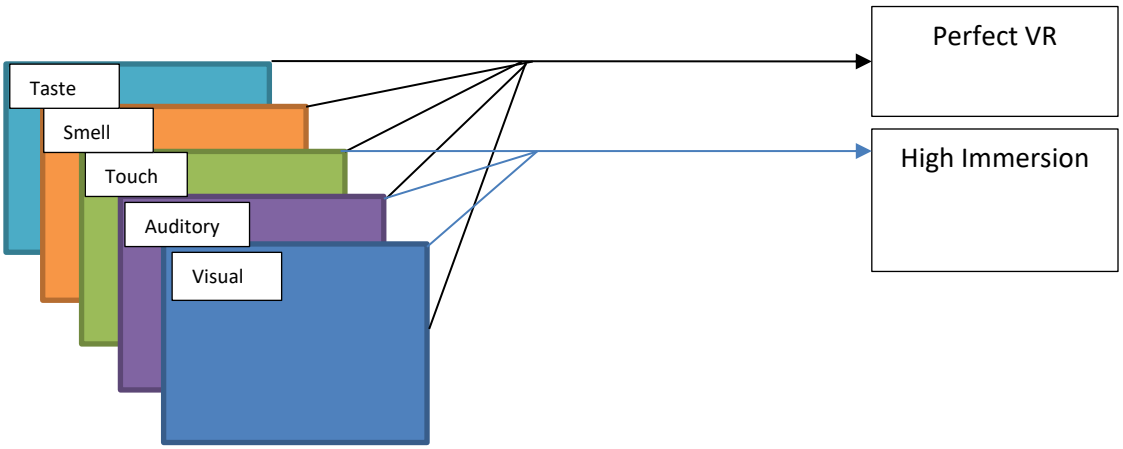

Figure 19: In the figure above, each of the colored squares represent a sensory (modal) layer in a multimodal VR application.

This multimodal perspective on rendering immersion is well supported, as Freeman and colleagues suggest that amongst the most important contributors to feelings of immersion and presence is synchronized multimodality (Freeman, et al., 1997). As mentioned earlier, another important reason to support this layered view of immersion is that it directly mirrors how modern implementations of VR applications are produced using game engines (game engines and their interfaces are discussed in chapter four). The developed games and environments can be broken down into their various layers and, in reverse, a game or virtual environment is the sum of all its layers. This should indicate to the reader why gaming publishers and developers generally have large teams, since each team focuses on a specific layer of the project.

2.5.2.1 The Need for Audio

> *On the invention of the phonograph, "Science had eventually left the cold realm of reason and the intellect to enter the domain of emotions."*
>
> — Elodie A. Roy

The need for multimodality in VR applications and, in particular, the auditory modality is investigated and presented throughout this body of work. Chapters three and four examine spatial audio and its necessity in VR development. However, it is worthwhile to briefly discuss this here, as the aims of this thesis pertain to the immersive audio component (or layers) of VR applications. It is also valuable to indicate that, although the visual aspect of VR is generally discussed and debated in the field, the development of audio during the same period has allowed for both technological streams to converge in the 21st century. Throughout VRs development, in fact, pioneers have stressed the importance of audio in the technology's development. Heilg's Sensorama system incorporated sound, smell, touch, and sight in the project (Martirosov & Kopecek, 2017). In detailing the project, he emphasized that 'directional sound' plays a crucial role in influencing perception in VR. Indeed, it has been suggested that the auditory sense is the modality that is most often at work in evoking illusory sensations in people, and is a critical aspect of VR experiences (Garner, 2017). Garner suggests that, although audio was deprioritized in the second era of VR technology, the audio technology sector has subsequently blossomed. He postulates that the convergence of VR visual technology and the progress of audio with inventions such as the headphone set the scene for contemporary VR technology (Garner, 2017).

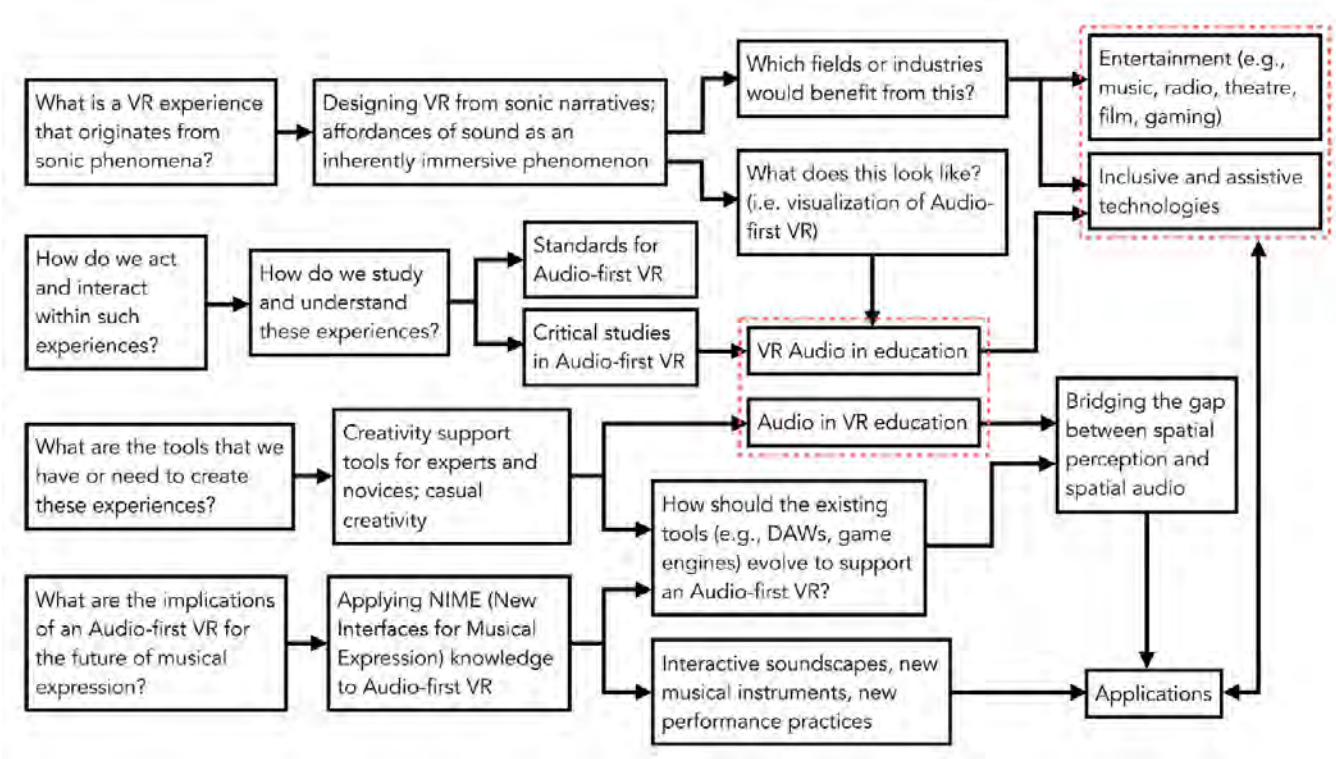

Figure 20: A diagram that posits how important audio is in the context of VR technology (Çamcı & Hamilton, 2020).

The above diagram forms part of a discussion put forth by Çamcı and Hamilton that looks to put audio first when considering VR as a medium for artistic production, and aims to motivate? the precedence of audio from a developmental perspective. In accordance with the theme of this section, their work provides a very real context in which audio in VR is treated as a "critical" area for the construction and implementation of immersive VR applications and systems.

2.5.2.2 A Scenario Describing the Layers of Immersion

In the physical world it is easy to abstract an operation into sensory layers. The following scenario attempts to provide a simple illustration of how the sensory modalities, when applied to an immersive VR experience, contribute to a sense of immersion.

If a person opens a door and passes from one room into another, this input (opening the door and their decision to walk forwards) results in their ability to move through the doorframe and into the other room (the output; see Figure 21). To accurately virtualize this simple interaction in Virtual Reality is already a complex scenario. To best describe this virtual interaction, it is useful to apply a layered approach when describing the scenario (see Figure 19). From a visual layer, the two spaces need to be rendered accurately. The user must be able to correctly perceive their environment, that they are in a room, and that there is a door leading into another room. The user also needs to be able to reach out and see themselves touch the door handle (visualizing their hands and the doorknob). Once they have opened the door, they then need to observe that there is a space they can maneuver to through the doorway (the second room). If implemented correctly, this virtual experience may have the visual fidelity to totally immerse users. However, additional sensory stimulation would contribute to an even more immersive implementation.

For example, if we use an auditory layer on top of the visual layer, this will undoubtedly benefit overall immersion. Perhaps there is some music that can be heard originating from the second room. As the user opens the door, we could trigger a 'door opening' sound to reinforce the sense that they have interacted with the doorknob. Once the user opens the door, we can increase the music's volume or provide more clarity to the audio by removing a high pass filter (a signal processing effect that filters out higher frequencies) that was applied to the music while the door was closed. This auditory layer would ostensibly increase the user's perceived immersion, as the experience would engage them on both a visual and an auditory level. Finally, to provide feedback on the interaction present in this scenario, it would be useful to employ haptic feedback on the interaction device used to render the user's hands in the experience. For example, if they touch the door handle, a controller could vibrate to indicate that they are successfully interacting with the virtual object. This would therefore include the sense of touch in the system and potentially contribute to immersion even more.

This way of abstracting a real scenario into a virtual one, underlines a few critical aspects of VR and immersion. Firstly, that for a truly immersive VR experience, the experience needs to incorporate as many human senses/modalities as possible. It is important to recognize that, in terms of the current technological context, the senses of smell and taste are not usually evoked in normal computing systems. Of course, sight is a critical human sense in the current Human Computing Interaction paradigm, but the scenario detailed above hopefully indicates that the auditory layer could also be perceived as critical. As mentioned, computer vision has been a primary focus since the inception of the first two-dimensional displays. However, the importance of audio in VR should not be underestimated. Much of the research conducted in this thesis looks to determine the importance of audio in a VR experience, and what impact different ways of implementing audio in a project influence the perceived immersion of the system.

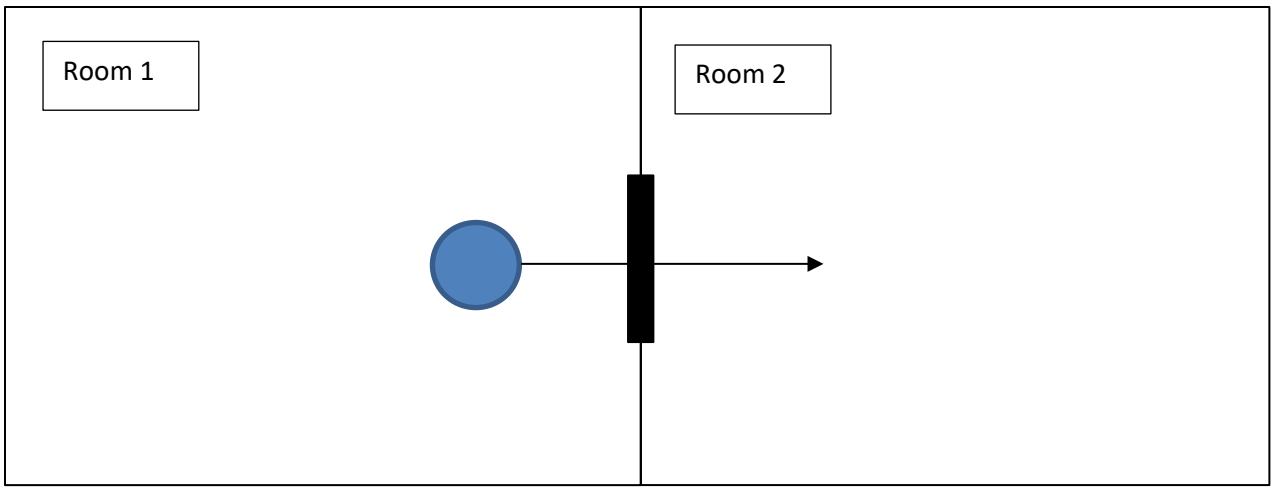

Figure 21: Physical layout of scenario. The blue sphere represents the person. the doorway is the black rectangle between both rooms, and the black arrow indicates the process of opening the door and passing through to the other room.

2.5.3 Grading Immersive Virtual Reality Systems

This concise section will show how VR systems can be graded according to the level of immersion they employ. Relevant literature on the subject was reviewed, and a classifying scheme has been adapted from Miller and Bugnariu. Kardong-Edgren and colleagues suggest that it is critical to correctly identify the qualities of a VR system so that it is possible to unify definitions of the technology in science and research. They indicate that the large variety of VR systems means that it is misleading to classify them all under one term, as they often provide different functionality (Kardong-Edgren, et al., 2019). The scheme of classification Miller and Bugnariu employ identifies immersive systems according to the level of immersion and presence they exhibit. Therefore, an immersive system can demonstrate *Low, Moderate*, and *High* levels of immersion. The scheme identifies these systems based on aspects of immersion that include the number of modalities employed in a system, the visual fidelity, motion capturing ability, and other features (Miller & Bugnariu, 2016). Table 1 represents this classification scheme:

Aspect of Immersion					
Level of Immersion	***Inclusive***	***Extensive***	***Surrounding***	***Vivid***	***Matching***
Low	Numerous signals indicating the presence of device(s) in the physical world (e.g., use of a joystick or mouse to control the VE, direct instruction from an experimenter during the task)	Only accommodates 1 sensory modality (e.g., auditory, visual, motor/ proprioceptive); stimuli are not spatially oriented	Computer monitor presentation with limited field of view	Low fidelity and visual/color resolution; display may replicate features of the simulated environment, but not in a detailed or specific manner	No motion capture; visual experience does not match proprioceptive feedback
Moderate	Some signals indicating the presence of device(s) in the physical world (e.g., noise from a computer fan, weight and movement restriction from wearing a safety harness)	Accommodates 1–2 sensory modalities (e.g., auditory, visual, motor/ proprioceptive); stimuli may or may not be spatially oriented	Large-screen projection with extended field of view	Moderate fidelity and visual/color resolution; display replicates some features of the simulated environment, but some detail may be missing	Body segment motion capture (e.g., head, hand); visual experience somewhat altered to match proprioceptive feedback based on head or body segment movement
High	Limited signals indicating the presence of device(s) in the physical world (e.g., the weight of an HMD or an eye-tracking device)	Accommodates >2 sensory modalities (e.g., auditory, visual, motor/ proprioceptive); stimuli are spatially oriented	Head-mounted device or surround projection	High fidelity and visual/color resolution; display closely replicates multiple features of the simulated environment in great detail (e.g., correctly placed, dynamic shadows)	Full-body motion capture; visual experience altered to closely match proprioceptive feedback based on whole body movement
Perfect VR	No signals indicating the presence of device(s) in the physical world	Accommodates all 5 sensory modalities	unintrusive device, or neural stimulation device	Exact replica of how user perceives physical world (1:1)	User's movements exactly and accurately map to virtual ones; no latency

Table 1: The table above provides a concise means to evaluate Virtual Reality systems based on their respective levels of immersion, and is adapted from Miller and Bugnariu's research (Miller & Bugnariu, 2016).

As can be seen, the different levels of immersion, when quantified using aspects of immersive systems, indicate a nuanced classification scheme by which VR systems can be categorized. The preceding table is provided as it appears in the research of Miller and Bugnariu, except the final level of immersion, *Perfect*

VR, has been added to indicate how it contrasts with contemporary immersive systems. If one observes the requirements for an immersive system that exhibits *High* levels of immersion, then it is reasonable to suggest that the system employed in the research conducted in this thesis conforms to this classification, despite full-body motion capture not being expressly enabled.

2.6 WHERE IS THE TECHNOLOGY HEADING?

> *"Virtual Reality is the most effective device ever invented for researching what a human being actually is – and how we think and feel."*
>
> — Jaron Lanier (Lanier, 2017)

In terms of recent technological trends, the most interesting has been the decision taken by Oculus to promote portable VR as opposed to tethered PC equivalents. This has been a shrewd decision, and it has been speedily vindicated, as the Oculus Quest 2 has become the number one selling VR headset. As of November 18th, 2021, Oculus is estimated to have shipped 10 million units globally.

A new paradigm of computing and Human Computer Interaction (HCI) is fast approaching. It should be underlined that, alongside the visual fidelity of the environment, the auditory and interactive aspect of the environment would also play a large part in realizing perfect VR. The incredible strides taken in recent years regarding computer vision, together with the adoption of immersive audio and object-based audio systems, seem to suggest that perfect VR is not an impossibility, but rather an inevitability. The evolution of the gaming industry also indicates an insatiable desire in people to be immersed in virtual environments, and the rise of eSports signifies the incredible control and precision that is possible within our current paradigm of human computer interaction. It is therefore not farfetched to envision a future in which VR interaction has also evolved to an equivalent point of accuracy and precision.

To conclude this section and chapter, a brief exploration of the ethics of VR will be supplied. Moor posits a model that describes how technological revolutions comprise three stages: the introduction stage, the permeation stage, and the power stage (Moor, 2005). Additionally, he suggests that "ethical problems increase as technological revolutions progress toward and into the power stage" – a sentiment that can be distilled further into a principle that states: "as technological revolutions increase their social impact, ethical problems increase". If one uses this model and applies it to the technology of VR, it could be argued that the revolution has reached the permeation stage and is heading into the power stage. As a result, Moor's principle should be heeded, and it should be recognized that as VR permeates throughout our society, it will undoubtedly influence numerous aspects of our being. In light of this, it is important to establish what VR technology might influence, and how we can prepare our ethical paradigms to adjust to these influences. Kenwright suggests that VR has potentially 'limitless' positive applications, but that some of the key potential ethical implications might include an influence on our cognition and our physiology (Kenwright, 2018). Perhaps how this chapter began is how it should end: as Richard Feynman suggests, we need to be wary about advances in both technology and science.

3 SPEAKER AND HEADPHONE-BASED IMMERSIVE AUDIO SYSTEMS

"Music is the literature of the heart; it commences where speech ends."

— Alphonse de Lamartine

This chapter will solidify and expand on some of the auditory concepts and technological systems that apply to this project. The chapter is divided into five sections. As it is important that the reader has some understanding of what spatial audio is, and how humans are able to discern the location of sound sources, the first section will provide some context to this discussion. The second section will introduce the notion of immersive audio, and the concepts of Object-Based Audio (OBA) and Channel-Based Audio (CBA). This section will conclude by investigating the adoption of immersive audio standards, and some of the initiatives being used to propel this process in the context of digital media and VR. The third section will present the mechanisms and theory of various rendering algorithms and, in so doing, reveal their potential use-cases and how each of the algorithms compare to one another. This section will focus on spatial audio for speaker-based systems, whilst the fourth section will offer an inquiry into how spatial audio is achieved for headphones. The fifth and final section of the chapter will provide a brief overview of two contemporary speaker-based immersive audio systems, and aims to outline the theory behind and background of the immersive audio system that has been utilized in this research project, ImmerGo.

3.1 AUDIO CONCEPTS

As mentioned above, this section will provide readers with a theoretical framework that should explain some of the fundamental concepts pertaining to how human hearing works, and how this knowledge has been leveraged in the field of audio engineering to provide 3D audio. The human hearing anatomy functions will first be examined, followed by definitions of core auditory concepts.

3.1.1 How we Perceive Sound

Sound is an interesting concept, as it has traditionally been defined according to the medium in which it exists, and not according to the object that it originates from (Pasnau, 1999). In the context of this research project, sound will be defined as a longitudinal pressure wave that is produced from a vibrating object. As such, sound waves are essentially vibrations that carry energy passing through a medium. A sound wave will propagate through a medium with certain properties, such as amplitude, intensity, frequency, wavelength, speed, and direction, and the source of a sound can therefore be defined as the position from which its sound wave(s) propagates. The position of a sound source is quantified using three different parameters, namely the azimuthal angle, the polar angle or angle of elevation, and the distance between the source and the listener, as indicated in Figure 22 (Ting, et al., 2021).

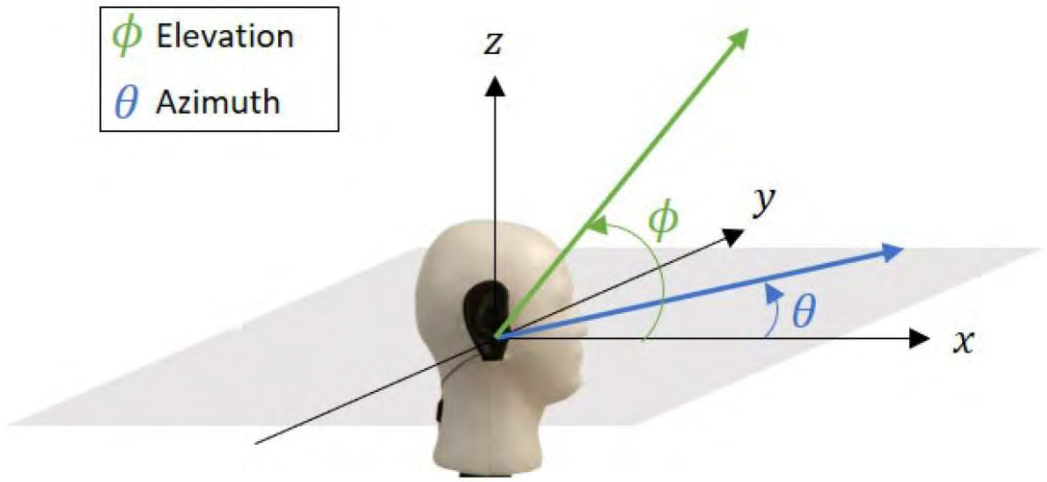

Figure 22: A diagram indicating the azimuthal angle and the angle of elevation (Ting, et al., 2021).

Like vision, hearing is a sense that aids a person's perception of the environment in which they find themselves. Similarly, just as vision is a perception of how light affects an environment, so it is with hearing and sound: hearing is the perception of vibration as sound (Alberti, 2001). The function of the ear is to translate the physical vibrations in the environment into nervous impulses that convey information to the brain. Alberti provides a useful analogy that compares the ear to a microphone. A microphone transduces vibration into an electric signal, while the ear transduces the vibration into a nervous impulse which is then processed by the brain (Alberti, 2001). The human hearing anatomy consists of specialized organs that include two ears and, crucially, a Cochlea for each ear. The Cochlea is the 'Sensory organ' of the system and is responsible for transducing the vibrations conducted to it by the ear. The exact mechanisms behind this process are beyond the scope of this work. However, Figure 23 provides a diagram of this auditory system, and the organs of which it is comprised:

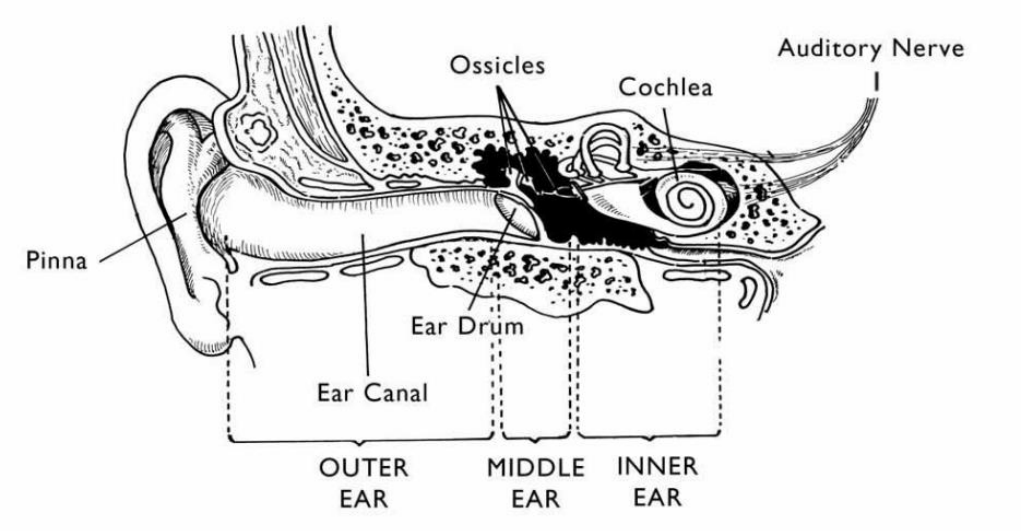

Figure 23: A diagram that indicates the various components of the human auditory system *(World Health Organization, 2006)*.

3.1.2 Localizing Sound

The previous section introduced the apparatus of the human anatomy that is involved in the perception of sound. It is now useful to introduce some of the qualities that sound possesses that enable us to interpret the location of objects emitting sound. The crucial aspect of our hearing is that we have two ears, and therefore use the differences in perception of each to discern the locality of a sound. This function can be considered as the sonic equivalent to visual parallax (which is detailed in chapter two). In this context, 'auditory parallax' refers to the ability to combine the information about an object provided by its sound arriving at each ear, to enable a more accurate perception of where the object is in three dimensions (Garner, 2017; Yost, 2018). Indeed, Genzel and colleagues provide evidence that auditory motion parallax is used by humans to evaluate the relative depths of two sound sources (Genzel, et al., 2018).

Terms that are often used in audio engineering are monaural and binaural auditory cues. Monaural cues refer to the qualities of sound that enable localization due to frequency pattern matching and pertain to the spectral information contained and exhibited by the sound (Hofman, et al., 1998). The monaural (spectral) cues can best be explained by recognizing that the shape of the ear is such that the "amplitude and frequency response changes, depending on where the sound source is located on the azimuth plane" (Ting, et al., 2021). Indeed, the pinna (see Figure *23*) is a natural filter that can attenuate certain frequency ranges and thus plays a significant role in enabling the auditory system to localize a sound source's distance and direction from a person. The pinna exhibits a complex asymmetrical structure that emphasizes specific resonances that depend on the incident sound source's location. It is suggested that these distinct resonances combine "direction-specific patterns into the frequency response of the ears" (Ting, et al., 2021). The resulting signal is then interpreted by our auditory system to provide directionality to the sound being perceived. This phenomenon and others, like energy absorption from the head and shoulders, are all monaural cues, which form the backbone of Head-Related Impulse Responses (HRIRs) and therefore Head-Related Transfer Functions (HRTFs) theory and implementation. Importantly, each person exhibits a unique set of HRTFs, and the determination of which, is discussed in the proceeding HRTF section.

Binaural cues refer to the information gained due to the difference in perception of a sound from each ear (i.e., auditory parallax). In principle, the disparity between what each ear perceives can be classified by a difference in intensity and in time (Cheng & Wakefield, 1999). The binaural cues can be further quantified into two key concepts, namely the *Interaural time difference* (ITD) and the *interaural intensity difference* (IID). The ITD can be defined as "the difference in arrival time of a sound's wavefront reaching each ear". Accordingly, the IID can be defined as the "difference in amplitude (intensity) between the incident signal reaching each ear" (Cheng & Wakefield, 1999). Figure 24 provides a diagrammatic representation of these two concepts. In the field of auditory spatial awareness, which is a term that denotes attaining spatial awareness from auditory stimulation, the term 'auditory parallax', which was introduced earlier, can more formally be described as an offshoot of duplex theory. The *duplex theory of localization* refers to the two mechanisms of ITD and IID combining to facilitate the localization of a sound source in a surrounding environment (Letowski & Letowski, 2012). However, the reader should be aware that another mechanism called the *interaural phase difference* (IPD) is used when discerning "continuous pure tones and harmonic complexes". IPD replaces ITD when these sounds are perceived, as they do not exhibit a clear interval in time (Letowski & Letowski, 2012). Indeed, IPD represents a

more subtle binaural cue that affects the perception of sound, but it is not documented as being as important as both ITD and IID.

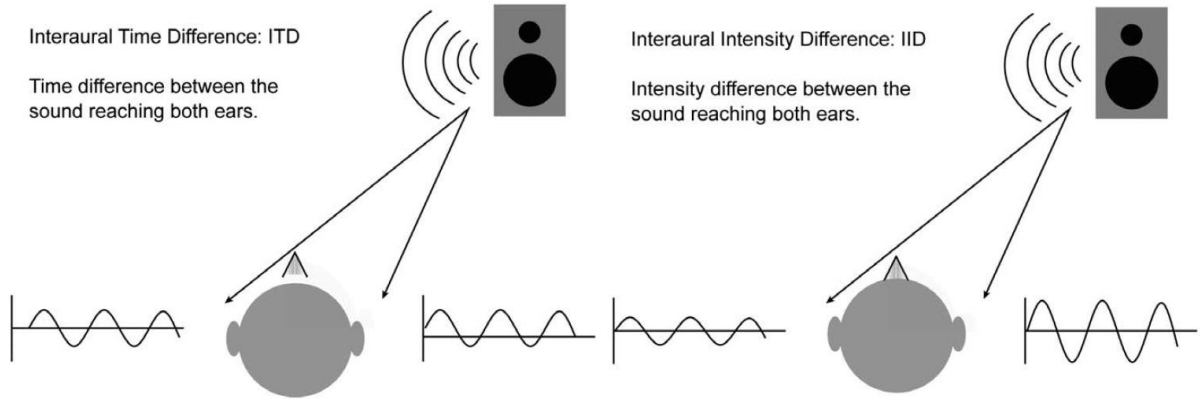

Figure 24: A diagram that shows how the two binaural cues (ITD and IID) function (Sinclair, 2020).

While ITDs and IIDs provide a comprehensive means by which humans can interpret the location of objects in their surroundings, there are some well-documented limitations to these cues. The phenomenon of front-back ambiguity indicates that, in certain scenarios, the cues are the same from sound sources directly in front of or behind the listener. In addition, the *cone of confusion*, which manifests due to "spatial ambiguity caused by left-right head symmetry" (Sinclair, 2020), is another limitation of binaural cues (see Figure 25). It has, however, been suggested that the cone of confusion problem is solved by a listener rotating their head (Liang, et al., 2015). It is generally agreed, then, that binaural cues provide effective horizontal localization, but are of limited use for vertical and front-back localization (Letowski & Letowski, 2012).

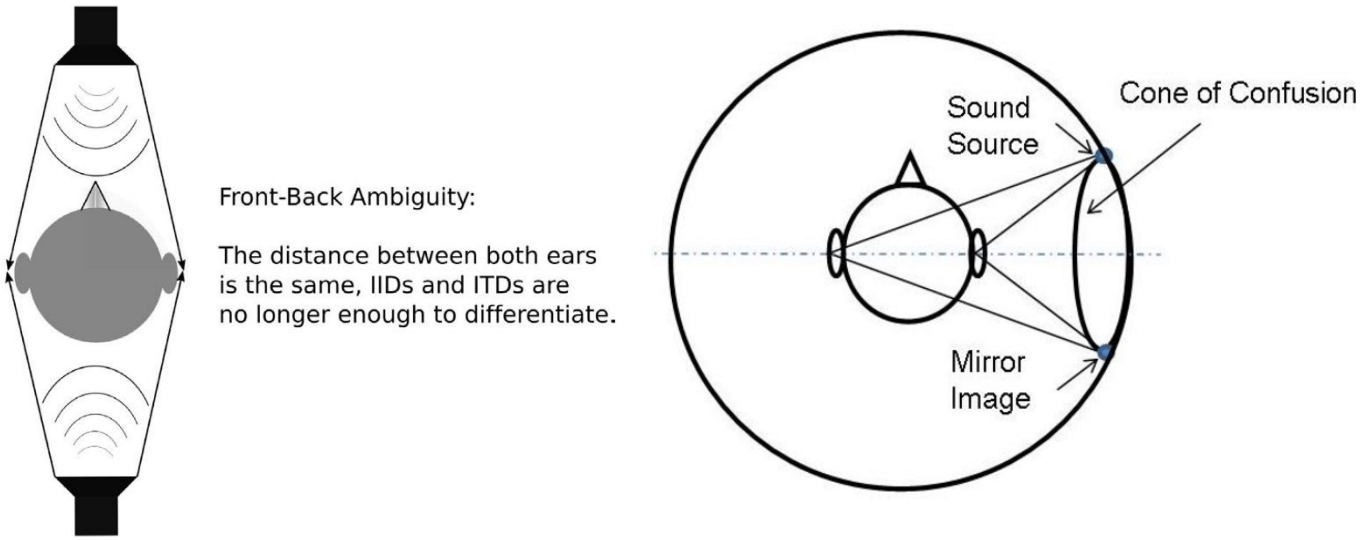

Figure 25: The left diagram depicts front-back ambiguity, while the right diagram provides an indication of how the 'cone of confusion' operates.

This section has provided the reader with a more comprehensive understanding of the core concepts that underpin auditory spatial awareness, namely concepts pertaining to monaural and binaural cues,

whereas Chapter four presents a more contextual understanding of these concepts from the perspective of implementing immersive audio into a VR application.

3.1.3 Spatial Audio

In the context of this thesis, the term spatial audio can be described as "the ability of an audio system to position sound objects within a specific and clearly localizable point in three-dimensional space" (Garner, 2017). Consequently, spatial audio can be viewed as an all-encompassing term that includes any audio processing method that facilitates the localization or position of a sound in a virtual space. This includes stereo-panning, surround sound, and 3D audio/immersive audio rendering techniques. To better appreciate the distinction between spatial and immersive audio, immersive audio can be said to utilize spatial audio to contribute to end users' sense of immersion when engaging with some or other media. This might, however, seem an unnecessary clarification, as the two terms are used interchangeably in various contexts. The following subsections will detail certain audio processing methods, and subsequently a brief historical overview of spatial audio will be provided.

3.1.3.1 Monophonic and Stereophonic Sound

These classes of spatial audio were the first to be developed. Monophonic audio refers to audio having simply one channel of sound, and the first recording systems were therefore monophonic devices (Rumsey, 2012). A channel can be defined as "a representation of sound coming from or going to a single point. A single microphone can produce one channel of audio, and a single speaker can accept one channel of audio, for example".[21] In contrast to monophonic audio, stereophonic audio implies that the audio track has two channels of sound, generally for a pair of left and right speakers. The adjustment of each audio signal results in the ability to pan, or move, audio sources on a horizontal plane between the two speakers, provided that the listener is directly in front of the speaker pair. Although it seems intuitive to think that more audio channels or multichannel audio would result in a greater ability to localize virtual sound sources, stereophonic audio remains the most popular type of spatial audio. It is argued that higher accuracy and quality of sound can be achieved with stereophonic sound than surround sound because the left and right speakers in a stereophonic system more closely resemble how humans perceive sound with two ears (Kraemer, 2001). This has resulted in a scenario in which sound designers and producers have 'perfected' stereo recording and production. However, the research that posits this idea is somewhat outdated when one considers the rise of immersive audio, and how digital media better leverages the benefits of a higher channel count than previously – that is, through correct speaker configurations and different algorithms enabling spatial audio. Additionally, the use-cases for better spatialization are much more apparent in contemporary technologies such as VR and the audio used in digital games.

3.1.3.2 Surround Sound and 3D Audio

Surround sound expands on the principles of stereophonic audio, and essentially increases the number of audio output channels. Of course, this rests on the assumption that, with a greater number of output channels, a more accurate representation of a recording can be reproduced. 5.1. surround sound systems utilize six output channels for audio playback (see Figure 26). The six channels can be grouped as follows: a front left and right stereo pair of speakers (L, R), a center speaker (C) placed in-between the pair, a surround left and right stereo pair (LS, RS), and finally a low frequency channel, which is known as the Low Frequency Effects (LFE) channel and is reserved to amplify low frequencies. In the channel description format, the LFE channel is represented by the ".1". In bass management, the ".1" channel is

[21] https://www.wildlifeacoustics.com/resources/faqs/what-is-an-audio-channel

reserved for the subwoofer (which is a loudspeaker designed to amplify low frequencies) which receives the low-end spectrum of frequencies from all other channels in the configuration (Rumsey, 2012). In accordance, a 7.1. surround sound system utilizes eight output channels (it includes a stereo pair of speakers for the back left and right of the array). Of course, there are numerous alternative surround sound configurations that increase the number of available output channels.

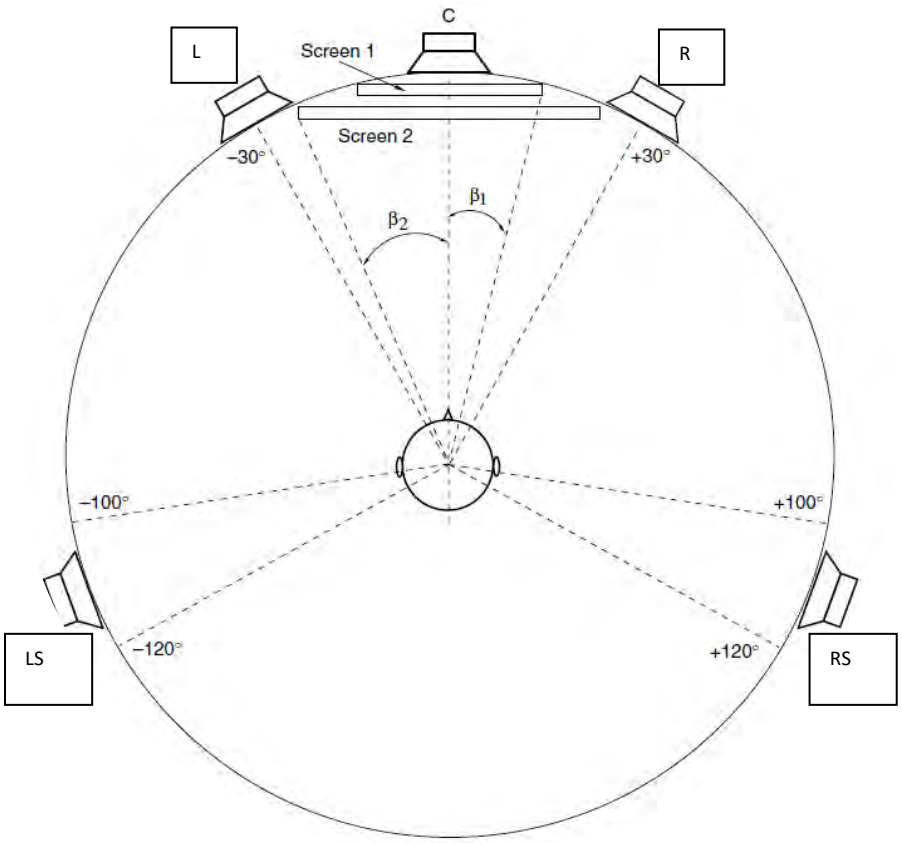

Figure 26: This diagram represents a 5.1 surround sound system that excludes the subwoofer. The layout conforms to the ITU-R BS.7 75 standard. This diagram was originally presented in 'Spatial Audio' (Rumsey, 2012).

Traditional surround sound systems can pan audio on a horizontal axis. A listener should be able to discern whether a sound is to their left or right, and in front of or behind them. However, a listener would struggle to locate a sound above or below them. Certain surround sound systems address this by adding stereo pairs of speakers above and below the listener, or by using a variety of mixing techniques to create an illusion of height, although there are certain limitations to these techniques. However, 3D audio attempts to address this issue by implementing a height layer. This type of spatial audio adds another layer of the depth, as it has the ability to position audio in three dimensions, thereby enabling the locality of a sound source to be placed across both horizontal and vertical axes (Garner, 2017). The ability is generally facilitated by employing two layers (an upper and lower set of speakers) in a speaker configuration to give a component representation of 7.1.4 where the ".4" denotes four speakers placed in the second layer above. As one might guess, 3D audio is highly sought after in the contemporary digital media paradigm, as the immersive qualities of audio in such media play a crucial role in both functionality and in contributing to overall perceived immersion. When this is achieved in a speaker-based system, it entails increasing the number of speakers in the system (from that of traditional

surround sound systems). Indeed, the system utilized in the experimentation of this research could be considered a 3D audio system with an 8.1.7 speaker configuration.

3.1.4 Spatial Audio: A history

As producers, researchers, and audio engineers have had access to increasingly more precise and innovative technology, they have continuously tried to improve audio realism in media by increasing the output channel count of their systems, or by employing more accurate rendering techniques (see subsequent section). This has only intensified with the current rise of new forms of media and technology. Virtual Reality, Augmented Reality, Mixed Reality and 360-degree video are all forms of media that require more realism than ever before (Sinclair, 2020). It might be difficult to argue that the audio component of these media is as important as the visual component. However, researchers agree that if either of these components is rendered inaccurately, a user's sense of disbelief is likely to be broken (Garner, 2017). By extension, traditional forms of spatializing audio are less effective when coupled with these new technologies, and this has given rise to a greater emphasis on audio technology and innovations. Surround sound systems do provide an accurate representation of sounds in a virtual scene, but 3D and therefore immersive audio are the next steps in enabling a more intuitive and believable experience. Garner posits that, during the hiatus of consumer VR through the 1990s and throughout the 2000s (see section 2.2 in previous chapter), spatial audio research mirrored this trend. Indeed, he suggests that the renewed interest in spatial audio can directly be attributed to the rise of new age media (Garner, 2017).

3.2 IMMERSIVE AUDIO

Immersive audio is synonymous with 3D audio and is concerned with being able to accurately reproduce an audio source or sources in three dimensions. The idea is not novel, and specific rendering algorithms were proposed decades before the renewed contemporary interest. Before analyzing a few of these algorithms, it is worthwhile to reiterate why immersive audio is important, and what areas of media technology it might influence. It is obvious that a more realistic representation of an audio (source) recording in space would profit multitrack music recording and reproduction, or that a film would benefit from being able to provide spatial audio more realistically to objects moving about a viewer. However, the advent of new technologies whose applications are still being developed seems to be where the real value of immersive audio lies. For example, Sinclair argues that, from the perspective of sound design for digital games (and by extension VR/AR/XR applications), a fully immersive audio environment can elevate and complement the corresponding visuals (Sinclair, 2020). Indeed, due to the horizontal and vertical localization capabilities of an immersive audio system, it has been suggested that, when used correctly, both in design and set-up, these systems generate a greater sense of immersion than surround sound systems (Sun, 2019).

3.2.1 Channel- and Object-Based Audio

One of the important developments from surround sound to immersive sound-based media, has been the categorization of Object-Based Audio (OBA), and Channel-Based Audio (CBA) immersive audio systems. These categorizations denote the difference between traditional surround sound and the more modern metadata driven approach. The following two sections will provide the reader with a clear understanding of how each type of system functions and, in so doing, compare the two approaches.

3.2.1.1 Channel-Based Audio

Traditional surround sound (2.1/5.1/7.1) formats are CBA systems (Sun, 2019). CBA involves an implementation of spatial audio whereby the task of the audio designer is to provide a set of audio signals. Each audio signal in this set is engineered to be played back over a specific loudspeaker in a

known predetermined position (see Figure 26 for a pre-defined 5.1 speaker layout) relative to the listener (Herre, et al., 2015). In these systems, sound production achieves spatialization of a virtual source by mixing the track according to the channel-based or surround sound format, and the audio designer is required to author these mixes in a speaker setup which is conforms to that of the playback format, i.e., the amplitudes of output channels are decided according to where they wish to position the source in the audio scape at any given time. In a multitrack score or soundtrack, this process needs to be precisely performed for each of the tracks that appear in the final mix (Herre, et al., 2015). In the context of films, this final mix is then encoded into the video track of the film. This implies that the audio production needs to be performed for each different CBA configuration, and results in a set of audio mixes that apply to each configuration (i.e., stereo, 2.1, 5.1 mixes). Of course, this is a costly and inefficient process, as sound producers need to accommodate all the popular configurations that consumers possess (Sun, 2019). There are, however, ways to automate upmixing and downmixing of a track to accommodate differing speaker configurations, but these techniques often result in a degradation of the quality of the audio. Herre and associates argue that the primary reason for CBA's persistence is that the production process is very well established. However, they do also suggest that the format incompatibility problem is particularly relevant in terms of the rise of new age media, and that OBA immersive audio systems would remove this issue (Herre, et al., 2015). Figure 27 clarifies how CBA content is captured, produced, mixed, and eventually reproduced.

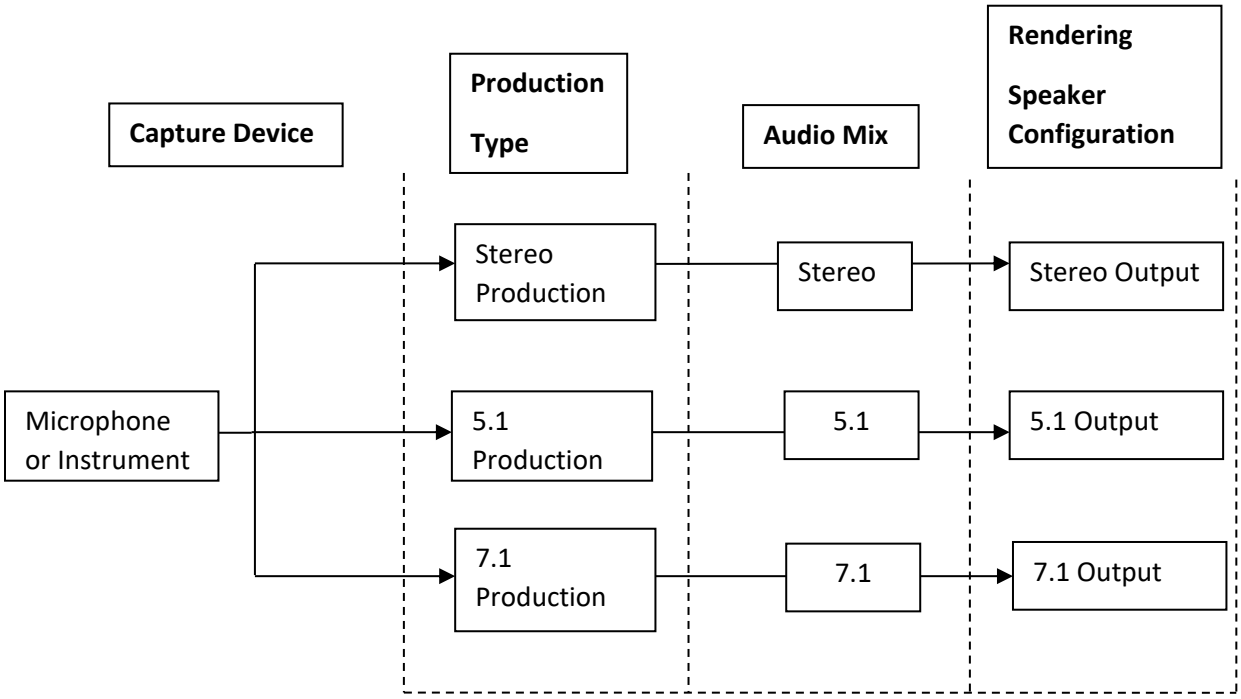

Figure 27: A diagrammatic representation of the workflow of a Channel-Based immersive audio system.

3.2.1.2 Object-Based Audio

Many contemporary immersive audio systems are object-based metadata driven systems. The most popular of these available commercially is Dolby Atmos (see section 3.5). At the core of OBA is a model that describes a virtual sound scene, and a renderer for the output system (see Figure 28). The virtual sound scene contains a group of audio objects (each object could represent an instrument in an

ensemble, or a character in a film; each object represents an audio event for the media). The audio for each of these objects is streamed to the renderer.

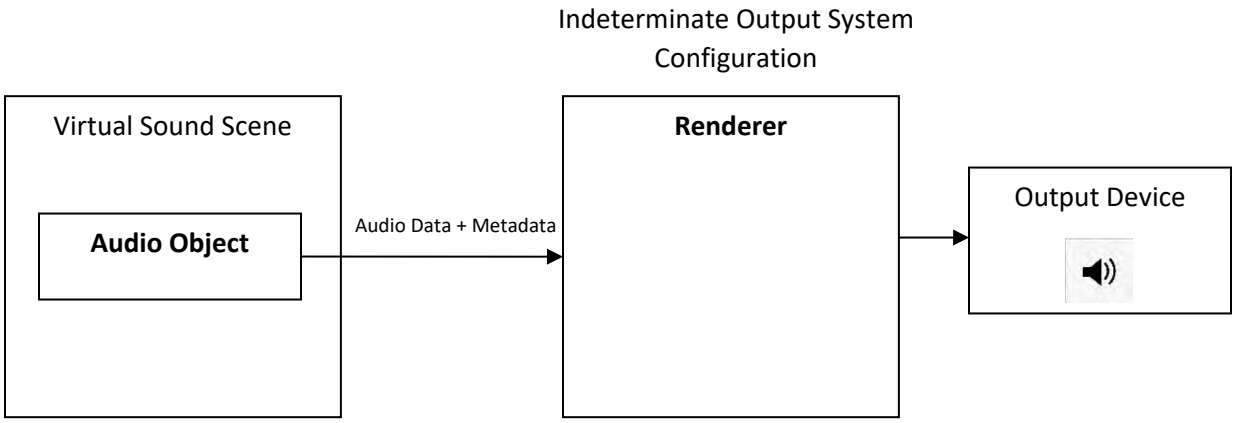

Figure 28: A simple diagram indicating the two primary components of an OBA immersive system

In addition to the audio samples, metadata for each object is also provided to the renderer. This metadata contains information such as the 3D coordinates and volume of the object in the scene (see Figure 29).

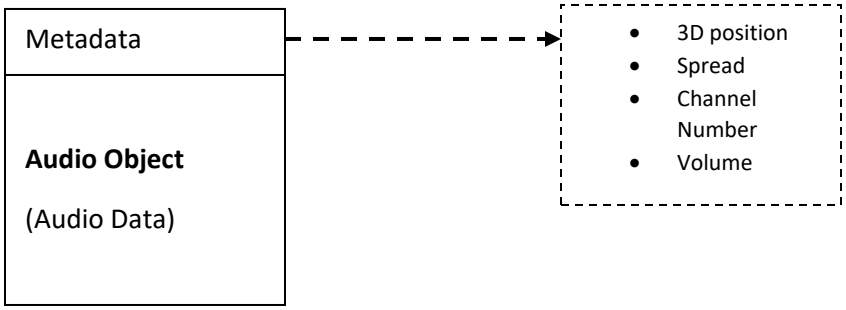

Figure 29: An example of an Audio Object and possible information that the metadata for this object might contain.

The renderer is involved with the reproduction of the audio based on the end user's output system, which could be headphones, stereo, or surround sound, or even a custom loudspeaker configuration. As such, the renderer interprets the virtual sound scene data and determines how the audio should output to each output device channel (Coleman, et al., 2018).

The benefits of OBA are interesting. Firstly, the actual placement of the object is fluid; the object's position can change over time and these changes are reflected in the object's metadata (Herre, et al., 2015). An OBA mix is defined during the production process, however, it is the job of the renderer to interpret the metadata and reproduce the audio based on the output system's loudspeaker configuration, which contrasts the processes of CBA. This is important, as it means that loudspeaker configurations are not considered during the production process (Sun, 2019). Instead, the specific loudspeaker configuration where the audio is being reproduced determines the appropriate mix levels for that configuration. This therefore enables a flexibility of speaker configurations, which until now have been defined by standards and fixed layouts (i.e., Stereo or 5.1. configurations). Of course, this

stands in stark contrast to CBA, which requires a particular mix for every standard for speaker layout (Figure 30 below provides a workflow for OBA).

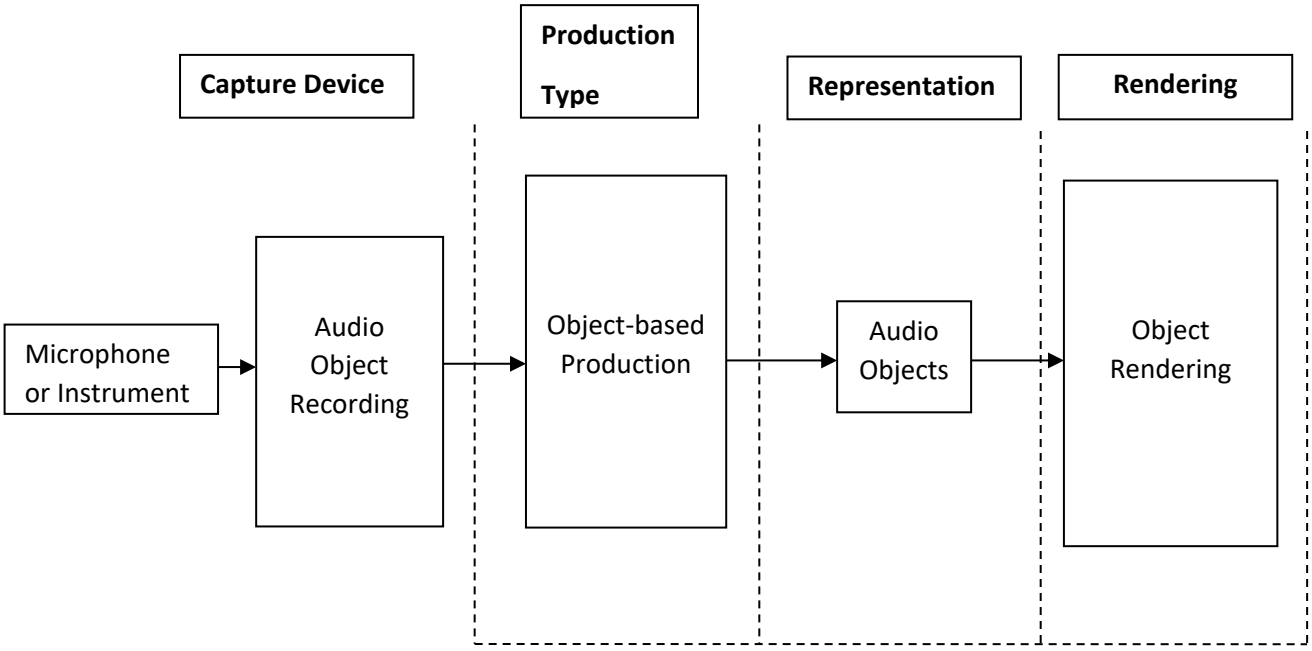

Figure 30: A representation of the workflow of an Object-Based immersive audio system.

Another particularly beneficial element of OBA is its ability to enable many concurrent audio objects at a particular instant in the media, which can be adjusted as required at the playback (reproduction) stage. This element of OBA is enabled by abstracting the audio into multiple sound objects that are typically rendered for a system with a much smaller number of audio output channels. By contrast, CBA can only render a small set of audio sources believably. It is therefore constrained by the audio producer having to manually "mix" and adjust speaker amplitudes for each of the active audio events in the scene (Herre, et al., 2015).

Panning techniques and tools used by audio designers in the CBA mixing process are designed to manipulate the amplitude and phase of a signal being output to a standard configuration of speakers. If an end-user's playback system does not match the mixing format, then audio source localization is compromised. Another limitation of CBA and stereophonic reproduction techniques is the general requirement that listeners inhabit a "sweet-spot" for optimal and realistic audio reproduction (see Figure 26). While this is also an issue for OBA rendering techniques, algorithmic solutions counteract the problem (Herre, et al., 2015). Other benefits of OBA include the ability of OBA media to be highly scalable, and adjustable according to a user's needs, i.e., an audio object representing dialogue can be identified and boosted should a user wish to hear voices more clearly. These adjustments are possible because the rendering of the audio is happening on the user's (immersive) audio system, thereby enabling the manipulation of metadata that is just not feasible in CBA systems.

3.2.2 Immersive Audio Format and Standardization

The calls for a standardization of OBA have existed for over a decade (Breebaart, et al., 2008) and, of course, the process of audio format standardization encompasses the various aspects of audio production, distribution, and reproduction. The Moving Pictures Expert Group (MPEG) has proposed the

MPEG-H standard, though alternative standards have also been put forward. However, it seems that MPEG-H is the standard receiving most endorsement. Indeed, a desirable format is one that benefits from smaller data size, and low computational overheads when interpreting and rendering the audio on reproduction. MPEG-H meets these requirements, and can therefore be introduced as a standard "for universal and efficient coded representation and rendering of high-quality spatial audio" (Herre, et al., 2015).

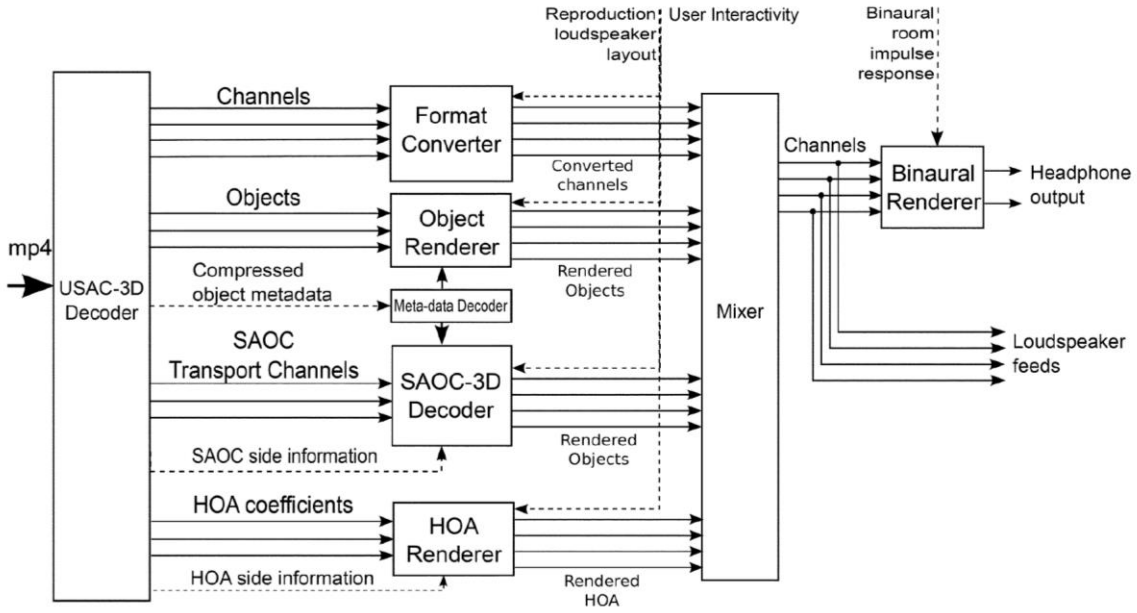

Figure 31: A block diagram of the MPEG-H decoder, courtesy of Herre and colleagues (Herre, et al., 2015).

MPEG-H is an audio coding standard that exists in three components, the USAC-3D decode, various renderers, and a mixer (see Figure 31 above). Essentially, the MPEG-H decoder utilizes a variety of protocols to decode, render, and output OBA. The functions of each of these protocols are beyond the scope of this research, but the most important of these protocols is the Spatial Audio Object Coding (SAOC) technique. SAOC encompasses "flexible, user-controllable rendering of multiple audio objects based on transmission of a mono or stereo downmix of the object signals" (Breebaart, et al., 2008).

The SMPTE ST 2098[22] standard provides a framework by which immersive audio formats and OBA should be employed within the film industry. It is suggested this standard is heavily influenced by Dolby Atmos (Sun, 2019). Similarly, project ORPHEUS is a broadcasting initiative which has proposed "a full set of tools and applications for all stages of the complete broadcast workflow".[23] These industrial entities will play a crucial role in facilitating the transition from CBA to OBA moving forwards.

3.3 Speaker-Based Immersive Audio Rendering Algorithms

Having defined audio concepts and provided the context for spatial and immersive audio implementations, some of the more technical aspects of the technology will now be reviewed. In particular, a few of the more popular and widespread immersive audio rendering algorithms will be examined, some of which are included in the loudspeaker rendering system used in this research.

[22] https://www.smpte.org/webcast/ins-and-outs-st-2098-2-immersive-audio-bitstream
[23] https://www.orpheus-audio.eu/

An audio rendering algorithm is used by an immersive sound system to accurately localize and playback an audio source in a sound field. The algorithm uses the positional data of an audio source and then determines the output for each of the speaker endpoints in the speaker array to provide accurate spatialization (Sun, 2019). Each algorithm is suitable for different needs and functions. The following sections provide a concise explanation of how each of the algorithms function and, in so doing, aim to establish a few of the differences between them. It is useful to recognize that the development of these algorithms can be attributed to the rise in interest in facilitating more accurate means of representing spatial audio, both domestically and industrially, during the 1970s and onwards. (Geier, et al., 2010). As the reader will recognize, much of the research was thus formulated well before the contemporary phase of digital media.

As the mathematical theory behind most of these algorithms is comprehensive, and the domain of this project does not lie in the actual implementation of these rendering algorithms, overviews and abstractions of each algorithm are provided below.

3.3.1 Vector-Base Amplitude Panning (VBAP)

VBAP is a popular amplitude rendering algorithm that has been successful in rendering spatial audio. It was not used for this project, but it is worthwhile reviewing its conceptual foundations. It has been implemented in commercially available renderers such as DTS:X[24] and Auro3D.[25] VBAP requires a "regular" loudspeaker arrangement to achieve accurate panning. This means that all speakers need to be equidistant from a pre-conceived listening position, i.e., on the surface of a sphere (Pulkki, 1997). A limitation of VBAP is that the algorithm also assumes the position of the listener is "known, fixed and restricted to a small area". This is because the algorithm is not intended for the placement of sources inside the speaker array, but atop the (generally hemispherical) surface of the array (Kostadinov, et al., 2010).

VBAP makes use of vector and vector-base algebra. Essentially, the algorithm requires the division of the speaker array into discrete sets. These sets should comprise two speakers for two-dimensional VBAP and three speakers for three-dimensional VBAP (see Figure 32 for a two-dimensional representation and Figure 33 for a three-dimensional representation). In the two-dimensional diagram, L_{51}, L_{12}, L_{23}, L_{34}, and L_{45} all represent vector base pairs. The restrictions on these vector-base sets are that the source signal can be applied to any one of the sets, but only one of the sets can be active at any one time per virtual source being panned. Additionally, each speaker can only belong to two pairs (Pulkki, 1997). Pulkki recommends that, in two-dimensional VBAP configurations, speaker pairs consist of speakers that are adjacent to one another.

[24] https://dts.com/dtsx/
[25] https://www.auro-3d.com/

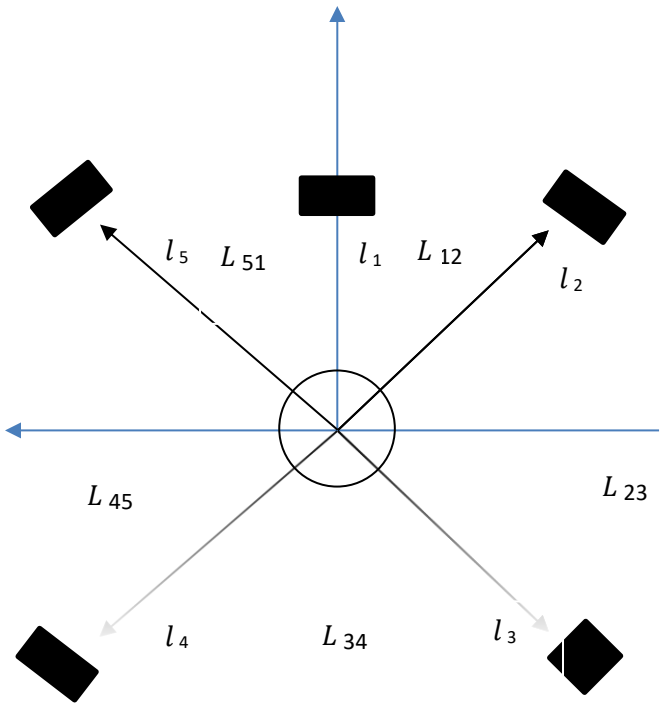

Figure 32: A diagram that shows a VBAP set-up for a two-dimensional five-channel system. I_1, I_2, I_3, I_4, and I_5 represent the vectors of the system, while L represents the composed pairs (Pulkki, 1997).

Figure 33 indicates four unit-length vectors. Vectors l_k, l_n, and l_m all originate from the listener position and point towards their respective loudspeakers, which therefore comprises a loudspeaker triplet. Vector p also originates from the listener position, but points towards the virtual sound source, which is positioned in-between the vector-base triplet l_n, l_m, and l_k.

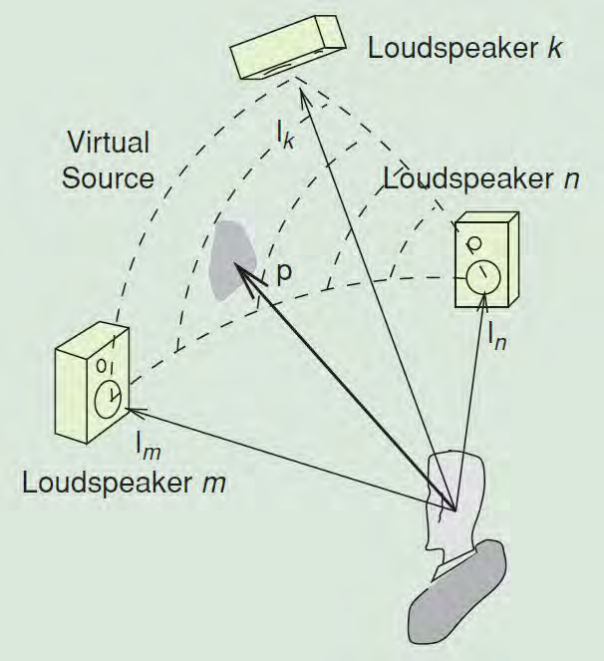

Figure 33: A diagram indicating a three-dimensional vector-base triplicate (Pulkki & Karjalainen, 2008).

The algorithm stipulates that differing vector-base sets should be activated to transmit audio based on the location of the virtual source in the sound field. Therefore, if one observes the figure above, the virtual source is located at a position that would activate the vector-base triplicate l_m, l_n, and l_k. Vector p can be expressed as a linear weighted sum of the other three vectors (Pulkki, 2000):

$$p = g_m l_m + g_n l_n + g_k l_k \quad (1)$$

In this linear weighted sum, g_k, g_n, and g_m represent the gain factors of each of the respective loudspeakers. Equation (1) should be viewed as a matrix calculation through which vector p can be determined. Each of the gain factors can be solved according to the following equation and matrix calculation:

$$g = p^T L_{mnk}^{-1} = [p_m \ p_n \ p_k] \begin{bmatrix} l_{mm} & l_{mn} & l_{mk} \\ l_{nm} & l_{nn} & l_{nk} \\ l_{km} & l_{kn} & l_{kk} \end{bmatrix} \quad (2)$$

where $g = [g_m \ g_n \ g_k]^T$ and $L_{mnk} = [g_m \ g_n \ g_k]$. Once the gain factors have been determined, they are applied to loudspeakers after normalization. The normalization function is indicated in the following equation (Pulkki, 1997):

$$g^{scaled} = \frac{\sqrt{C}g}{\sqrt{g_m^n + g_n^n + g_k^n}} \quad (3)$$

As with DBAP, once the gain factors have been normalized, the equal intensity panning principle will have been adhered to, and the resultant factor can be applied to the loudspeakers. C represents a scaling constant in the previous equation.

VBAP is a computationally efficient rendering algorithm that performs well when compared to other rendering algorithms. The low overheads and easy requirements for configuring VBAP speaker configurations make it a useful algorithm when compared to other spatial algorithms, despite its various limitations (Pulkki & Karjalainen, 2008). However, the algorithm is only efficient due to the integration of a lookup table, in which vector values are stored. This table enables the loudspeaker triplets to determine positive gain values quickly without having to perform the calculations indicated in the previous equations.

3.3.2 Distance-Based Amplitude Panning (DBAP)

Amplitude panning as a principle involves the transformation of the amplitude of a digital signal into amplitude differences at reproduced speaker endpoints.[26] It can also be defined as a technique "in which a monophonic audio channel is applied to all or a subset of loud-speakers with different gains" (Pulkki & Karjalainen, 2008). Indeed, Blumlein was the first person to suggest the theory of amplitude panning applied to a stereo pair in his 1931 patent (Pulkki, 2000).

[26] https://soundbridge.io/amplitude-panning/

DBAP is a rendering algorithm that is used to localize virtual audio sources in a non-fixed speaker array. The speaker arrangement can be flexible, which enables a large variation of arrays that are DBAP compatible. This aspect of the algorithm makes it not only reliable (in terms of localizing virtual audio sources), but also adaptable to differing requirements, soundscapes, and virtual environments. Another important detail of the algorithm is that it applies the principle of equal intensity panning, which means that a constant sound energy is maintained across all speakers at all times. Thus, the sum of the intensities of each speaker should remain constant through time. This is an extension of the principle of constant intensity stereo panning to multiple channels (Lossius, et al., 2009). An actual change in amplitude of a given speaker is determined by how far the virtual audio source is from the speaker. Another aspect of the rendering algorithm is that there is an assumption that all the speakers in the configuration are always active, and that the amplitude of a given speaker depends on its distance from the virtual sound source. The distance from the virtual source to a speaker is calculated using:

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \tag{4}$$

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2} \tag{5}$$

where the distance calculated (d) represents the distance between the virtual sound source and particular speaker in the arrangement. The values that populate the first formula pertain to the x and y values of both the virtual source $(x1, y1)$ and the speaker's position $(x2, y2)$ in a two-dimensional Cartesian coordinate system. If the speaker arrangement consists of both upper and lower layers, such that the virtual source can be panned in three dimensions, then the second formula would be employed and the z values would represent the third set of coordinates in the three-dimensional coordinate system (virtual source $(x1, y1, z1)$ and speaker position $(x2, y2, z2)$).

The DBAP algorithm therefore makes two assumptions. The first assumption, as mentioned, is that, irrespective of the virtual source's position, the intensity of the speaker array remains constant. This is implemented via the following equation:

$$I = \sum_{i=1}^{N} v_i^2 = 1 \tag{6}$$

where I is the overall intensity, N is the number of speakers in the array, and v_i is the amplitude of the of the ith loudspeaker.

The second assumption of the rendering algorithm is that all the speakers in the array should always remain active, and therefore that the amplitude of any one speaker in the array depends on the distance of the speaker to the virtual source:

$$v_i = \frac{k}{d_i^a} \tag{7}$$

where k is a coefficient that depends on the position of the virtual source and all the speakers in the array, and a is a coefficient calculated from the rolloff R.

$$a = \frac{R}{20log_{10}2} \tag{8}$$

A rolloff of R = 6 dB is equal to the inverse distance law for sound propagating in a free field (Lossius, et al., 2009). This rolloff value should be determined based on the environment in which the speaker array is being deployed. A combination of the equations that inform the two assumptions that the DBAP rendering algorithm makes (equations (6) and (7)), provided the constant k, is as follows:

$$k = \frac{1}{\sqrt{\sum_{i=1}^{N} \frac{1}{d_i^{2a}}}} \tag{9}$$

The amplitude value to be applied to speaker n is then:

$$v_n = \frac{1}{d_n^a \sqrt{\sum_{i=1}^{N} \frac{1}{d_i^{2a}}}} \tag{10}$$

As discussed by Devonport and Foss, listening tests indicated enhanced localization when a was set to 2, and shifting the value between 1 and 2 provided a varying spread of sound, with a value of 1 providing enhanced spread (Devonport & Foss, 2018).

The following diagram provides an illustration of how the DBAP algorithm would be employed to localize a virtual sound source with a four-speaker configuration. In the figure, the speaker number (S) denotes which calculated distance (d) pertains to it.

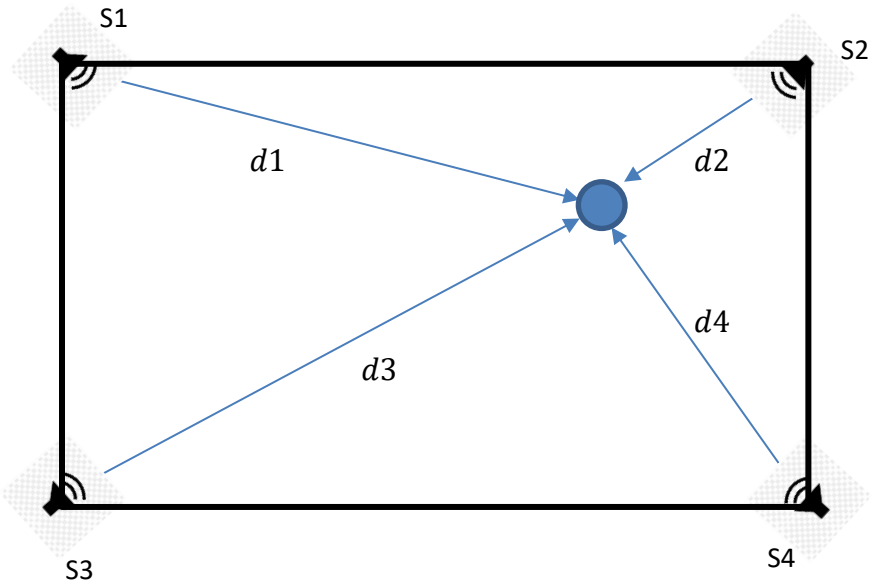

Figure 34: A diagram that indicates the layout of a four-speaker system and the distance between the virtual sound source (blue sphere) and each speaker (Kapralos, et al., 2015).

When reviewing literature that seeks to analyze how this algorithm performs against other rendering techniques, it has been argued that its "simplicity" and "flexibility" make it desirable (Kostadinov, et al., 2010). In addition, the listener does not need to inhabit a "sweet spot" (Lossius, et al., 2009). This obviously adds to its desirability for application in VR experiences that provide users with the ability to navigate a virtual environment physically (they are not bound to a particular location and can move within the speaker arrangement).

3.3.3 Wave field synthesis (WFS) and Ambisonics

Neither WFS nor Ambisonic implementations were used in this project. However, the following two sections will provide a brief overview of the two algorithms, and how they operate. Both aim to physically reconstruct sound fields when audio is reproduced (Daniel, et al., 2003).

3.3.3.1 WFS

Wave field synthesis is an immersive audio rendering technique that is unique in its approach to spatializing audio. It entails the synthesis of an artificial or virtual acoustic environment (Geier, et al., 2010). The technique was proposed almost thirty years ago (Berkhout, et al., 1993), and essentially employs a large array (with a two-dimensional, three-dimensional or hybrid configuration) of loudspeakers to create a simulated (virtual) auditory scene. A virtual audio source's wave field can be described by a mathematical model that predicts how a sound wave will propagate through a known two- or three-dimensional space from the sound's origin. The underlying theory and mathematics behind the rendering algorithm are relatively dense, and so the following paragraphs will summarize some of the key principles underpinning the algorithm. If the reader is interested in the theory, the supplied sources should provide them with a more in-depth view of how the algorithm can be implemented.

As alluded to above, the rendering algorithm is based on certain tenets of acoustic theory, a field that is a subsidiary to the field of fluid dynamics and pertains to the description of sound waves and how they propagate through a medium. The most important of these tenets is the Huygens-Fresnel principle,

which states that “every point reached by the acoustic wave front becomes a source of a new acoustic wave, and the envelope of these new waves is used for construction of the new wave front” (Piechowicz, 2011).

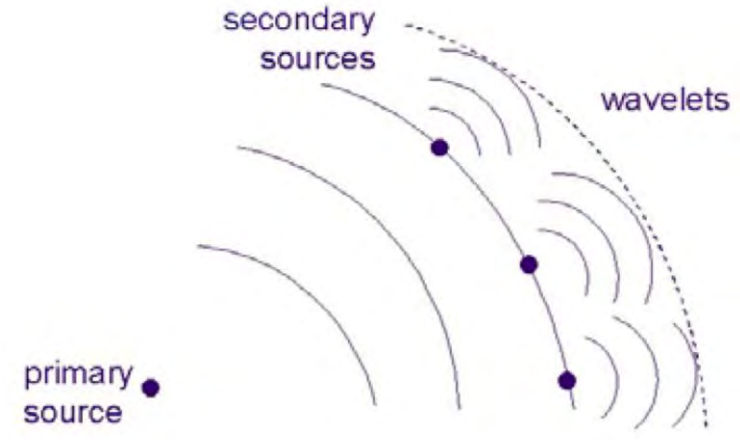

Figure 35: A diagram illustrating Huygen's Principle (Daniel, et al., 2003).

The Huygens principles is indicated in the figure above (see Figure 35). This principle has been mathematically expressed by the Kirchhoff-Helmholtz integral which states that, “if the sound pressure and particle velocity in any point at the surface of a source free volume is known, then the sound pressure at any point within this volume is determined.”[27] This integral has therefore been used in the development of WFS algorithms to solve for a sound field (with a known area) if the virtual audio source’s sound wave qualities are known (Berkhout, et al., 1993). If each loudspeaker’s exact location is established, it emits sound waves according to the ‘incident’ virtual audio source’s waves qualities (its velocity, directional gradient, and intensity being key). When the virtual sound waves arrive at a specific speaker, the speaker becomes what is known as a secondary source that propagates waves according to the Kirchhoff-Helmhotz integral (see both Figure 35 and Figure 36).

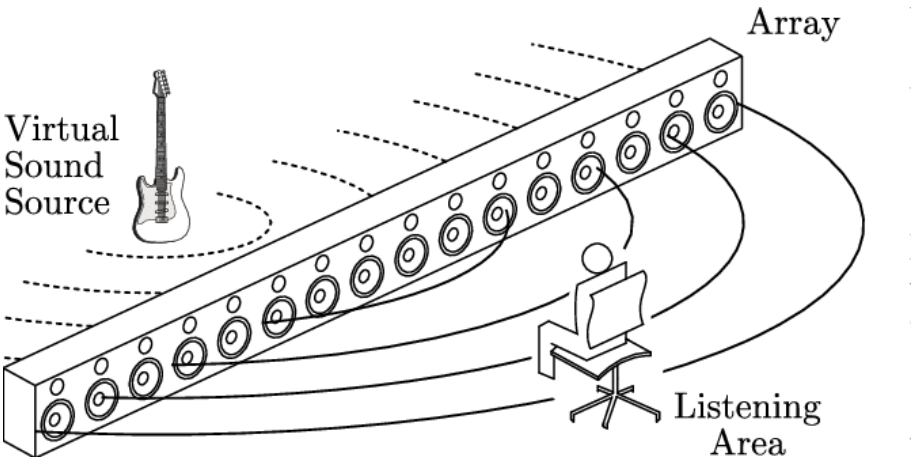

Figure 36: A diagram that illustrates how a virtual sources' propagating waves can be reproduced after reaching an array of speakers (Pueo, et al., 2011).

[27] http://www.syntheticwave.de/Wavefieldsynthesis.htm

There are a few issues with WFS that have been detailed. A continuous distribution of secondary point sources is required to exactly reproduce a wave field from a virtual source. This is of course theoretical, and unfeasible, as loudspeakers are discrete entities, and therefore any WFS system is a discrete approximation of a continuous distribution of secondary sources. As Ahrens and colleagues suggest, “a loudspeaker system can be regarded as an inhomogeneous boundary” (Ahrens, et al., 2008). These discrete WFS systems result in the manifestation of amplitude artifacts due to sound wave interference from the secondary sources. However, the closer the system is to the approximation of a continuous distribution system (i.e., the addition of secondary sources/speaker endpoints), the fewer the perceived artifacts. Figure 37 illustrates this interaction. It has been suggested that various modifications to the driving function, can also mitigate against these artifacts.

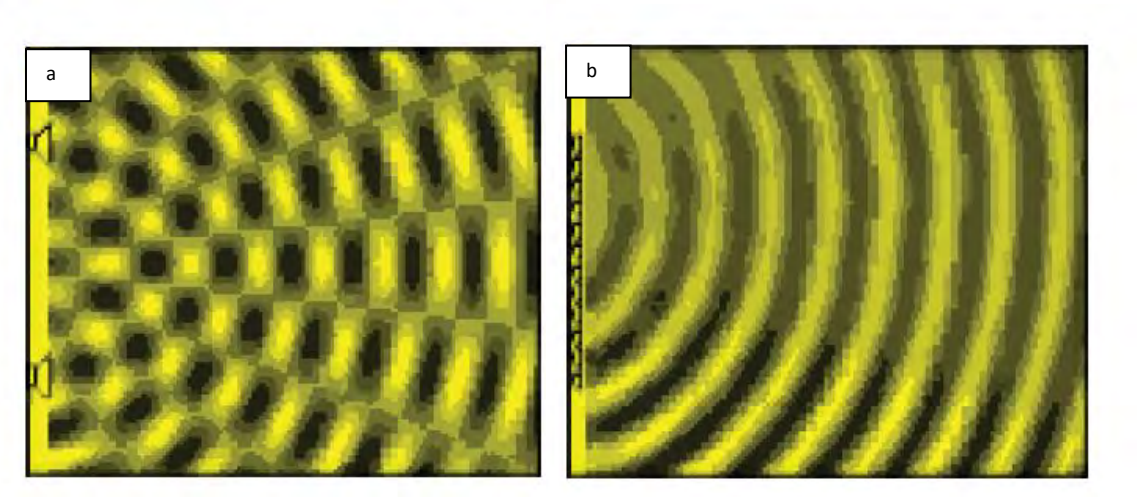

Figure 37: The left image indicates how sound waves interfere with one another in a linear speaker array with only two loudspeakers. In opposition, the right-hand image shows how an array with numerous loudspeakers better represents a continuous distribution of secondary sources, which reduces amplitude artifacts and wave interference (De Vries & Boone, 1999).

While the performance and accuracy of a WFS-based immersive audio system is desirable, its usefulness in the application of immersive audio for VR remains challenging, due to the overheads of implementing one of these systems. Indeed, a three-dimensional WFS system would be ideal for VR applications, but due to the unfeasibility of these set-ups, the algorithm will remain an unlikely candidate for this particular use-case, albeit that it does enable large listening areas.

3.3.3.2 Ambisonics

The principles that form the foundation of ambisonics were proposed in 1973 by Gerzon (Gerzon, 1973). Fundamentally, ambisonics enables the capture of a desired sound field according to the principles of cylindrical/spherical harmonics and the realization of this sound field via a loudspeaker configuration (Yao, et al., 2015). In sound engineering and recording, this implies the use of a tetrahedral arrangement of sub-cardioid microphones colloquially known as ‘soundfield’ microphones. The algorithm is a truly three-dimensional rendering algorithm, as the recordings represent a total 360 degree recording of a sound field with height. The output and rendering system then determines how this recording should be reproduced in three dimensions. Indeed, there are no preconceptions of what, or how the output system is configured. When discussing the stages of the algorithm, it is pertinent to realize that the algorithm consists of two major stages, the encoding and decoding of the sound field. The first stage refers to the recording and encoding of the sound signals present in a sound field, whilst the second

refers to the decoding of the signals encoded in the sound field, and thereby the reproduction of said sound field (see Figure 38).

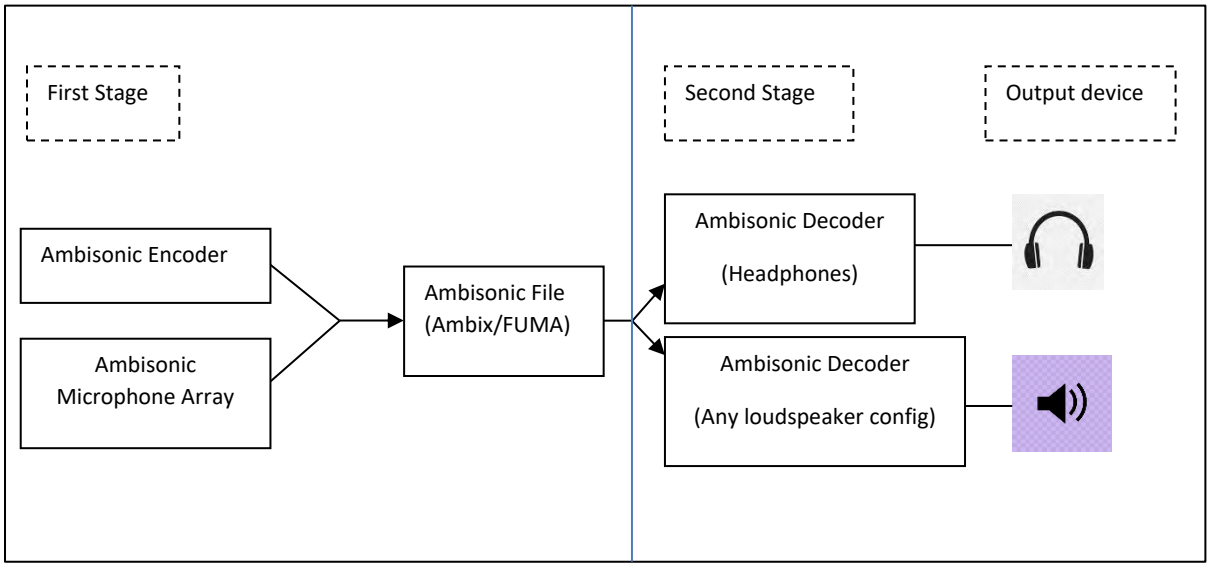

Figure 38: A diagram that indicates the two stages of ambisonics.

The ambisonic recordings create four individual sub-cardioid channels that output an A-format ambisonic file. This A-format can be considered as the raw recordings of the sound field and, as such, does not conform to a standard for interchange. Accordingly, B-format ambisonic recordings are A-format recordings that have been encoded into B-format to provide standardization and compatibility between ambisonic rendering.[28] A B-format ambisonic recording represents a 'total' directional sound field with the use of four audio signals, which are denoted W, Y, X, and Z (Furness, 1990). The Y signal represents the right to left signal, the X is the back to front signal, and the Z is the down to up signal. The W signal can be called the omnidirectional or pressure signal, which contains sound information from all directions with equal gain. Ambisonic implementations utilize notions of ITD and IID to determine signal strength of incident sound waves. This thereby enables the reproduction of a source according to the differences in amplitude and direction recorded from the composite microphone. Ambisonics does not comply with the OBA paradigm and can rather be categorized as scene-based audio (SBA). There are no metadata or individual audio objects in Ambisonics storage/transmission. As with OBA, the audio channels of an ambisonic recording do not dictate how the loudspeaker configuration needs to be set up for audio decoding and reproduction (see Figure 38).

The ambisonics reproduction and decoding process makes no assumptions about a user's output configuration. Indeed, ambisonics can be rendered to both loudspeaker arrangements and headphones (Frank, et al., 2015). The decoding (rendering) is performed using a suitable decoder matrix that should be determined by the output configuration. The implementation of an appropriate decoder for irregular speaker arrays can be developed using a suite of tools, such as 'The Ambisonic Decoder Toolbox'. This toolkit suggests ways to design decoders for hemispherical domes and multilevel rings, such as the speaker configuration utilized in this project (Heller & Benjamin, 2014). One of the formalized

[28] https://www.pro-tools-expert.com/production-expert-1/2019/12/10/ambeo-and-ambisonic-formats-a-bluffers-guide

approaches for decoding is the AllRAD[29] approach, which decodes ambisonics into a virtual loudspeaker arrangement (Frank, et al., 2015) . This approach results in a decoder matrix that adheres to mode-matching principles, which attempts "to perfectly reconstruct incident sound fields using a surrounding spherical arrangement of loudspeakers" (Zotter, et al., 2010). Interestingly, the implementation functions by mapping the virtual loudspeaker arrangement to real loudspeakers using VBAP (Frank, et al., 2015).

Ambisonic recording was seen as a favorable technique to employ in VR development during the 1990s (Garner, 2017). The recent adoption of .ambix B-format standards by Facebook and Unity[30] for their immersive audio platforms indicates the relative momentum that the rendering technique is acquiring. Garner posits that another reason ambisonics is gaining momentum in the context of VR applications is that it enables a large amount of head responsiveness when compared to traditional binaural techniques, which require static head positions for the effect to work (Garner, 2017). Although it is possible to implement ambisonic microphone recordings of sound sources within a virtual scene (through various plugins and via audio middleware), a more common approach to implementing ambisonics with virtual sources is to use a monophonic source signal and its three-dimensional location in a virtual scene to encode B-format versions of the signal. These are then decoded (rendered) on scene playback, .i.e., creating virtual microphone ambisonic recordings (Garner, 2017).

3.4 Head-Related Transfer Functions

The previous section has analyzed speaker-based rendering algorithms. The discussion will now turn to Head-Related Transfer Functions (HRTFs) and how they enable spatial audio for headphones. Indeed, the comparison between headphone-based and speaker-based spatial audio is a fundamental goal of the research performed in this thesis, and as a result, the theory and implementation of HRTFs requires substantial examination. HRTFs can be defined as a "an acoustic (frequency-dependent) transfer function between a point sound source in the free-field (without room information) and a defined position in the listener's ear canal" (Li & Peissig, 2020). Another applicable definition of an HRTF, which is derived from a more mathematical approach, would be that it is "the ratio of the (frequency) spectrum at the ear-drum to the spectrum at the sound source" (Al-Sheikh, et al., 2019).

This then presupposes a way in which a monophonic, non-localized source can be convolved with a particular pair of HRTFs (one for each ear) that denotes a specific location in a three-dimensional space. Convolution, mathematically, is a term used when one function is filtered with another to produce a resultant function (Sinclair, 2020).[31] Figure 39 indicates, diagrammatically, how a pair of HRTFs can be blended into a source signal and then applied to the left and right channel of a headset/headphones. An HRTF is derived from a HRIR, which is a time domain representation of the function, and is the product of a Fourier Transform applied to a recorded HRIR (Li & Peissig, 2020).

[29] https://plugins.iem.at/docs/allradecoder/
[30] https://www.pro-tools-expert.com/production-expert-1/2019/12/10/ambeo-and-ambisonic-formats-a-bluffers-guide
[31] https://mathworld.wolfram.com/Convolution.html

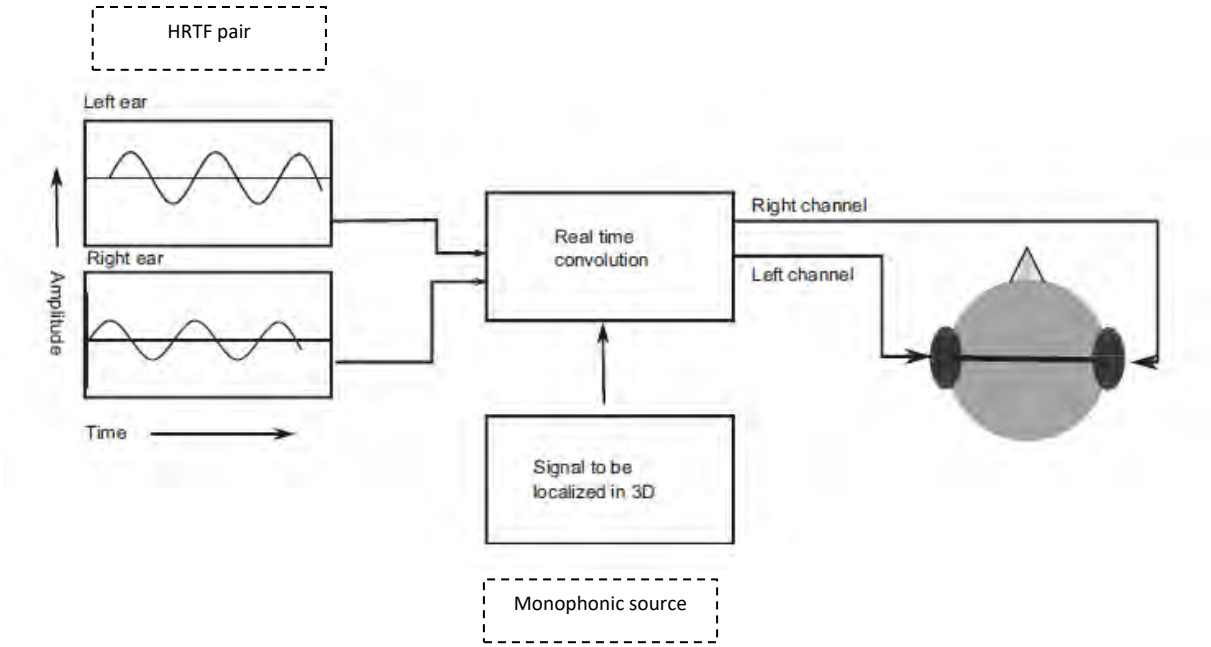

Figure 39: A diagram illustrating the process by which an HRTF pair is convoluted with a source signal to localize the source (Sinclair, 2020).

The nature of an HRTF is such that it contains all the necessary auditory cues: both spectral monaural cues and binaural (ITD and IID) cues. If one reviews section 3.1.2 on how the human auditory system localizes sound, then one might have realized that a critical issue with HRTF implementation is that each person possesses a unique set of HRTFs. This can be attributed to the fact that we all have a unique physical make-up, and that our ears, head, and shoulders (see Figure 40) all uniquely influence the spectral quality of a sound wave as it arrives in our ear canal (Letowski & Letowski, 2012). At present, some applications of HRTF-based spatial audio can choose to provide the ability for users to utilize their unique HRTFs or provide a set of generic HRTFs. Individualized HRTFs require significant calibration but provide more accurate virtual source localization than generic HRTFs (which essentially cater for the average).

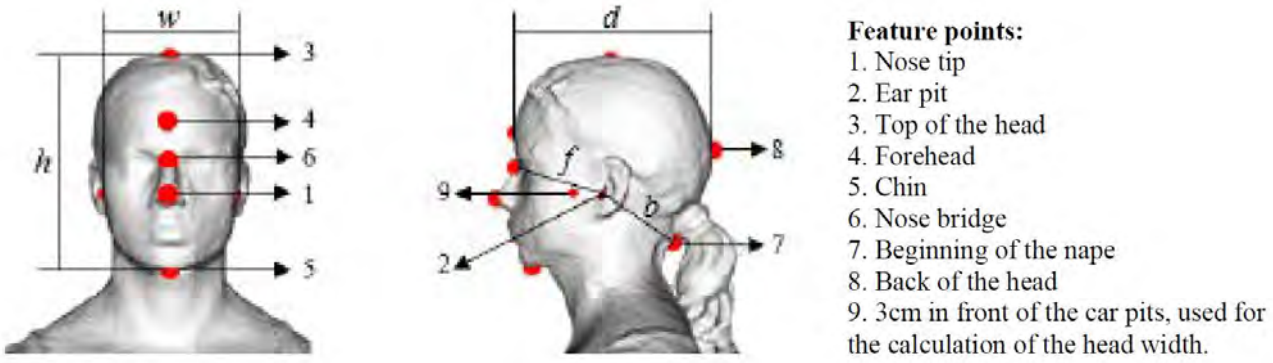

Figure 40: This figure indicates the various feature points of a person that play a role in determining their individualized HRTFs. $h, w, and\ d$ represent the head height, width, and depth (Hoene, et al., 2017).

Traditionally, applications of HRTF-based audio have utilized large sets of generalized HRTFs when trying to localize virtual sound sources. Of course, this approach can be seen as a way of catering to the many,

without requiring any significant overheads, albeit that certain research has proposed ways of providing effective approximations of individualized HRTFs without the requirement for a user to go to a specialized audio lab (Hoene, et al., 2017). The KEMAR HRTF data set is a popular source of generalized HRTFs that were used to sample 710 different positions at different elevations. This research was performed at the MIT media lab, and the database of these HRTFs is available to use for research and in applications (Gardner & Martin, 1995). Indeed, proprietary sets of HRTFs are also generated and used in applications. The procedures employed for the recording and generation of HRTFs are well documented, and the *direct HRTF measurement* method is the most popular of these. It entails transmitting test signals from different (documented) locations, and then measuring the resultant signals as they arrive at each of the manikin's ear canals (Zhang, et al., 2009). Usually Knowles Electronics Manikin for Acoustic Research (KEMAR)[32] manikins are used in place of the human listener, and an anechoic chamber is used for recording.

An important development in the HRTF paradigm was the standardization of HRTF data formats due to the introduction of the Spatially Orientated Format for Acoustics (SOFA), which was conceived in 2013, and has subsequently become the standard HRTF data format (Majdak, et al., 2013). This has facilitated the ability of audio developers to utilize generalized HRTF databases without having to sanitize/format the data before using it. In the context of VR, it has been suggested that generalized HRTFs might achieve the spatial requirements for immersive VR applications (as opposed to individualized HRTFs) due to cross-modality, which denotes that the exposure of both audio and visual stimuli facilitates an improvement in the perception of virtual audio sources (Berger, et al., 2018). The findings in this research further vindicate the need for the research performed in this thesis, as they suggest that the addition of a visual layer to an auditory performance analysis of spatial audio rendering algorithms may aid in determining their ability to successfully localize virtual sources. Of course, these multisensory studies should also be accompanied by exposure to audio-stimulus-only studies.

Headphones are ubiquitous in contemporary society, and this means that HRTF-based spatial audio represents a form of spatial audio application that has far-reaching impact (Garner, 2017). It is also worth recognizing that the importance and usefulness of immersive audio cannot be understated. As more digital media require the precise localization of virtual audio sources, immersive audio will become a real force in both professional and consumer spaces. The range of rendering algorithms detailed in the previous sections indicates that the sphere of research will evolve in both the academy and in industry, which will only contribute to the techniques and technologies proliferating in numerous different spheres of our society. It can be presumed that this trajectory will mirror that of VR technology, as they are both inextricably linked.

3.5 Existing Speaker-Based Immersive Audio Systems

Having discussed some of the principal ways in which immersive audio can be implemented in immersive audio systems, this section will now introduce two of the immersive audio systems that are currently being deployed. More importantly, the specific Immersive Audio system (ImmerGo) that was utilized for this research will be discussed.

[32] http://kemar.us/

3.5.1 Dolby Atmos

Perhaps the best known commercially available immersive audio system is Dolby Atmos, which is a hybrid channel-object-based system that was introduced in 2012 which has since become a major influence in both the adoption of object-based audio production and distribution. The system can enable up to 400 speakers in a cinema environment, and up to 34 in a home theatre setting (Roberts, 2021).

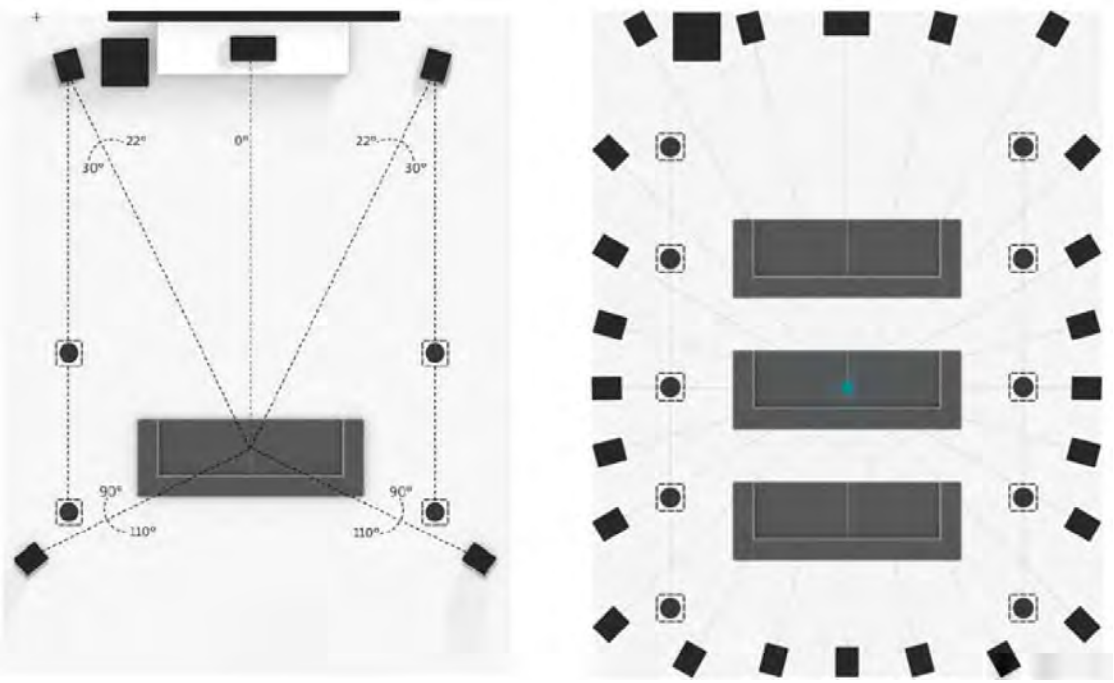

Figure 41: Two images that indicate different Dolby Atmos speaker configurations. The left image indicates a Dolby Atmos set-up with five floor speakers and four overhead speakers (5.1.4). The right image indicates a maximal home theatre Atmos set-up with 24 floor speakers and 10 overhead speakers (24.1.10). Each set-up also has a subwoofer (Dolby, 2018).

The system can be classified as an audio file format that enables object-based audio production and reproduction, while also allowing for ten channel-based tracks. Figure 42 illustrates the workflow that producers would utilize in their production of Atmos-enabled content. In accordance with OBA standards, Atmos functions by implementing audio objects into virtual scenes. The software enables up to 118 object-based channels, thereby allowing for a huge number of audio objects in any given scene. If one were to use a 5.1.4 Atmos configuration, one would be able to utilize 118 audio objects in a virtual scene or mix (Dolby, 2021). This provides both filmmakers and indeed VR/game developers with the ability to viably spatialize every audio source in even the most frenetic scenes.

The system has been widely adopted and, as of July 29, 2021, the solution is supported by all the major Hollywood studios. Dolby Atmos has over 450 cinemas committed to or using the solution globally. In addition, over 2000 feature films or series have committed to or use Atmos in their production (Dolby, 2021).

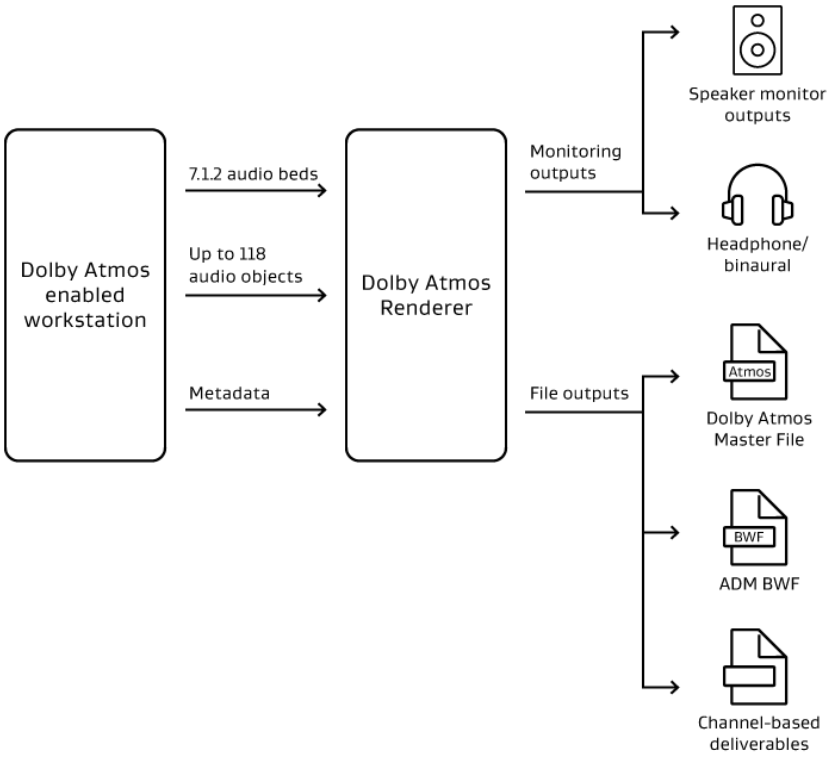

Figure 42: A diagram indicating the workflow for Dolby Atmos content creation (Dolby, 2021)..

3.5.1.1 Dolby Atmos Immersive Audio Format

An important aspect of the Dolby Atmos Audio Format is that it contains a mix of both CBA and OBA principles, which is indicated in Figure 42. The Dolby Atmos Renderer renders three different file formats. The Audio Definition Model Broadcast Wave Format (ADM BWF) format is based on the Audio Definition Model (ADM) specification, which is provided by the European Broadcasting Union (EBU). As defined by the guidelines, "the ADM is standardized metadata model for describing the technical properties of audio. ADM metadata can be attached to audio files to ensure the audio is correctly handled" (European Broadcasting Union, 2021). The Dolby Atmos Master Files contain .atmos audio and metadata files and enable punch-ins and metadata updates that the ADM BWF format does not allow for (Dolby, 2021). The 'Channel-based deliverables' file format is used to provide channel-based outputs for the audio for non-Dolby Atmos distribution (Dolby, 2021).

Dolby recognized the limitations of channel-based audio (see Channel-Based Audio section), and decided to adopt an OBA approach to solve the inefficiencies of CBA production and to facilitate high-fidelity immersive audio into both studio and home solutions (Dolby, 2018). In reproduction/rendering, the system functions by providing a software suite that enables one to indicate their exact speaker configuration (placements). Once the software is aware of the type of each speaker as well as its position, the Object Audio Renderer (OAR) analyzes each audio object in a scene or mix via its metadata and determines how to position the audio source relative to the supplied speaker configuration in real-time (Dolby, 2018). However, as was mentioned earlier, Dolby Atmos enables backwards compatibility for CBA. Figure 43 indicates the User Interface of the Dolby Atmos Renderer application that is used in

the configuration of the system's speaker layout, and the positioning of audio objects in three dimensions.

Figure 43: A screenshot that indicates the User-Interface of the Dolby Atmos Renderer application (Dolby, 2021).

3.5.2 ImmerGo

This section will briefly introduce the ImmerGo spatialization system and its mode of operation. After this, the underlying standard (and its suite of protocols) that enables ImmerGo's operation will be introduced. Thereafter, the components of ImmerGo will be detailed. ImmerGo is a speaker-based spatialization system that employs a client-server approach in enabling immersive audio rendering (Devonport & Foss, 2019). It can be described as a Spatial Audio Workstation that utilizes Ethernet Audio Video Bridging (AVB) for communication across a network. The system has been developed to interface with Digital Audio Workstations (DAWs), which can be defined as "software for editing and processing digital audio" (Leider, 2004). ImmerGo has also been developed and integrated to operate with the REAPER Digital Audio Workstation (DAW).[33] However, the underlying mechanisms of this integration are not centered on a plugin-based approach like other equivalent spatial audio workstations.

The system subscribes to an OBA based approach to spatial audio and enables the localization of audio sources in three dimensions by virtue of assigning audio tracks to audio objects that contain metadata. This metadata contains a variety of information pertaining to a specific object or track that is used by the Audio Renderer (AR) to calculate mix levels, which, in turn, are distributed to the speaker endpoints. The system utilizes a Track Object Model, with a track identifier, which connects a track to its associated metadata (see Table 2).

[33] https://www.reaper.fm/

The metadata is separate from the actual source (track) signal. Each of the variables contained in the metadata (apart from the Track ID) can be dynamically changed in real time to enable positional and playback changes of virtual audio sources depending on input from a user. The input is provided from the user through the ImmerGo client UI (see Figure 44 below).

Track Object Model	
Metadata Variables:	**Variable State:**
Position	Dynamic
Time Stamp	Dynamic
Spread	Dynamic
Volume	Dynamic
Track ID	Static
Mute	Dynamic
Solo	Dynamic
Selected	Dynamic

Table 2: A table indicating the ImmerGo Track-Object Model.

3.5.2.1 Ethernet AVB

As has previously been noted, the ImmerGo system uses miniDSP AVB-capable NDAC-8 devices to receive audio channels and to perform appropriate matrix mixing of these channels to attached speakers. The AVB standard was engineered to better meet the needs and requirements of real-time multimedia being distributed across a Local Area Network (LAN). Indeed, traditional Ethernet networks are not suitable for real-time transmission of audio and visual media since Ethernet packets are unable to guarantee set latencies and packet arrival. In other words, these traditional networks are unable to support the Quality of Service (QoS) required by real-time multimedia streaming applications in loaded networks (Lim, et al., 2012).

The AVB standard is defined by IEEE 802.1 Audio/Video Bridging Task Group, which provides the specifications for streaming services through IEEE 802 networks. The AVB specification facilitates "time-synchronized", "low latency", and loss sensitive audio/visual streaming (Alderisi, et al., 2012; Lim, et al., 2012). The specifications can be further divided into three components, the first being the IEEE 802.1AS Time Synchronization specification, which denotes the precise time synchronization of distributed local clocks. The second component is the IEEE 802.1Qat Stream Reservation specification, which enables resource reservation within switches on a network between a sender and receiver. The final component is the IEEE 802.Qav specification, which is used to differentiate different network traffic according to whether it is time-critical or not (Alderisi, et al., 2012). In addition to the IEEE 802.1 set of specifications, the IEEE 1722 specification informs the transport and configuration protocols of an Ethernet AVB network. The following table indicates both the standards and the protocols that they advise.

Standard:	**Protocol:**
IEEE 802.1AS	Precision Time Protocol (PTP)
IEEE 802.1Qat	Stream Reservation Protocol (SRP)
IEEE 802.1Qav	Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS)
IEEE 1722	Audio Video Bridging Transport Protocol (AVTP)
IEEE 1722.1	Audio Video Discovery, Enumeration, Control and Connection Management (AVDECC)

Table 3: A list of standards and accompanying protocols that apply to Ethernet AVB.

The AVB standard reserves bandwidth for the express purpose of real-time multimedia streaming via AVB packets. Due to the three components listed above, this standard can make certain Quality of Service (QoS) guarantees for multimedia streaming on an Ethernet AVB enabled network. The standard is loss sensitive due to the reservation of bandwidth across the network (SRP). In the event of network congestion, the AVB packets will be treated preferentially and will gain forwarding precedence. All AVB nodes on an Ethernet AVB network are accurately aligned to a common master clock on the network via the Precision Time Protocol (PTP), and according to the IEEE 802.1AS specification. A further important protocol used by ImmerGo is the AVDECC Protocol (IEEE 1722.1), which enables control of the NDAC-8 device matrix mixers (Devonport & Foss, 2019).

An AVB device can be a listener, a talker, or both. Each device creates an AVB specific entity that is contingent on its specific MAC address. This enables the devices to be exclusively addressed across the AVB network. Talkers on the network stream media (audio or visual) out into the network, while listeners receive these incoming streams. AVTP is utilized for the streaming of this media, whilst the AVDECC protocol enables the connection of devices on the network and then the subsequent control of their parameters. Once devices are connected, audio can be streamed (Devonport & Foss, 2018).

3.5.2.2 The ImmerGo Client

The traditional client of the ImmerGo system provides a web-based User Interface (UI), which is used to position a particular track in three dimensions (see Figure 44). As the interface is browser based, it can be used by an array of different devices. The client was developed in Raphaël, which is "a JavaScript library designed specifically for artists and graphic designers". [34] The client can communicate in real time with the ImmerGo server via TCP/IP messaging. Its interface can be divided into two parts:

- The Object Control Window enables users to position track objects both horizontally and then vertically.

- The bottom part of the UI, or Track Object Deck is used to control track object selection, whilst also enabling playback controls like 'muting' or 'soloing' the specific track.

The ImmerGo client's requirements for interfacing and communicating with the server are detailed in chapter five, as the capability that has been developed in this research serves to replace the traditional client with a Unity-specific version, thereby enabling the spatial audio provided by the ImmerGo server to be utilized in VR and digital game development.

[34] http://raphaeljs.com/

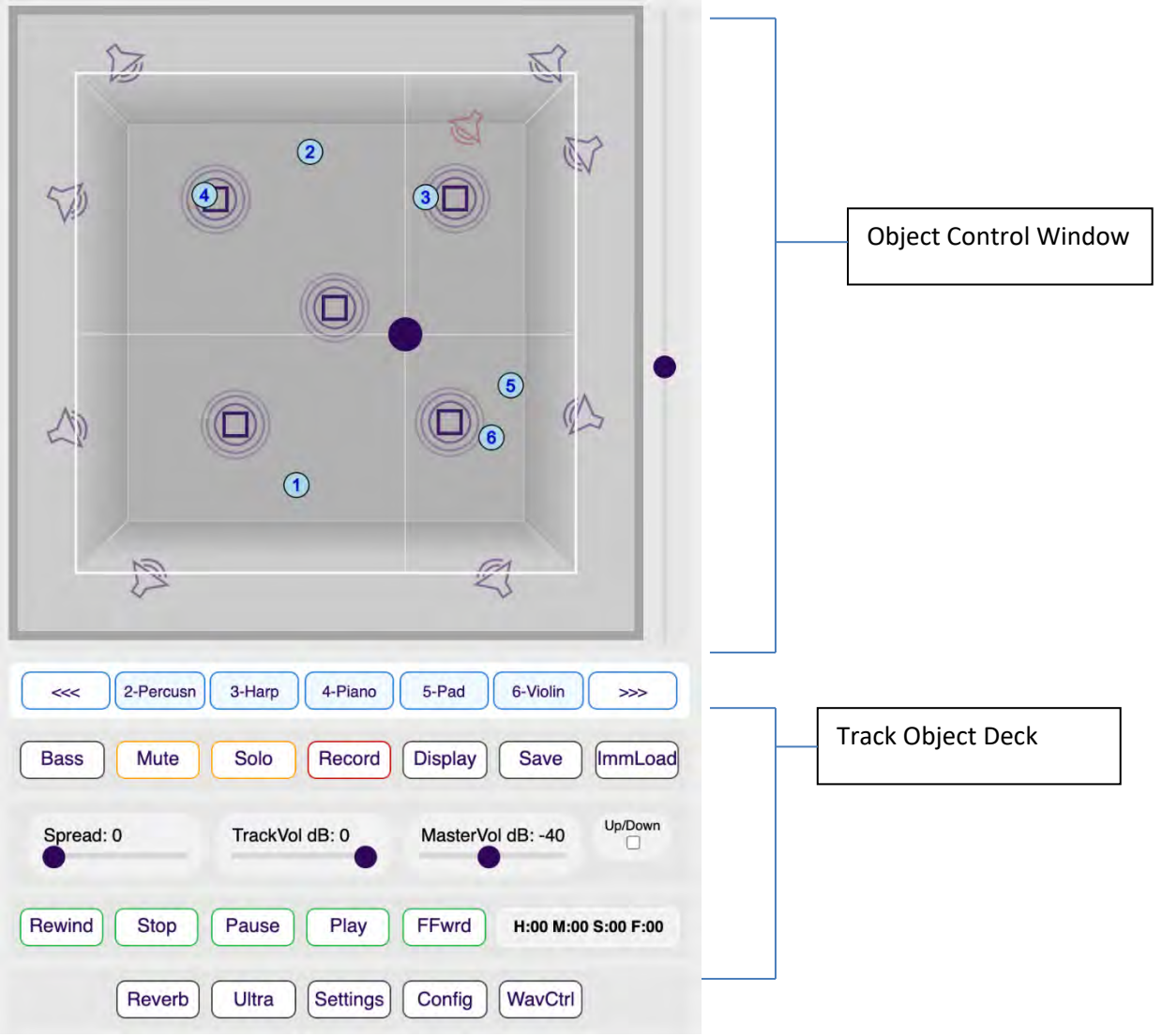

Figure 44: A screenshot of the ImmerGo UI Client.

3.5.2.3 The ImmerGo Server

The server component of ImmerGo houses fundamental capabilities of the ImmerGo Spatial Audio Workstation, namely:

- Providing transport control over a DAW as well as an in-built multi-channel WAV player.
- Performing rendering calculations
- And providing AVB communication.

Each of these areas will briefly be discussed, and an overview of the workflow of the system will also be provided. Initially though, the development of the application will be detailed, and an overview of the

desktop UI will be given. The Immergo server is hosted using NodeJS [35] and a collection of NPM[36] modules to provide additional functionality. This framework made it possible to port the server to both Windows and Unix-based Operating Systems (OSs). The Windows application User Interface (UI) can be seen in Figure 45. A developer using the system will need to navigate to the working directory in which the speaker layout file is located. They will be able to see the network on which they are operating, and will be provided with the URL for the client application. In addition, the number of AVB compliant devices (to which speaker-end points are connected) on the network is also displayed. The final area of the UI indicates the MIDI ports by which the server communciates with the DAW (detailed below).

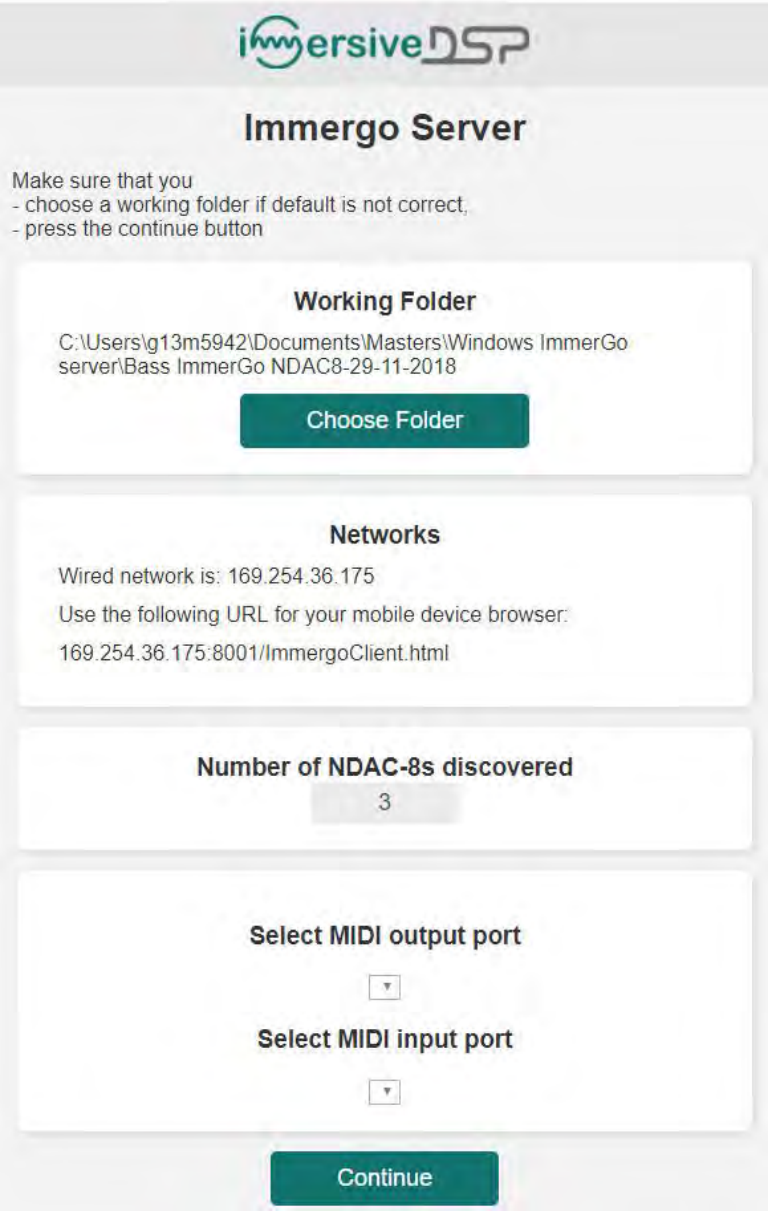

Figure 45: A screenshot that indicates the typical interface of the ImmerGo server.

[35] https://nodejs.org/en/
[36] https://www.npmjs.com/

3.5.2.3.1 Interfacing to the DAW

The server interfaces with the DAW by using Musical Instrument Digital Interface (MIDI) commands. MIDI is "an industry standard music technology protocol that connects digital musical instruments, computers, tablets, and smartphones from many different companies" [37] In this context, MIDI is used to control the playback of audio tracks housed within the DAW. After receiving specific client messages pertaining to the playback requests that a user can employ, the server then forwards them through the designated MIDI output port. These requests are then processed by the DAW.

Playback Commands	MIDI Output
Play	`[0xB0, 0x1A, 0x00]`
Pause	`[0xB0, 0x1B, 0x00]`
Stop	`[0xB0, 0x1C, 0x00]`
Rewind	`[0xB0, 0x1E, 0x00]`
Fast Forward	`[0xB0, 0x1F, 0x00]`

Table 4: This table shows the playback request and the appropriate MIDI message that is forwarded to the DAW

Of course, this MIDI communication functionality is not employed in the system that was developed in this project, as the audio was authored using FMOD (see the following chapter detailing Audio Middleware) and streamed from Unity to a standalone playback server and not a DAW.

3.5.2.3.2 ImmerGo rendering algorithms

One of the most desirable features of ImmerGo is that the system can utilize a suite of audio rendering algorithms in its reproduction of virtual audio sources. This can be attributed to the OBA mixing approach that it uses (Devonport & Foss, 2018). The metadata for each of the tracks (see Table 2) is employed to facilitate the specific rendering algorithm that is selected. Indeed, the mix levels of each speaker are determined according to the principles of the rendering algorithm and the three-dimensional location of the virtual sound source (track). At present, the system supports three of the rendering algorithms mentioned in the preceding section: Distance-Based Amplitude Panning (DBAP), Vector-Base Amplitude Panning (VBAP), and Ambisonics (Devonport & Foss, 2018; Devonport & Foss, 2019). Each of the rendering algorithms presupposes a particular ImmerGo based speaker configuration. The DBAP algorithm has been implemented using standard loudspeakers, while the VBAP and Ambisonics configurations use Power over Ethernet (PoE) endpoints (loudspeakers).

This flexibility in rendering algorithm enables ImmerGo to exist as a very modular and case-specific spatial audio workstation, since a user can decide which algorithm best suits their needs and utilize this implementation in their projects. The pros and cons of each of these algorithms are detailed in the previous section on Speaker-Based Immersive Audio Rendering Algorithms.

3.5.2.3.3 AVB communication

Once the mix levels for each of the speakers have been determined for a particular source, they are applied to the speaker via AVB communication, which involves the use of an *updateSpeakerMixLevels* function that is provided with the metadata of the Track Object(s) as well as the speaker positions. After the mix levels are calculated, an AVDECC compliant packet is populated with the information required and distributed over the network.

[37] https://www.midi.org/

3.5.2.3.4 System workflow

The workflow of the system is now outlined. The reader should appreciate that the process of employing this server for VR and digital game development differs in a few areas, and they should therefore review chapter six to recognize these differences. Before launching the ImmerGo application, a speaker layout file needs to be populated with the exact positions of the speakers in the physical speaker configuration. If the incorrect speaker positions are supplied, then the mix levels calculations will be inaccurate, as they would reference a position that a speaker does not inhabit. The following code block indicates how a speaker position in the file would typically look.

```
//Speaker 1
<speaker number = "1" xpos = "-106" ypos = "0" zpos = "106" stype="normal">  </speaker>
```

Code Block 1: Speaker one's position in the speaker array. As can be seen, its identifier, and coordinates are presented along with a description of the speaker type. The reader should note that the unit of measurement in this file is one centimeter.

On first starting the server, the user will be asked to start a client, if there is not a speaker configuration file in the working directory. When the client is started, the server will initially display a configuration screen for speaker entry. The server enables the identification of MIDI ports for transport control and time frame display, and will also indicate all the AVB Network devices that are active. When the user selects the “continue” button, the ImmerGo client will be hosted, and they can navigate to it from any device that is connected to the network.

The user can now position Track Objects in three dimensions using the client UI. The Track Objects’ identifier will correspond to the track arrangement in their DAW or chosen multichannel WAV file, thereby enabling spatial audio for use within their DAW multitrack project (or multichannel WAV file).

3.5.2.4 The Allure of this System

A highly attractive feature of Immergo is that it has been developed to provide flexibility with regard to sound sources and their origin. The native system facilitates audio playback through an interface with DAWs, whilst also providing an in-built WAV player for multichannel audio tracks that are not housed within a DAW. This feature of the immersive audio system enables a modified ImmerGo system, in tandem with the immersive audio capability that has been developed, to provide audio playback of FMOD audio events through Unity, which is a fundamental requirement of the research and is detailed in chapters five and six.

3.5.3 Dolby Atmos and ImmerGo Comparison

It is necessary to compare the immersive audio systems that have been introduced. The major difference between the two systems lies in their approach to OBA, and, specifically, the relationship between audio assets/data and the metadata associated with a particular audio object. Figure 46 indicates this relationship. The data for every audio frame of the audio object can be separated into two distinct components, namely the audio assets and the metadata. An audio frame can be defined as an audio element that contains the appropriate audio asset and metadata for a given audio object/file. In addition, an audio frame does not overlap with sections of the audio data to which it pertains (Botha, 2017).

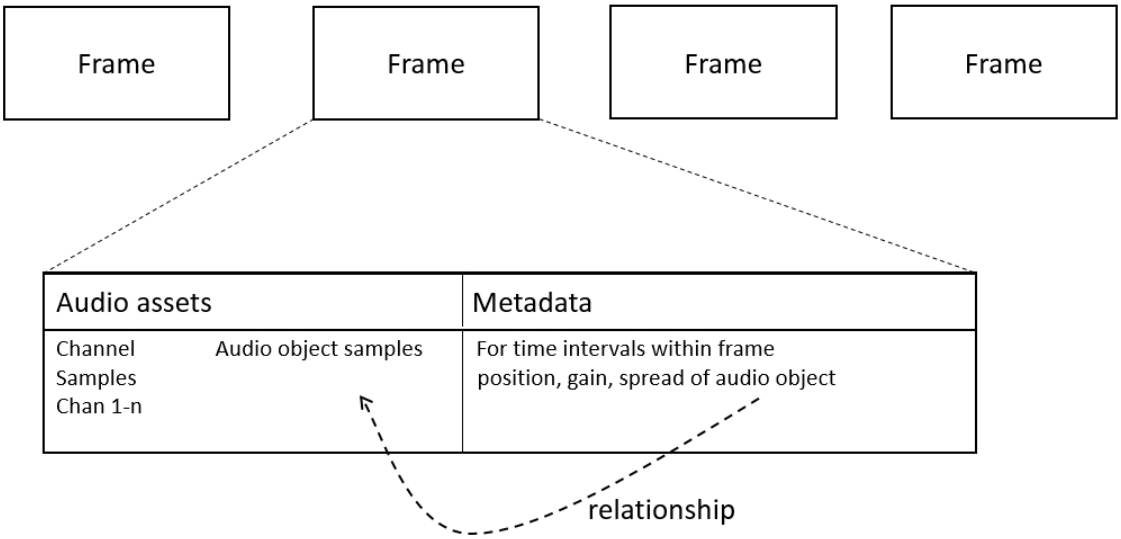

Figure 46: A schematic diagram that indicates the relationship between the audio assets and the metadata that pertain to them (Foss & Rouget, 2016).

In the case of Dolby Atmos, the audio assets and metadata, which are defined using the Dolby Atmos Renderer application (see Figure 43), are bundled together and output into a master file. This master file then contains both the assets and the metadata associated with a particular audio object and thus audio channel. On reproduction, the audio assets, and metadata are interpreted by the renderer from a single source, i.e., a data file that contains both the audio assets and the metadata.

In contrast, the ImmerGo spatialization system provides a different approach to the relationship between audio assets and metadata. In its implementation, the two components exist as discrete entities. The metadata exists in XML files that are used to provide the structure of the object model to which the system subscribes (Foss & Rouget, 2016). Table 2 indicates the metadata variables that are stored within this file. The audio object's audio assets (audio samples) are stored in multichannel WAV files, or in WAV files associated with a DAW. Upon reproduction, the spatialization system uses the variables within the metadata file to synchronize with the audio data per audio frame to provide accurate spatial audio.

In summary, the two systems provide synchronizes audio frames for output and immersive audio spatial rendering but do so in different ways. Dolby Atmos provides a holistic approach to the relationship between audio assets and metadata, while ImmerGo stores each of this data separately.

4 AUDIO MIDDLEWARE AND THEIR GAME ENGINE INTERFACES

"Reality is broken. Game designers can fix it."

— Jane McGonigal

This chapter will describe the two software frameworks which this project uses. As such, an overview of the Unity game engine is discussed, as well as the FMOD Audio Middleware solution. Subsequently, the way in which FMOD can be integrated into the Unity engine is also explored. This integration firstly facilitates greater game audio control, and in the context of this project, enabled the development of the custom DSP plugin that forms an important component of the immersive audio capability that has been implemented (detailed in chapter five).

The chapter is structured into an introductory section that analyzes the role of audio in VR and digital game applications, and how immersive audio is an important component of VR and game development. After which, there is a discussion on game engines with a particular focus on the fundamental aspects of the Unity game engine which facilitate VR and digital game development. Having given the context to game engine development processes, Audio Middleware will be defined and the workflow of the FMOD Audio Middleware solution will be provided. The final section of this chapter will then expand upon the steps taken to integrate the FMOD middleware solution and the Unity game engine. This chapter therefore provides context to the implementation of one the primary components (the *ImDSP* plugin) of the immersive audio capability, while also describing the software frameworks employed in the implementation. As such, this chapter contributes to the theoretical and practical frameworks by which the primary and secondary goals of this project were realized.

4.1 THE ROLE OF AUDIO IN VR AND DIGITAL GAME APPLICATIONS

The introduction of this thesis briefly indicated the reasons as to why audio is an essential component in VR and digital game applications. This section will look to further establish the importance of audio in these applications. The role of (immersive) audio in VR and digital game applications can be abstracted into three primary functions— to inform, to entertain, and to immerse a user in the virtual world (Sinclair, 2020). In the context of this project, and the immersive audio capability that has been developed, the principal purpose of the primary functions is to inform and immerse the user when utilizing a speaker-based spatialization system. However, as a conceptual understanding of immersion and how audio contributes to said immersion is provided in chapter two, the immersive qualities of audio are omitted in this discussion. Indeed, the function of entertaining the user is also omitted. However, the reader should recognize that as the realms of computer vision/animation/3D modelling all contribute to the visual aesthetics and therefore the quality of entertainment that a user might experience, so it is with the construction of immersive and stimulating soundscapes for the audio of VR and digital gaming applications.

4.1.1 Audio Conveying Information

If the reader reviews the Localizing Sound section in the previous chapter, they should recall that auditory spatial awareness is a crucial factor in determining general spatial awareness. Of course, spatial awareness can be defined as the "awareness of the surrounding space and the location and position of our own body within it. Thus, it is a multisensory awareness of being immersed in a specific real or virtual environment" (Letowski & Letowski, 2012). This informs a relationship between audio and visual stimulation. To accurately perceive the world (physical or virtual) around us, we exhibit a constant perceptual integration of our sensory inputs (Fleming, et al., 2020). Indeed, the cross-modal integration of perceptual information has been suggested to improve reaction times, detection rates, and the ability to identify objects when one or both modes of stimuli are weak. To better contextualize this relationship between visual and auditory information for virtual environment development, it is worthwhile to recognize two critical elements of this relationship that facilitate accurate spatial perception of virtual objects.

Firstly, the temporal coherence of both stimuli refers to the synchronization of the stimuli to establish the origin of a character or object (Fleming, et al., 2020). An example of this in the realm of digital games would be the synchronization of a characters' lips with their audio. Of course, if the temporal coherence of these two inputs is inaccurate, the player/user would immediately recognize something wrong with the implementation (a character talking when their lips or facial expressions are static). The second critical element can be defined as the spatial alignment of the two stimuli. This refers to the location of both the stimuli indicating the same position for the virtual character/object. If one reviews the previous example provided for temporal coherence, then inaccurate spatial alignment could result in a situation in which the audio for a character originates from a location that is different to their actual (visual) position. In this situation, the player/user would once again immediately recognize the discrepancy between both modes of information (visual and auditory). Accordingly, the primary aim of this project is to analyze the spatial audio provided by different spatial audio rendering systems. More formally, this involves a comparison of the spatial alignment of audio-visual information from differing audio output systems. Indeed, the experiments do not consider temporal coherence as it is assumed that both environments are indeed real-time systems, whose stimuli are temporally aligned (this assumption is confirmed in chapter seven).

It has been suggested that the primary elements of auditory spatial awareness are auditory localization, auditory distance estimation, and auditory spaciousness assessment. Each of these elements refers to the ability of humans to classify the direction of a sound, the distance from the sound to the sound source, and the ability to categorize the size and characteristics of the physical/virtual space that affects how the sound propagates through this space (Letowski & Letowski, 2012). Now that some understanding of the critical elements that affect spatial awareness and the relationship between the audio and visual modes of conveying information has been provided, the following paragraphs will look to discuss these concepts from the context of the initial notion that spatial audio needs to inform a player/user about their virtual environment.

4.1.1.1 Environmental Modeling

Environmental modeling refers to affecting audio propagating in a virtual scene according to the specific geometric parameters (architectural elements) of the virtual environment (Sinclair, 2020). As such, the concept of *environmental modeling* looks to virtually address the element of auditory spaciousness

assessment. Indeed, the audio in an outdoor virtual environment should sound different to the audio in a virtual room (indicated by changes in roll off, reverberation, and attenuation of the audio). If the audio source is obstructed by some or other virtual object, then certain elements of the audio should be occluded to better model how audio is affected by obstacles.

Thus, *environmental modelling* enables audio and game developers to provide information about an environment based on how the audio is affected. Additionally, a soundscape that is affected by an environment offers a much more compelling player experience (PX). This phenomenon has an even greater effect on speaker-based spatialization systems, as these systems generally also need to consider the acoustic characteristics of the physical space in which the audio is being reproduced, alongside those of the virtual environment. This concept of understanding how the physical characteristics of a reproduction space affect the reproduced audio has been formally defined by a room response frequency (Cecchi, et al., 2018).

4.1.1.2 Distance and Location

As indicated, both the distance and the location of a virtual sound source are essential components in spatial audio implementations. The assessment of the core elements of auditory spatial awareness suggests that there is a difference between both the distance and the localization mechanisms that humans employ when determining the characteristics of a sound (Letowski & Letowski, 2012). Whilst the distance of a virtual sound source obviously refers to the difference in distance between the source and the listener, the locality of a sound source encompasses a Euclidean understanding that defines the source's relative position according to its azimuth, elevation, and distance from a listener. Therefore, distance is one of three important aspects that determine source locality (Letowski & Letowski, 2012).

In practice, the perception of an audio source's distance from a listener is principally based on the ratio of dry and wet sounds reaching a listener's ear (Sinclair, 2020). A dry sound refers to the direct sound (no obstructions or reflections) arriving to a listener, while a wet sound is an indirect or reflected sound reaching the listener (see Figure 47 for a diagrammatic representation). The further away a sound source is from a listener, the more reverberation (wet signals from the source) is heard (Sinclair, 2020). Consequently, in developing the audio of a virtual environment, an audio engineer and developer should utilize reverberation implementations to model the distance of a virtual sound source. The locality of sound is referenced and described throughout the previous chapter with regard to both speaker-based and headphone-based implementations of spatial audio.

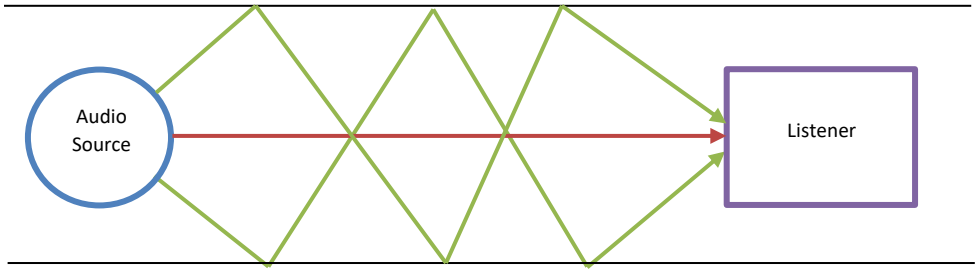

Figure 47: A diagram indicating the difference between dry and wet audio signals. The red arrow represents a dry signal that does not experience any reflection. The green arrows indicate wet sounds and exhibit the sound waves experiencing reflections throughout the environment before reaching the listener.

4.1.1.3 Environmental Feedback

A final way in which audio is utilized to provide a user/listener with information about virtual environments is by providing feedback to users in their various interactions with a virtual environment. Auditory environmental feedback is a way in which developers and sound designers can use audio to confirm actions and interactions in an environment (Sinclair, 2020). In the Introduction of this thesis, the example of a user pressing a virtual button, and an associative sound being triggered was presented as a form of environmental feedback. However, this is a simplistic example, and the real value of auditory environmental feedback lies in more complex scenarios.

For instance, if one considers a combat scenario in a fantasy Role Playing Game (RPG), then a typical way of augmenting auditory feedback to a player, would be by providing a specific sound when the player successfully attacks their opposition (e.g., a sound of metal hitting metal if the player's sword contacts the enemy's character model). Conversely, if their attack misses and fails, the sound would not trigger. This thereby constitutes an audio mechanism which establishes positive feedback with the player. Herein lies the role of audio to complement visual feedback in a virtual environment.

4.2 GAME ENGINES

Game engines were initially developed and gained traction in the 1990s in lieu of the "increasingly content-heavy nature of professional videogame development". Nicoll and Keogh suggest that the advent of game engines can mainly be attributed to the shift of digital game development from programmer-centric to content-centric paradigms (Nicoll & Keogh, 2019). This change refers to the increasing demands of consumers for digital games to exhibit high-fidelity (relatively speaking) graphics and stimulating gameplay. Up until this point, digital games had generally been created solely by programmers who focused on all aspects of the game. However, the content-centric paradigm refers to the process of digital game development providing more focus on aesthetics and art than just the gameplay mechanics (provided through programming). This shift then also denotes a change in the developmental approach whereby programmers needed to help artists and designers by implementing content creation tools. Indeed, this also presupposes the formation of large teams of people, all exhibiting different skillsets, facilitating the creation of digital games (Nicoll & Keogh, 2019). This therefore led to the founding of game studios and publishers whose employees specialize in the various of areas that game development encompasses. The shift of the gaming industry and the resultant supplementary software that programmers developed to aid in the design and implementation of digital games thereby gave rise to early concepts of game engines.

Sinclair suggests that a game engine is a collection of dedicated sub-systems that communicate and interact with one another to facilitate the production of interactive digital media. He suggests that a better way of conceptually understanding a game engine is to conceive of it as a sum of many parts, rather than assuming it to be one large and singular entity (Sinclair, 2020). Indeed, this perspective indicates a relative flexibility of the concept, as any number of relevant systems consisting of smaller sub-systems could therefore be classified as game engines. In accordance, research to taxonomically classify game engines and accompanying tools suggest that there should be a distinction between the concepts of a game production/development pipeline and the core software/engine that stiches all the content generated for a game together (Toftedahl & Engström, 2019). A game production pipeline can be defined as the set of tools (including the game engine) that are used in the production of assets that inform the content and mechanics of the game. Toftedahl and Engström argue that the issue with the

classification of a game engine lies in the complexities, plethora of tools, and competences required in the overall production process of game development.

Despite the ambiguity alluded to in the preceding paragraph, in the context of this project, a formal definition of a game engine is required. As such a game engine can be defined as a software solution that enables the production of real-time interactive digital media (applications), and the accompanying code framework that facilitates the functionality of the underlying sub-systems. These sub-systems generally encompass providing the development process with functionality pertaining to the graphical rendering of virtual environments, the audio and physics systems within these environments, the animation of virtual objects, and game networking facilities (although Figure 53 refers specifically to the Unity game engine, the diagram provides a representation of the underlying sub-systems/layers that a game engine is comprised of).

Before discussing the specific game engine that was utilized in this project, it would be beneficial to recognize that game engines were initially implemented to augment the in-house production pipeline of game studios and publishers. The reasons for the development of these proprietary pipelines stems from a desire for game studios to reduce the dependencies of their development process on software provided by competing companies (Freedman, 2018). The contemporary era of game development has since seen a "democratization" of the tools used in the construction of interactive digital media (Nicoll & Keogh, 2019). The Unity game engine can be attributed to being a large proponent of this initiative. The framework is ostensibly open-source, and apart from relying on updates from its developers in its evolution, benefits from its users continuously contributing additional functionality. The first iteration of the Unity game engine was released in 2005 (Haas, 2014). Therefore, the engine has been an important player in the game development landscape for over 15 years.

4.2.1 The Unity Game Engine Framework

Unity is "an umbrella term referring to a suite of technologies and tools whose collective purpose is to help developers create interactive software products, most notably (digital) games" (Thorn, 2013). Indeed, this definition provides a clear view as to how Unity should be defined in the context of this project. Whilst VR experiences are not necessarily digital games, they should be conceived of as "interactive (immersive) software products". It is suggested that Unity exists as three critical components

- A game engine, which facilitates the implementation, testing, and deployment of an application.
- The user interface for the engine, which provides a preview of the game graphics, and the playback control of the application being developed.
- And finally, the Integrated Development Environment (IDE) forms the third component and enables the development of application logic through code (Buyuksalih, et al., 2017).

These three components can be seen as the core components of the framework, but a Unity application often consists of many smaller components in addition to this core.

4.2.2 Unity Fundamentals

This section will discuss three aspects of the engine that faciliate its use in constructing interactive software solutions (such as the VR experience detailed in chapter six). The first of these aspects comprises the base constructs and classes that faciliate the construction of a virtual scene populated by virtual objects. Indeed, a discussion on how these virtual objects interact will also be provided. The second aspect to be examined will be that of the layered approach by which the Unity game engine functions. In addition to this layered approach, there is an examination of the execution order in which the engine operates. This execution order benefits from the knowledge of how the different layers of the engine combine at application runtime. The final aspect to be examined will be the UI that the Unity engine provides, and the windows of the game engine that are presented to a developer.

4.2.2.1 Unity GameObjects and Components

This section will detail the fundamental mechanisms and abstract objects which comprise the Unity game engine. The Unity Engine subscribes to an Object-Oriented Design (OOD). As such, a discussion of the various elements that a Unity application is comprised of is provided, after which, the elements that facilitate the construction of a virtual scene in Unity are discussed. The Unity engine operates at a series of different levels/scales. The following section describes the elements that exist at each of these levels (Nicoll & Keogh, 2019). Figure 48 provides an overview of a Unity application, and the levels that it is comprised of. The diagram indicates how each level is nested within a subsequent level. As the figure is an abstract representation of these levels, only a single GameObject is indicated, but the reader should recognize that even a simple Unity scene will generally consist of numerous objects (see Figure 56). The following diagram should be referred to when reviewing the following sections.

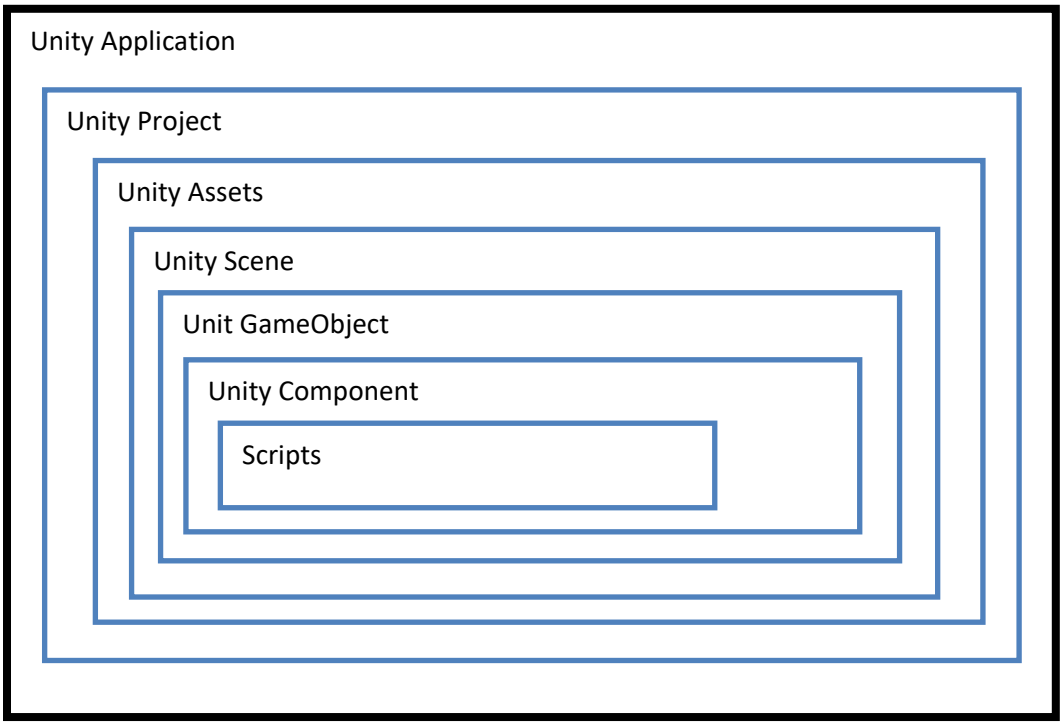

Figure 48: A schematic diagram indicating the different levels or layers by which the Unity engine operates.

4.2.2.2 Projects, Assets, and Scenes

A Unity *Project* operates at the highest of these levels and refers to the Unity application in its entirety. The *Project* therefore encompasses all the files associated with the project. This therefore includes all the elements that are utilized throughout a Unity application (i.e., the textures, scripts, models, etc.). These elements need not be Unity generated and, as such, they are generally created outside of the Unity framework in software relevant to their creation.

The elements that a project consists of generally fall under the umbrella term of a Unity *Asset*. The files that a *Project* possesses are therefore its *Assets*. In general, should the functionality of the game depend on something developed outside of the engine, this functionality (in the form of a file) is imported into the *Project*. Unity's asset pipeline (the parameters and processes used in importing functionality into a *Project*) is often regarded as easy to use, compatible with a huge array of asset development software, and follows an industry standard for associated functionality in game engines (Haas, 2014).

When constructing a Unity application, the primary components of the *Project* are Unity *Scenes*. As such, a single project consists of one or many *Scenes*. This nomenclature mirrors that of the film industry, and accordingly, a *Scene* can be seen as a "single unit of the project, such as a particular level of a videogame, or a test scene wherein a developer can try out new mechanics" (Nicoll & Keogh, 2019). Indeed, a *Scene* is also a Unity *Asset*.

4.2.2.2.1 GameObjects

As defined by Unity, a GameObject is a "base class for all entities in Unity Scenes" (Unity Technologies, 2021). [38] Therefore, GameObjects are the primary abstract class that populate *Scenes.* As mentioned, a *Scene* is generally composed of one or more GameObjects. A developer populates a scene with objects by either dragging pre-defined objects (see Prefabs) into the scene from their project *Assets*, or by creating a GameObject from their hierarchy window (see Figure 49).

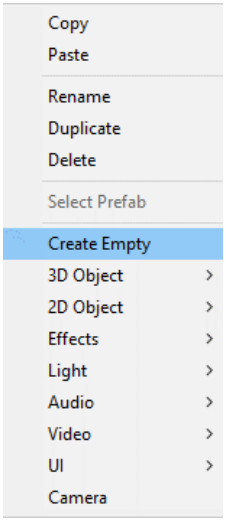

Figure 49: A screenshot of a drop-down menu in the Hierarchy window, which facilitates the creation of GameObjects in a Unity Scene.

[38] https://docs.unity3d.com/ScriptReference/GameObject.html

GameObjects don't perform any function by default (Polančec & Mekterović, 2017). The function of a GameObject is determined by the *Components* attached to it. Unity is designed to implement a "component pattern." A single entity (GameObject) can potentially span multiple domains (i.e., it can contain logic pertaining to the physics, audio, rendering, or AI system domains), and, to retain domain isolation, functionality for each entity is provided in its 'component' class. Thus, a Unity GameObject can also be defined as a "container of components" (gameprogrammingpatterns, 2021). [39] This implementation pattern can be primarily attributed to the desirability to decouple domains from one another, thereby rendering codebases reusable across these domains.

4.2.2.2.2 Components

As mentioned, a *Component* can be defined as the "base class for everything attached to GameObjects" (Unity Technologies, 2021). [40] *Components* therefore encompass a whole host of different functions depending on the functionality defined by their class. Figure 58 provides a graphical example of how *Components* are attached to GameObjects in Unity. In essence, components enable the construction of complex GameObjects with specific functionality. Instances of *Components* are attached to 'sockets' on the GameObjects (gameprogrammingpatterns, 2021).

As a *Component* of a GameObject refers to anything that can be attached to it, the fundamental *Component* of any Unity GameObject is the *Transform* of the object. This *Transform Component* provides the positional, rotational, and scale attributes of an object in the *Scene.* As such, it is the *Component* that allows a developer to manipulate locational information of an object in the virtual space (two-dimensional or three-dimensional). Every GameObject in a Unity *Scene* has a *Transform* (Unity Technologies, 2021). [41] Figure 50 provides an example of a *Transform* for a GameObject.

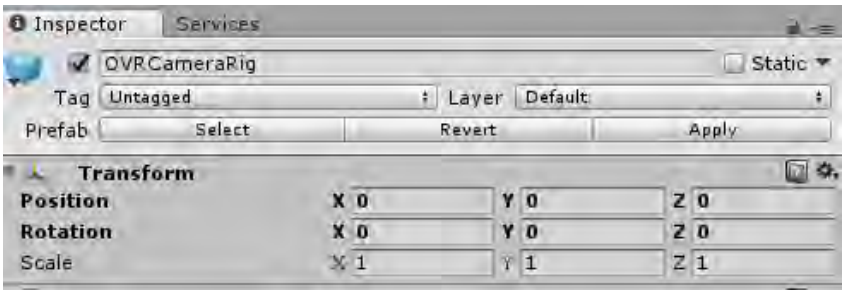

Figure 50: The Transform of the OVRCameraRig Game Object.

As can be seen, the object is located at the origin of the *Scene*. However, the visible Position attribute of a *Transform* refers to the *localPosition* of the object, and, as such, the information of this attribute refers to its position in relation to its parent object's own *Transform*. If the object has no parents (it is at the root of the *Scene*), then the *localPosition* of the object is that of the scene (i.e., the *Scene* origin). A *Transform* of a GameObject does however also provide the position of an object in world space. When referencing the *Transform* of a game object from script, the *localPosition* and *position* attributes refer to its local and global position respectively.

[39] https://gameprogrammingpatterns.com/component.html
[40] https://docs.unity3d.com/ScriptReference/Component.html
[41] https://docs.unity3d.com/ScriptReference/Transform.html

4.2.2.2.3 Scripts

Having now discussed the fundamental Unity entities (GameObjects and *Components*), it is worthwhile to briefly discuss what a Unity script is. The utilization of scripts within Unity facilitates the provision of game-specific behaviors in a Unity application. While Figure 48 indicates that scripts belong to a level beneath that of Unity *Components*, this is slightly misleading, as scripts can affect all the levels of a Unity application. Although custom scripts developed in this project provide functionality outside of that required by specific GameObjects, scripts will still be referred to as a type of *Component* attached to GameObjects.

Unity refers to scripting as a means of fulfilling the requirement of most applications "to respond to input from the player and to arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behavior of objects, or even implement a custom AI system for characters in the game" (Unity Technologies, 2021). [42] Indeed, in this project scripts are utilized beyond the gameplay requirements of the VR application that was developed, and have facilitated the provision of three-dimensional coordinates to the ImmerGo Spatialization System external to the Unity framework.

Unity scripts are generally authored in the C# or JavaScript programming languages. As was mentioned in the opening paragraphs of this section, one of the Unity game engine's critical components is that of an IDE. While Unity does provide its own IDE in MonoDevelop,[43] a developer can integrate an IDE of their preference into their Unity *Project*. In the instance of the Unity VR application that was developed for this research, the Visual Studio IDE was used. The IDE attached to Unity facilitates the authoring and construction of the scripts that provide requisite functionality to the Unity application. Indeed, Unity's Scripting Application Programming Interface (API) is an extensive library that provides the relevant namespace which a custom script needs to reference and adhere to. For example, if the developer needs to capture input from a user, the *UnityEngine.UI* namespace provides the requisite functions for recording input from the computer (be it key presses or mouse movement).

4.2.2.2.4 Prefabs

The previous sections all indicate the Unity application organization and primary mechanisms by which the game engine operates. While *Prefabs* do not necessarily represent a primary mechanism of the engine, they are a useful mechanism that enables a developer to "create, configure, and store a GameObject complete with all its *components*, property values, and child GameObjects as a reusable Asset" (Unity Technologies, 2021). [44]

Prefabs are therefore an integral component of the Unity engine that has been utilized in the construction of the immersive VR application. Indeed, the majority of Unity *Assets* that a developer employs within their project provide *Prefabs* of critical GameObjects with predetermined *Components* in that *Asset*'s implementation. Within the context of the application that has been developed, the Leap Motion and Oculus SDKs both provide *Prefabs* with controller objects that were utilized to render virtual hands (through hand-tracking) and to facilitate VR-specific functionality to the Unity *Scene.* Figure 51 gives an overview of some of the *Prefabs* that the Oculus Integration[45] *Asset* provides Unity developers.

[42] https://docs.unity3d.com/Manual/ScriptingSection.html
[43] https://www.monodevelop.com/
[44] https://docs.unity3d.com/Manual/Prefabs.html
[45] https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022

This integration *Asset* is reviewed in section 4.2.4. In the image, the *OVRCameraRig* is the selected *Prefab* that supports references to the underlying Oculus SDK, thereby facilitating the mechanisms by which a VR experience can be created in Unity.

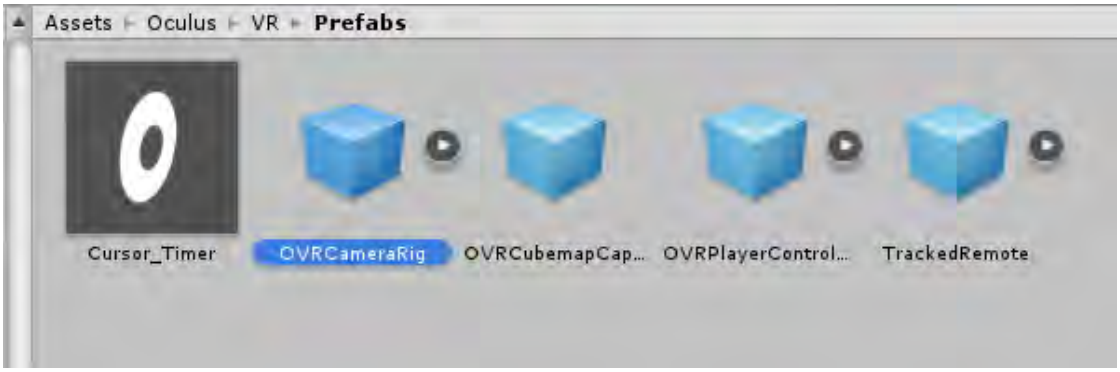

Figure 51: A screenshot of some of the Prefabs provided by the Oculus Unity Integration Asset.

Figure 52 indicates the *Components* that the *OVRCameraRig* game object possesses once it has been imported into a Unity scene. Of course, once a *Prefab* GameObject has been placed within the Unity *Scene*, the various attributes of the object can be modified according to the requirements of the Unity application, and of the scene. In this instance, the *OVRCameraRig* object provides the *OVRManager* script as a component, which houses the fundamental interface between the Oculus Unity integration library and the underlying Oculus SDK. Various rendering parameters and tracking options can be modified by the developer according to their requirements.

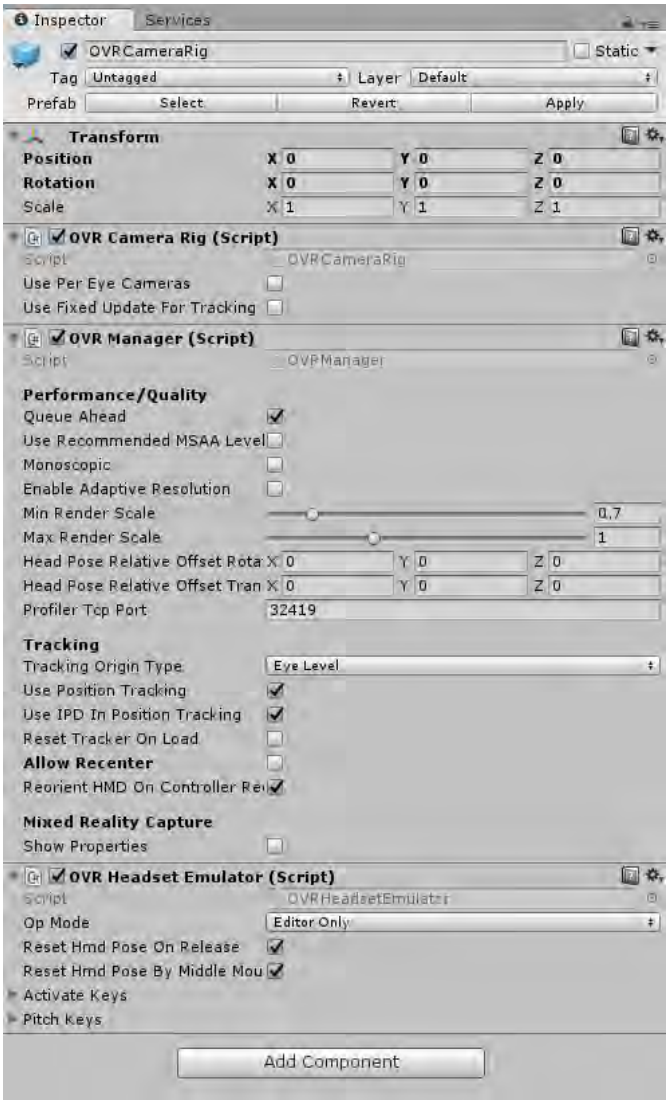

Figure 52: The OVRCameraRig GameObject's Components once it has been imported into a Unity Scene.

This section has provided the reader with a more concrete conceptual understanding of the fundamental mechanisms and design principles that the Unity game engine ascribes to. The following sections will seek to provide more context as to how the engine operates both in terms of execution, and from a developmental perspective (i.e., the layout of the graphical interfaces of the engine).

4.2.2.3 Unity Layers and Execution Order

The layered approach followed by Unity, and the execution order of the engine are interlinked concepts. The layered approach will be introduced before a discussion on how the execution of these layers is structured. If one considers the definition that has been provided for the Unity game engine, then one should recognize that the creation of an interactive Software application encompasses a variety of different aspects. Indeed, these aspects can be considered layers, and a complex Unity-authored application could consist of numerous different layers, each of which performs a specific role in rendering the application in its entirety.

Any complex application exhibiting graphical, auditory, Physics, networking, and interaction functionality would require references to each of these layers (see Figure 53). As such, the layers of Unity refer to each of the different systems that is utilized in providing specific capabilities to interactive software. The graphical (or rendering) layer would denote the component of the game engine that deals with rendering the virtual world to some or other output display. The audio layer of the engine would encompass audio event triggering and playback control. Each of the layers attempts to facilitate accurate control over one or more aspects of the application. The engine is comprised of many more layers than those that have been mentioned, but these can be conceived of as the primary ones.

The layered approach according to which one can categorize the components of the Unity engine, is a similar approach to that which characterizes immersion, as was explained in chapter two. The parallels are obvious, as a user's sense of immersion in an immersive experience benefits greatly from the addition of sensory modalities (sensory layers) to the system. Similarly, the complexity of a Unity application is directly proportional to the number of layers (components) that are used in its implementation.

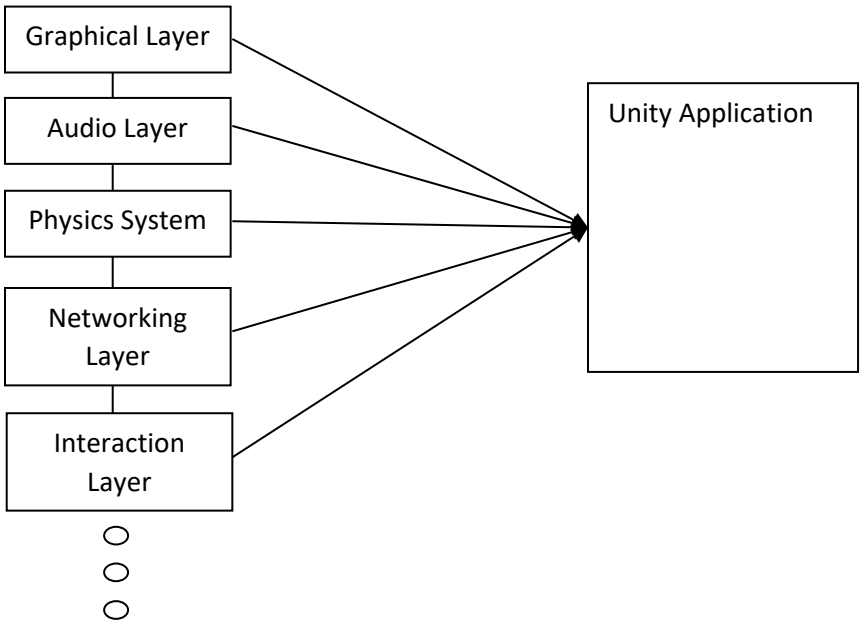

Figure 53: A diagram indicating all the layers that converge to create a (complex) Unity application.

As has been indicated, Unity as a software suite that exists in multiple layers, and the order of execution of these layers is indicated (implicitly) by the execution order of Unity scripts. Figure 54 provides an analysis of the life cycle of a Unity script. The colored areas represent low level multi-threaded functions that the internal Unity engine is performing. Within each of these areas, are specific execution/event functions that are performed throughout the developer's application scripts during the low-level function calls. The order of these event functions in the diagram is the order in which they are called from the script if they are referenced. Although the colored areas indicate the low-level execution of the Unity engine, they also serve to enforce the layered classification of the engine. Indeed, the green area could be defined as the phase in execution whereby the physics system functions are calculated. The red area represents the phase in which the graphical rendering functions are performed.

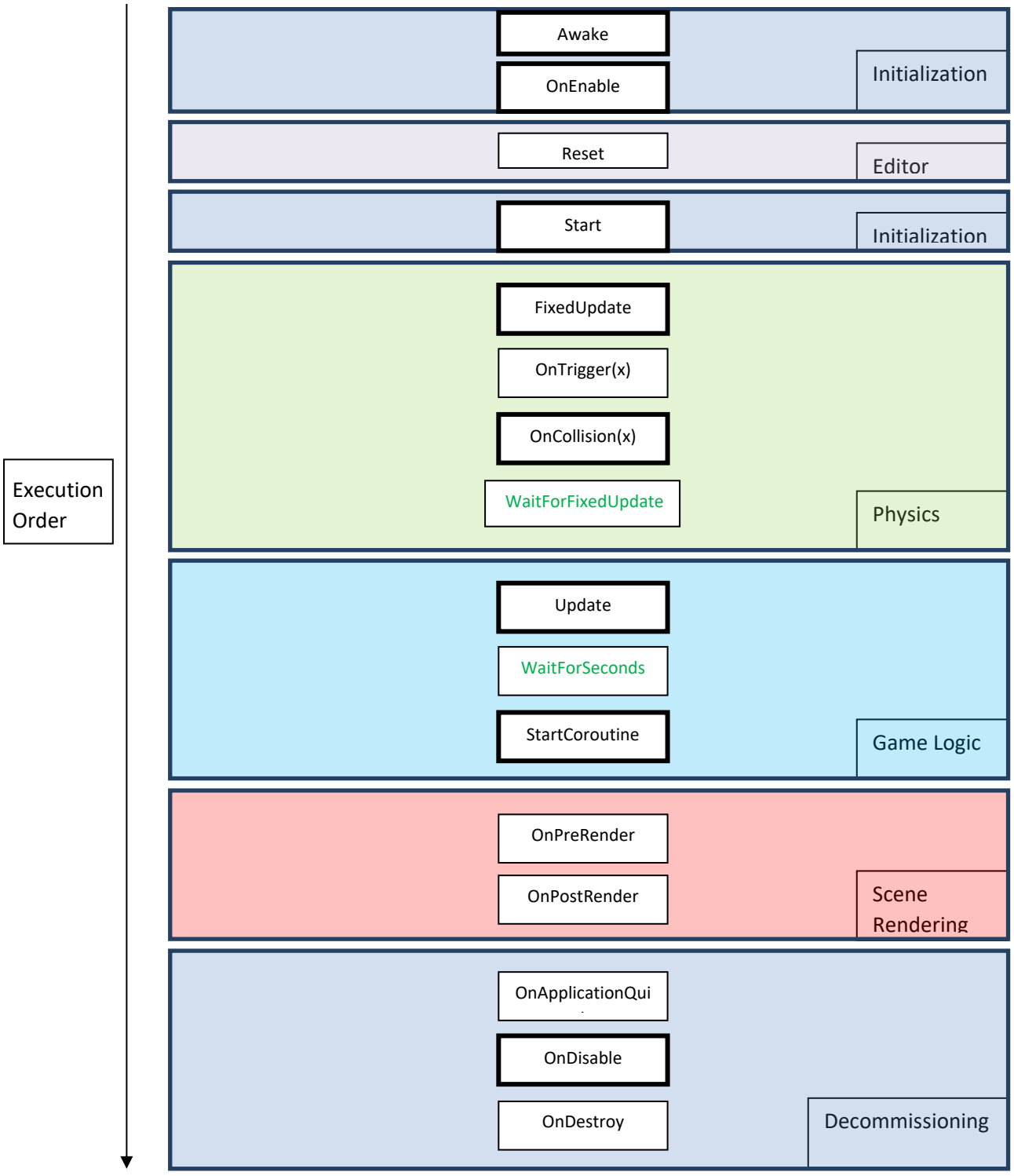

Figure 54: A lifecycle diagram of a Unity Script. This diagram was adapted from a Unity resource (Unity Technologies, 2021). [46] *The functions whose border is emboldened are execution functions of importance from the context of the Unity VR application that was developed.*

[46] https://docs.unity3d.com/Manual/ExecutionOrder.html

4.2.2.3.1 Execution Functions

While Figure 54 does provide a comprehensive view of the life cycle of a Unity script, many additional functions in this life cycle were omitted from this specific diagram. The functions that are presented in this figure are the functions of most importance to the implementation of the Unity VR application detailed in chapter six. All of these execution functions are not necessarily required in custom scripts, so the onus is on the developer to understand and utilize them as necessary. As such, a brief overview of a few of the utilized functions is provided (see Table 5).

Function Name	Function Description
Awake()	The first function to be called in a script's execution. Only called once.
OnEnable()	The first function to be called after the *Awake()* function. It is called every time the game object, to which the script is attached, or the script itself, is enabled.
Start()	A function that is called on the first frame in which a script is enabled.
FixedUpdate()	An update function that is executed at a fixed interval and is therefore useful for game physics operations.
OnCollision()	This function is called when the game object, to which the script is attached, collides with another game object.
Update()	Like the *FixedUpdate* function, this function executes every game frame. However, the time in which a frame is executed is not fixed. It is useful for game logic, but not physics operations.
StartCoroutine()	Executed after the *Update* function and launches a coroutine defined by the developer.
OnDisable()	A function that is executed when the script or game object it is attached to are disabled.

Table 5: A table that provides basic information about each of the Unity execution functions

Before discussing the particulars of these functions, it is worthwhile to note that Unity execution operates according to frames being rendered and presented to the output display. As such, each frame indicates a rendered image that is output to the display. These frames indicate points in (execution) time when functions should be called, or when executing functions need to return values. The Unity system indicates discrete points in the application execution that a developer can leverage. For instance, if a developer pauses the application execution from the Unity Editor, the application will receive this instruction and pause the application after the final frame that is queued is rendered (a discrete execution point whereby all active functions in the editor and application scripts have finished executing).

The *Awake* and *OnEnable* functions are useful in that they are the first of the functions in the script to be called. As such, they provide the ideal place to declare and initialize variables. The *Awake* function is only called once when the instance of the script is loaded and is therefore generally executed in the first few frames of the application. The *OnEnable* function is called when the *Component* script is enabled from a disabled state (see Figure 55). If the developer has a reference to a particular GameObject in their Unity *Scene*, they can affect whether it is enabled or disabled through this reference. Thus, the *OnEnable* function is useful in determining the current state of the game, and how this should affect the newly activated script. The *Awake* function is called irrespective of whether the script is enabled or disabled, which once again indicates that it is a useful point to declare necessary variables.

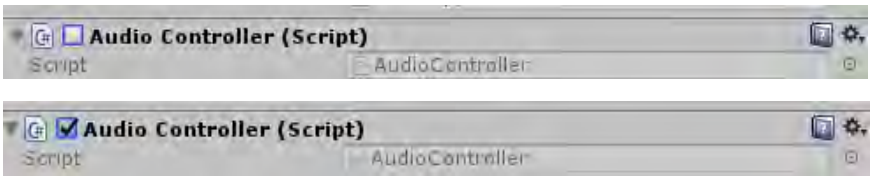

Figure 55: The inspector views of an AudioController script that is attached as a component to a GameObjects. The top image indicates that the script is disabled (unticked), while the bottom image indicates the script is enabled and will be active.

The *Start* function is called on the frame in which a script is enabled, and executes before any of the update methods are called for the first time. It is similar to the *Awake* function in that it is only called once in the lifetime of the script. It is general practice in Unity development to use this function to instantiate variables that might not have been instantiated from the *Awake* or *OnEnable* functions. Indeed, the *Start* function is one of two functions that is provided to a developer by default upon the creation of a new script. When a developer creates a custom script, a template for the script is provided by default. This template contains references to the *UnityEngine* namespace and includes both the *Start* and *Update* functions.

Once a Unity script has moved from the initialized phase of execution, it generally proceeds to the game logic phase unless there are references to functions in the physics phase. The *OnCollision* function is part of the physics phase and is called when the GameObject, to which the Unity script is attached, has collided with another GameObject in the scene. This is determined when there is contact between two *rigidBody* or *Collider* components of the two (or more) objects. As such, this function is generally utilized to trigger some or other event upon the collision of the objects. For example, if two objects collide, an audio event might trigger to indicate a "crash" or one of the object's colors could change. In the context of the immersive VR application that has been developed, this function contributes to the interaction mechanism that is detailed in chapter six, although it is never expressly referenced.

The execution functions adhere to frame-by-frame execution, there are a few exceptions to this precedent. The Unity physics system executes at fixed intervals in linear time. This type of execution is implemented in the physics system so that the virtual simulation of the physical interactions between GameObjects can occur at consistent intervals irrespective of a fluctuating application frame rate. Indeed, due to this fluctuation in application frame rate, if the physics system were based on a frame interval, the frame rate would affect the simulated physics of the system. Therefore, using linear time and fixed intervals between function calls provides a robustness and consistency that is a requirement for physical phenomena simulation.

The *FixedUpdate* and *Update* functions are generally used to "perform most tasks" required by the script. [47] These functions execute routinely throughout the script's lifetime. The physics system's utilization of time-based execution was introduced because the *FixedUpdate* is an update function that initiates the physics phase of execution and is followed by all physics calculations. Its execution is based on the time-based system of the physics system, and it is therefore guaranteed to execute at set time intervals, which is, as mentioned, a necessity when determining physical calculations (hence the function being *Fixed*). The *Update* function is executed on every application frame and is thus the optimal location in a Unity *Script* to perform game logic. It is in this function that developers generally provide gameplay interactions and event triggering, i.e., interactions that are not time dependent. The

[47] https://docs.unity3d.com/430/Documentation/Manual/ExecutionOrder.html

following equations indicate how often the *Update* function will execute in systems exhibiting different framerates.

$$60x = 1000ms$$

$$x = 16.67ms$$

$$30x = 1000ms$$

$$x = 33.3ms$$

(11)

where x is Frames Per Second (FPS)[48] and ms is milliseconds. These two equations show that the *Update* function is expected to execute every 16.67 ms in an application with a framerate of 60. For an application exhibiting a framerate of 30 FPS, the *Update* function is expected to execute every 33.3 ms.

Of course, a developer should strive to maintain a constant framerate in an interactive software application, but the framerate of an application depends on the processing power of the system hosting it.

Although *StartCoroutine* functions are executed after the *Update* function, the location in code whereby they return control to normal execution is determined by the developer. The *WaitForSeconds* and *WaitForFixedUpdate* functions in a script's lifecycle determine where the coroutine should return control to the executing script. The definition of a Unity coroutine is provided in chapter five in a discussion of the transmission of coordinates to the ImmerGo spatialization server. However, due to their importance in the implementation of the Coordinate Transmission component of the immersive audio capability it is worthwhile noting how they operate.

Unity describes coroutines as methods that "allow you to spread tasks across several frames. In Unity, a coroutine is a method that can pause execution and return control to Unity but then continue where is left off on the following frame" (Unity Technologies, 2021). [49] This then enables a process to be executed at a set interval that is independent from the framerate of the application. Indeed, the same functionality can be implemented in either the *Update* or *FixedUpdate* functions, but a coroutine provides a useful way of encapsulating a process outside of these functions. In the context of the VR application that was developed, a send coordinate coroutine facilitates the transmission of positional data to the ImmerGo server with an interval that is expressly set to 10ms which is a requirement of the spatialization system. If this process was performed in the *Update* function, then the transmission interval would be dependent on the application framerate. If the *FixedUpdate* function was used, then the transmission interval would depend on how regularly the physics system was set to update. This therefore indicates a model use for a coroutine, as the process is not dependent on either the framerate or the physics system time.

[48] The context in which the term FPS is used in this research, will determine its meaning. In previous chapters, FPS has referred to the digital game genre: First Person Shooters.
[49] https://docs.unity3d.com/Manual/Coroutines.html

Finally, the *OnDisable* function mirrors that of the *OnEnable* function, except that it is called when the component script is disabled as opposed to enabled. Indeed, the *OnDestroy* and *OnApplicationQuit* functions should be used for the deallocation of all allocated resources in the script (variables, etc.).

4.2.3 Unity's User Interface

This section will describe some of the essential "windows" of the Unity game engine. In this context, "window" refers to a rectangular component of the Graphical User Interface (GUI) that provides some or other information pertaining to the virtual Unity scene and/or application. In the context of the Unity framework, this GUI is said to be the "Unity Editor". The most used of these windows are the *Hierarchy, Scene View, Game View, Inspector, Project Browser,* and *Asset* windows. Of course, various layers of the Unity game engine such as the animation layer would also exhibit their own specific windows. Figure 56 indicates each of these primary windows that a user encounters when creating a new Unity project (application). The *Hierarchy* window is surrounded by a blue border at the top left of the figure. The *Scene* and *Game* windows are located at the center of the GUI and are surrounded by the purple border. A red border surrounds the *Inspector* window to the right of the image. An orange border indicates the *Project* window at the bottom left of the figure, and finally, the *Asset* window is shown with a green border at the lower half of the image. The *Game View* window is not visible in the screenshot below, but a developer can easily navigate to it by selecting its tab atop the central window (purple) in the figure.

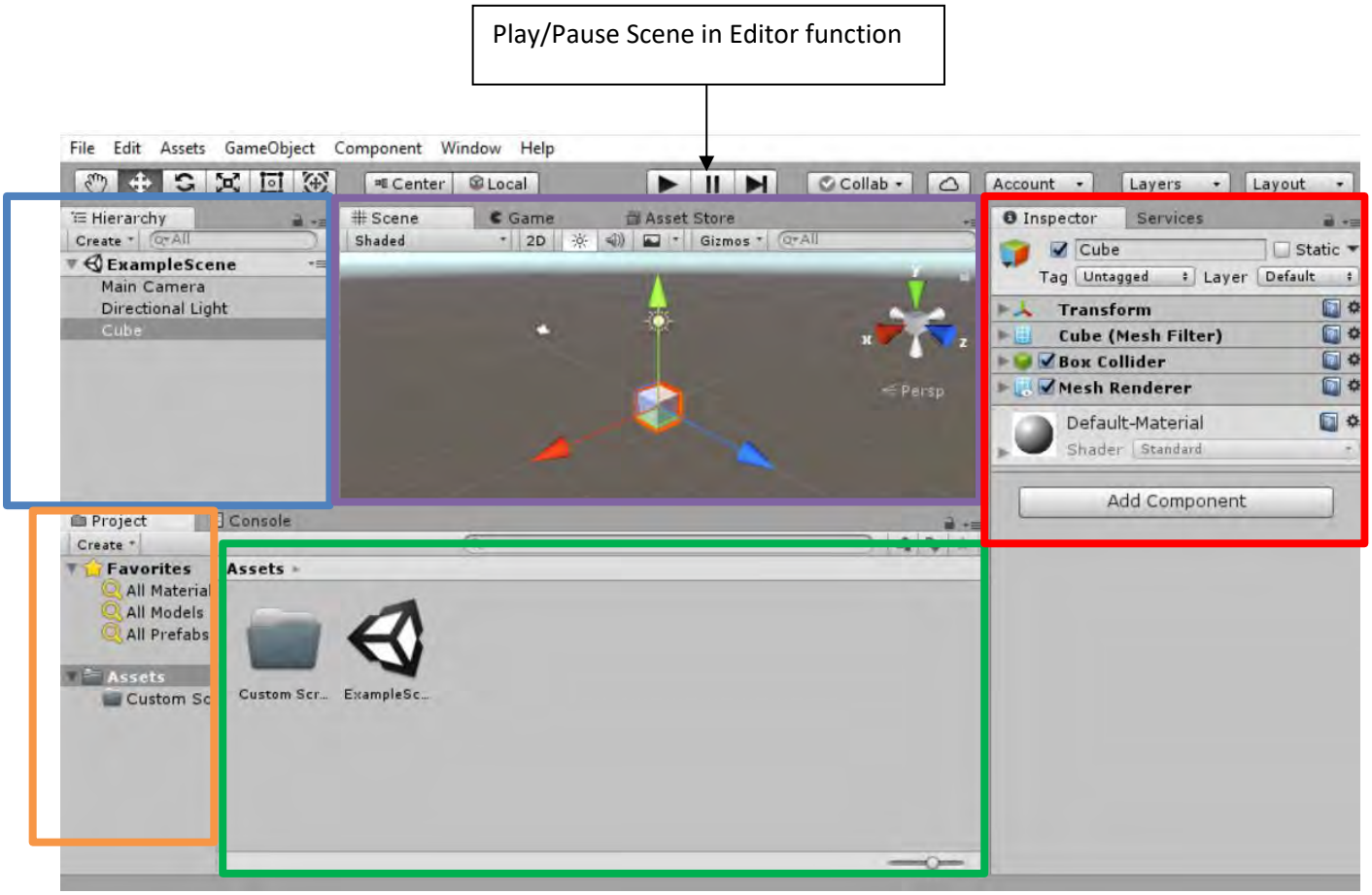

Figure 56: A screenshot of the GUI for the Unity Game Engine. The primary windows of the GUI are color coded.

The *Hierarchy* window indicates the objects that are present within the Unity scene. If one inspects the preceding figure, one should notice that that scene consists of three GameObjects, namely, a *Main Camera, Directional Light,* and *Cube* object. Of course, the more complex a virtual environment, which can generally be defined by an increased number of GameObjects and interactions in the Unity scene, the more populated the *Hierarchy* window. The more complex *Hierarchy* window (see Figure 57) exhibits Unity's parent-child model. Indeed, from the *Hierarchy* window, GameObjects that are children to parent objects are seen to exist beneath the parent object (see indented GameObjects). A developer can attribute a child object to a parent object by dragging an object in the *Hierarchy* on top of another object in this window.

Figure 57: An example of a Unity scene that is vastly more complex due to the number of game objects and their interaction mechanisms.

The *Scene* and *Game Views* are both windows that provide a graphical preview of the Unity scene that is under construction. The *Scene View* is essentially the scene builder window that enables a developer to select and position objects from any perspective in three dimensions. As such, it is an essential window that allows developers to build their virtual worlds. The perspective of the window is determined by the developer, i.e., they can view the scene from any perspective. This view is provided with traditional three-dimensional handle controls, and 'grid-snapping' enables the precise positioning of GameObjects in the scene. The *Game View* window provides a similar function to that of the *Scene View* window, except that this window offers the perspective of the main (active) camera in the scene. As such, it is a useful feature that allows developers to see how the virtual scene would be rendered at runtime—"What You See is What You Get" (WYSIWYG). When the application is played back inside the Editor, the *Game View* takes precedence over the *Scene View*, and essentially becomes a window that graphically represents how the application would function when compiled. It is therefore a means to preview the application before compilation (Haas, 2014).

The *Inspector* window is a crucial element of the Unity Editor that facilitates the inspection of specific GameObjects, and the *Components* that it might be comprised of. It thus indicates the details of a GameObject and enables the modification of information pertaining to the object. As one will recall, the only fundamental *Component* that a GameObject requires is its *Transform*. Every additional component provides additional features that the object should exhibit. An example of a more complex GameObject (many components) and how this affects the *Inspector* window is exhibited in the screenshot in Figure 58. If one observes the *Components* of this GameObject, it will be noticed that the first five *Components* pertain to the GameObject's position, shape, and appearance in the virtual scene. Indeed, the *Collider* and *RigidBody Components* provide the object with "physical" parameters so that it might interact with other GameObjects according to the parameters of the Unity physics system. The next two *Components* indicate that the GameObject is a light source in the scene, and the final four *Components* are all custom scripts that provide specific functions to the GameObject.

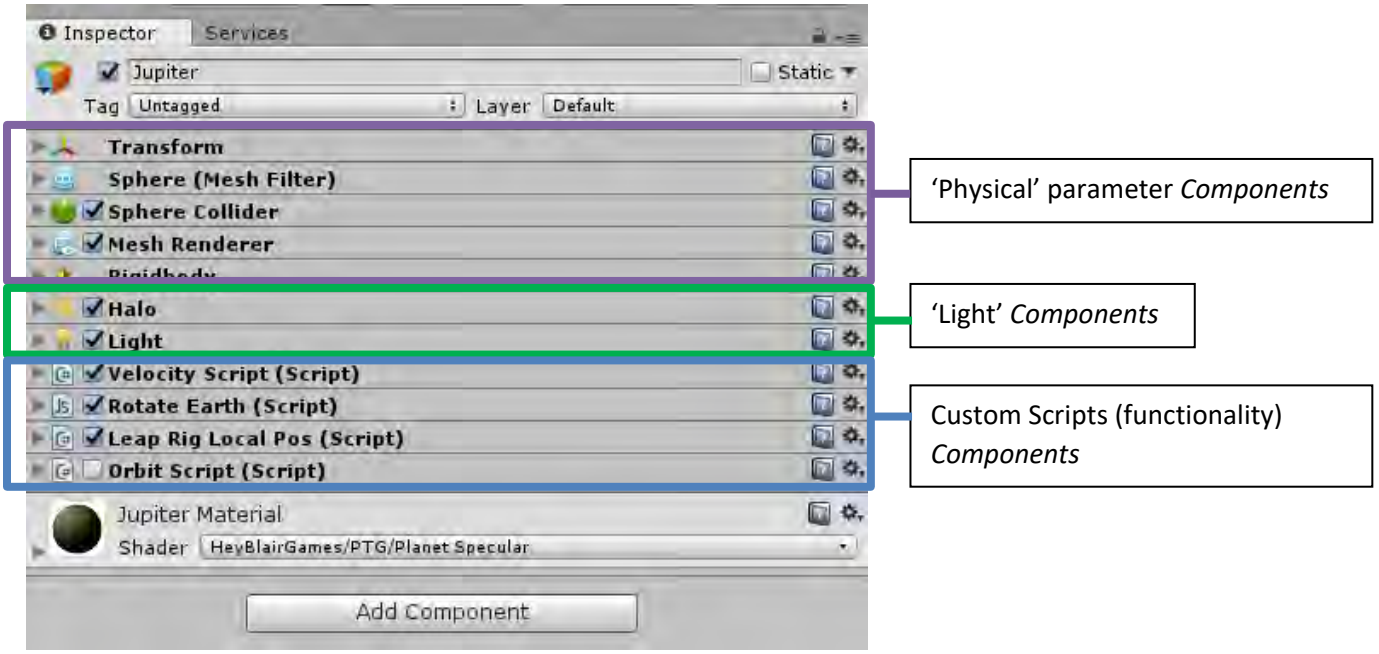

Figure 58: The Inspector window of a Unity GameObject called Jupiter. Beneath the name, all the components that this GameObject is composed of are indicated.

The *Inspector* is therefore a crucial element of the Unity Editor, as it enables developers to inspect the various *Components* that make up a GameObject in the scene.

The final two windows that are discussed are the *Project Browser,* and *Assets* windows. It should be noted that both these windows provide the same function, except that the *Project* window indicates the hierarchy of all the assets that are present within a Unity project, while the *Asset* window is used to inspect the contents of the assets indicated by the *Project* window. Both windows therefore indicate the assortment of Unity Assets that have been imported into the project.

4.2.4 Oculus VR Development in Unity

Having discussed the Unity UI and the attributes that enable the game engine it's functionality, the following paragraphs will discuss how proprietary software can be imported into the native engine. The OpenXR software Suite from the Khronos Group has enabled cross platform VR development. [50] This has realized a situation whereby a single SDK can be used for the creation of VR applications that are compatible across the host of commercially available VR HMDs. However, this shift in the VR development ecosystem has been recent, and as such, before Oculus' recent endorsement of this shift, the Oculus SDK was required by developers to construct VR applications that are compatible with Oculus devices. As such, the immersive VR application that was created for this project was developed with the utilization of the Oculus SDK. Essentially, developers need to ensure that the Oculus application, and requisite HMD is correctly configured on their host machine (see Figure 59). When developing standalone VR applications for Oculus headsets (i.e., applications for the Oculus Quest and Quest 2 HMDs), the Oculus application is installed on the device by default.

[50] https://www.khronos.org/openxr/

Subsequently, from within Unity, the developer is required to import the Oculus SDK (as a Unity Asset) into their respective project. This process is detailed in the following paragraphs; however, it is useful to recognize that the primary function of the SDK is to facilitate the communication between the Unity game engine and the Oculus VR application, thereby exposing an API to the developer that makes direct references to the Oculus HMD through the Oculus application and API.

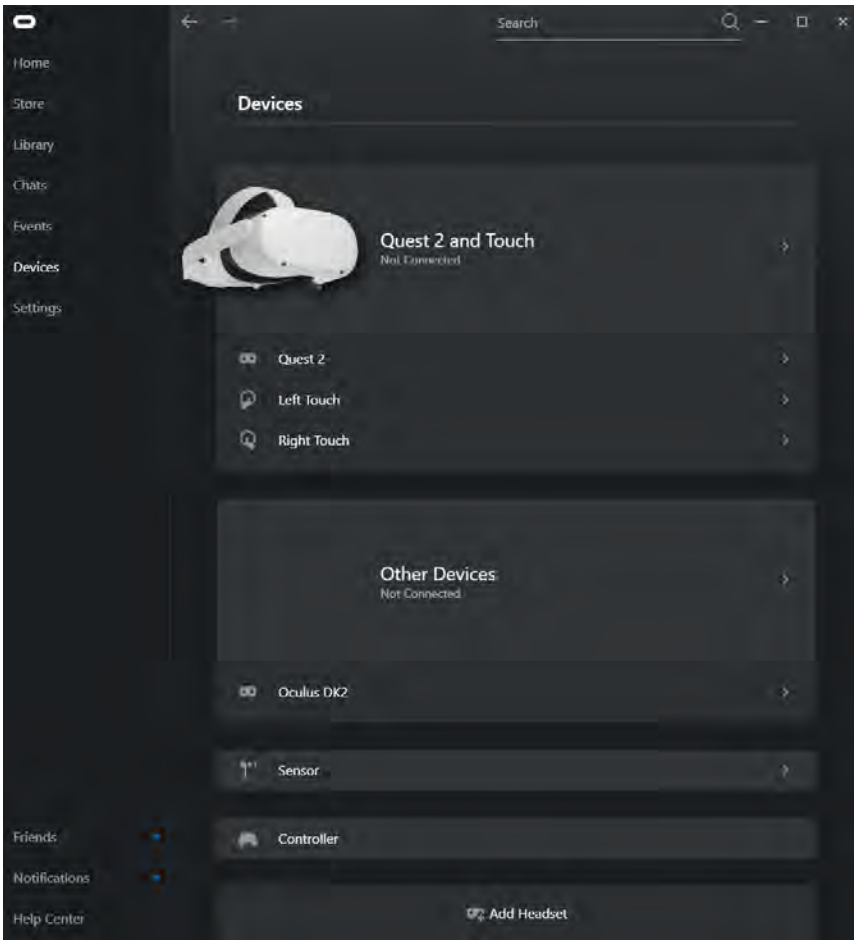

Figure 59: A screenshot of the Windows Oculus application, whereby the Oculus Quest 2 and Oculus DK2 can be seen as available (albeit not connected) HMD devices.

The Unity-compatible Oculus SDK is entitled the Oculus Integration Package, which is downloadable through the Asset Store window in a Unity project, or directly from the Oculus website.

Having successfully integrated the Oculus Integration package into a Unity application, the developer has access to the numerous scripts and prefabs that the Oculus SDK exposes to developers. Of course, most of this functionality is beyond the scope of this project, but it is important to recognize that the package also provides developers with working examples of primary VR functionality that the SDK provides. The developer can use these examples to better understand primary implementations of VR specific functionality, such as locomotion and interaction in VR. The most important prefab that this SDK enables is the *OVRCameraRig*, which was detailed in section 4.2.2.2.4. This Integration package provides a foundation which a developer can use to implement a variety of functionality within their Unity VR application.

4.3 AUDIO MIDDLEWARE

From the perspective of distributed computing, middleware can be defined as "software technology that enables the modular connection of distributed software" (Astley, et al., 2001). Although this definition presupposes a system designed for network applications, it nevertheless provides an analogous description of how audio middleware augments the audio layer of a game engine.

The initial conception of game engines envisaged the utilization of numerous different middlewares, each aiming to provide functionality that the developing application required (Nicoll & Keogh, 2019). More contemporary game engines provide a more unified presentation of all these middleware solutions. In the context of the audio layer or subsystem of a game engine, it has been suggested that certain aspects of the native audio engine can be difficult for audio developers to come to terms with (Horowitz & Looney, 2014). However, before formalizing the distinct features that audio middleware provides game engines, it is worthwhile to briefly discuss the need for more complex manipulation of audio in game development.

Historically, sound designers would create audio for a digital game or application using authoring tools such as Digital Audio Workstations (DAWs) outside of the game engine. Once the audio had been created, the sound designer would provide a game programmer with these sounds (in some or other file format) to subsequently deploy and manipulate the audio according to the requirements of the virtual scene, i.e., within the game engine (Horowitz & Looney, 2014). Of course, this process was suitable for simple digital games, but as the complexity of digital audio games increased, the inefficiencies of this process become more and more pronounced. As such, the creation of audio middleware solutions sought to enhance the efficiency of this process. These solutions "allow the (sound) designer, who may not be a programmer, to have more control over how, when and where their sounds are triggered in a game" (Horowitz & Looney, 2014).

This usefulness may be easily understood if one were to consider the process of deploying audio into a virtual scene. Historically, as already noted, a sound designer would simply author the sound and provide it to the programmer. After this, the programmer would decide how the audio should be triggered/manipulated according to parameters in the scene. When audio middleware is employed, this scenario changes significantly, as the programmer can notify the sound designer of the parameters in the scene that will influence how the audio should be triggered/manipulated in the digital game. As such, the sound designer will create the sounds needed for the virtual scene, and in addition will manipulate the audio using parameters that the audio middleware enables.

Once these audio middleware audio events have been authored, they are provided to the programmer, whose job is significantly simplified, as they need only provide the audio event with the 'actual' parameters in the game (i.e., providing 'hooks' to the audio event) that trigger the audio. This is because the sound designer has already facilitated the manipulation logic of the audio event (Horowitz & Looney, 2014). This process can be abstracted into a simple solution whereby one can attribute audio middleware as a software process that converts the concept of a sound into an audio event (Somberg, 2016).

A further advantage of audio middleware is that a specific audio middleware solution is generally compatible with numerous different game engines, and vice versa (Nicoll & Keogh, 2019). This benefits the audio production process in digital games as a sound designer or game audio developer does not

need to necessarily understand and operate all the different game engine solutions. Instead, they need to have a mastery of audio middleware that does not necessarily consider the differences between the various game engines. The functionality of the audio middleware remains consistent across these different frameworks within which digital games and applications can be developed.

The primary function of middleware mirrors that of a DAW, except that the resultant audio tracks are grouped into events, and the total audio project is built into sound banks that can then be referenced from within a game engine. Therefore, middleware should be seen as a software solution that sits in between the creation of the audio, and its deployment into the virtual scene/digital game (Horowitz & Looney, 2014).

The following diagram indicates the difference between DAWs and audio middleware, while also showing the domains in which sound designers and game programmers operate using these different software solutions. In this diagram, the audio sub-system is presented as being discrete from that of the game engine to provide more clarity on the differing workflows of audio for both sound designers and game programmers. Of course, the reader should recognize that a similar game development production pipeline exists for all sorts of other middleware solutions that might focus on other elements of the game development process, such as three-dimensional modeling of characters or game objects (Horowitz & Looney, 2014).

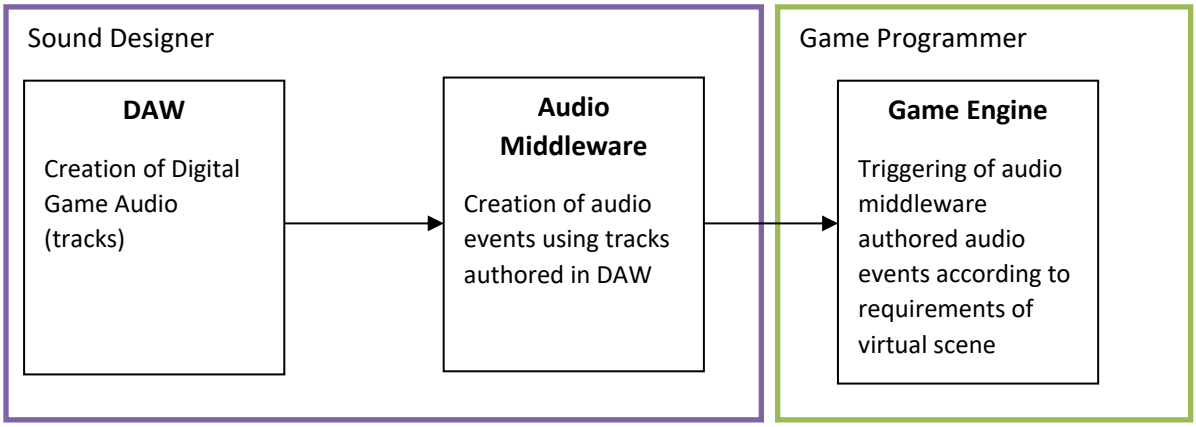

Figure 60: A diagram indicating the software domains of the sound designer and game programmer.

4.3.1 The FMOD Audio Middleware Solution

This section will introduce the FMOD audio middleware solution and thereafter detail the important components of this software solution. Given that the manner in which middleware augments and facilitates better audio control for Sound Designers and Game Programmers has already been discussed, the specific audio control that FMOD enables will be investigated. FMOD has been defined as "a cross platform audio runtime library for playing back sound for video games". Indeed, it has been suggested that FMOD set the standard for audio middleware editing, particularly its mirroring of classical DAW functionality (Horowitz & Looney, 2014). Essentially, the FMOD audio middleware solution exists in two discrete, albeit communicating, systems.

4.3.1.1 The FMOD Low Level System/API

This system implements the triggering and playing back of events and provision of the fine control of FMOD audio data, and numerous other functionalities besides. As such, it is called the FMOD Low Level

System/API. The Low Level API “gives access to the fundamental abilities of loading and playing sounds, creating DSP effects, setting up FMOD channel groups, and setting sample-accurate fade points and start/stop times.” As stated by FMOD, “the Low Level API is a programmer API that is intended to cover the basics/primitives of sound. This includes concepts such as ‘Channels', 'Sounds', 'DSP', 'ChannelGroups', 'Sound Groups', 'Recording' and concepts for 3D Sound and occlusion. It is standalone and does not require any sound designer tools to interface with. The features are all implemented by the programmer in code” (Firelight Technologies Pty, Ltd, 2021).

The Low Level API is largely composed of C++ libraries that provide efficient memory management and low-level system calls. In addition, the language is supported by a robust function library, which is highly advantageous for audio processing and audio driver functionality and communication. The Low Level systems function specifications are presented in C++ for the most part. However, the Unity integration package, which is detailed in section 4.3.2, is implemented in C# (which is the standard language for Unity scripting). The Low Level system provides function handles, which a Unity developer can reference in their custom scripts and application implementations.

4.3.1.2 The FMOD Studio System/API

The second system is the FMOD Studio (High) Level System. This system exists above the first (low level) system and is used to facilitate the functionality provided by the FMOD Studio Application (which is detailed below). As such, it primarily focuses on enabling the construction of distinct FMOD data types such as FMOD audio events, which are complex (high-level) data types that the Low Level System does not make explicit references to. In essence, the Studio API enables “programmers to interact with data driven projects created via FMOD Studio at run time” (Firelight Technologies Pty, Ltd, 2021). The system was developed to facilitate the FMOD Studio Application functionality, and the user interface it employs.

The Studio API provides the visual abstraction of importing audio Events (tracks) into the FMOD Studio Application, the binding of filters to these events, and the grouping of these events into channel groups, etc. These processes are visual abstractions of processes performed by the Low-Level system. However, these abstractions are useful, as they enable sound designers to perform functionality that would otherwise belong to the domain of a programmer. As indicated by FMOD, the Low Level API (system) can be utilized by developers without the need of the Studio API at all. However, the Low Level system is unable to load sound banks or play studio-authored FMOD audio events, which therefore indicates the need and importance of the Studio API (Firelight Technologies Pty, Ltd, 2021). Before discussing the integration of the FMOD Audio Middleware framework with the Unity framework, the Studio Application will be introduced by analyzing the GUI that sound designers employ, whilst also describing the processes that the FMOD Studio Application enables (i.e., FMOD audio event creation, sound bank assignment, and the attachment of DSP effects to the decks of FMOD audio events).

4.3.1.3 FMOD Studio Application

The Studio Application exists as the interface between the sound designer and the game programmer. The following features will be described.

- The construction of FMOD audio events.
- The binding of DSP plugins to these audio events.

- The configuration of the FMOD project according to the requirements of the Unity project and audio driver.

- And finally, the building of the FMOD audio events into sound banks that are referenced from within Unity.

Figure 61 and Figure 62 show the primary windows of the FMOD Studio Application. In Figure 61, the FMOD project is an example project, and, as such, no FMOD audio events have yet been created. However, the blue box indicates the location in which FMOD audio events are created, the assignment of audio events to audio banks, and a library of audio assets that have been imported into the FMOD Studio project. The designer can create these events in various ways, and Figure 62 shows a project that is populated by FMOD events, which can be seen on the left-hand side of the image.

Figure 61 indicates the track's timeline, which is surrounded by the green square. The red square indicates the track *Deck,* which is where a designer would attach DSP plugins or effects to the audio event. Figure 62 shows that the "EarthO" FMOD event has a Fader effect attached to its *Deck*. The yellow blocks in both figures describe the track *Overview*, which indicates the parameters of the effect DSP plugins that might be attached to an FMOD audio event. The application also allows for the addition of notes and tags to an FMOD event (Robinson, 2019).

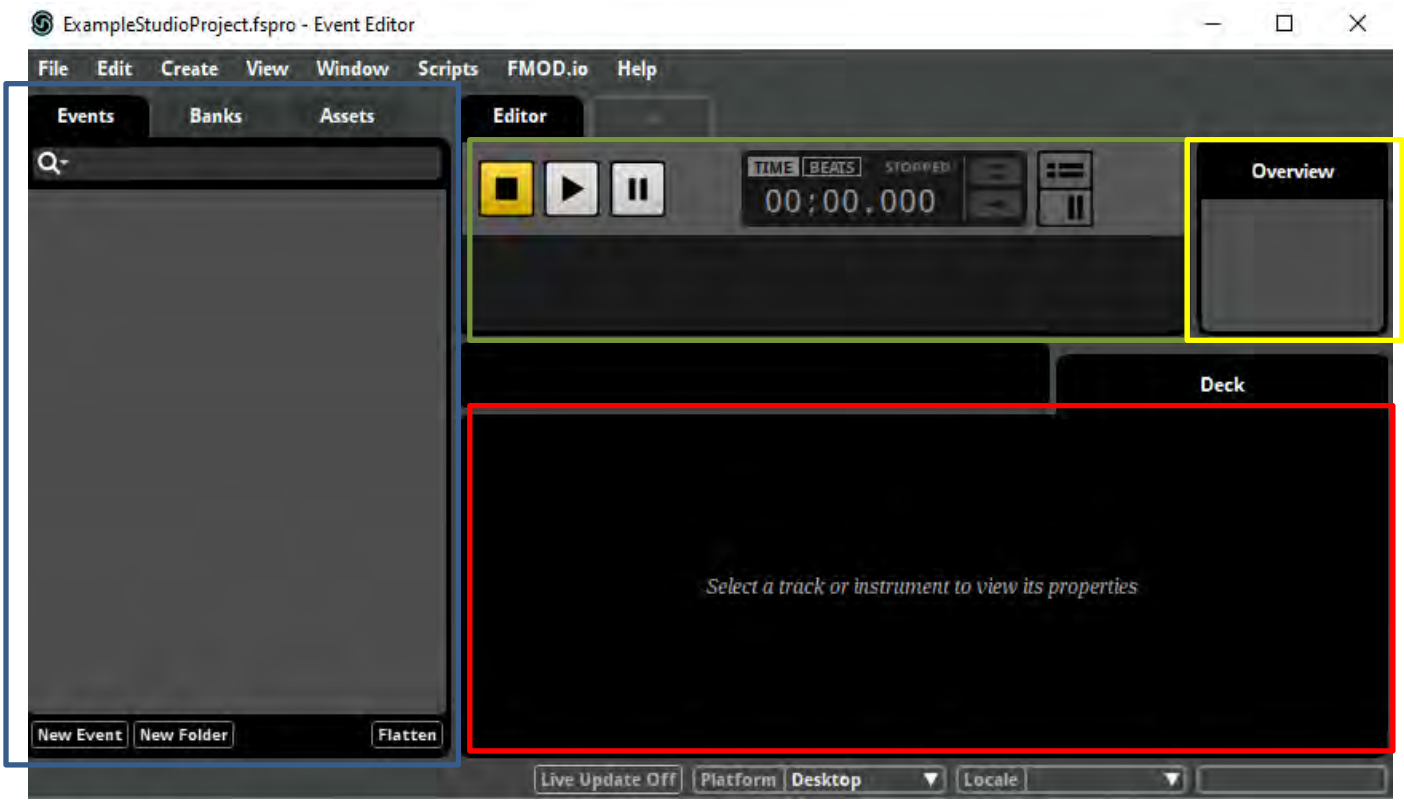

Figure 61: A screenshot of an empty FMOD Studio Application project, which indicates the different windows that the application is composed of.

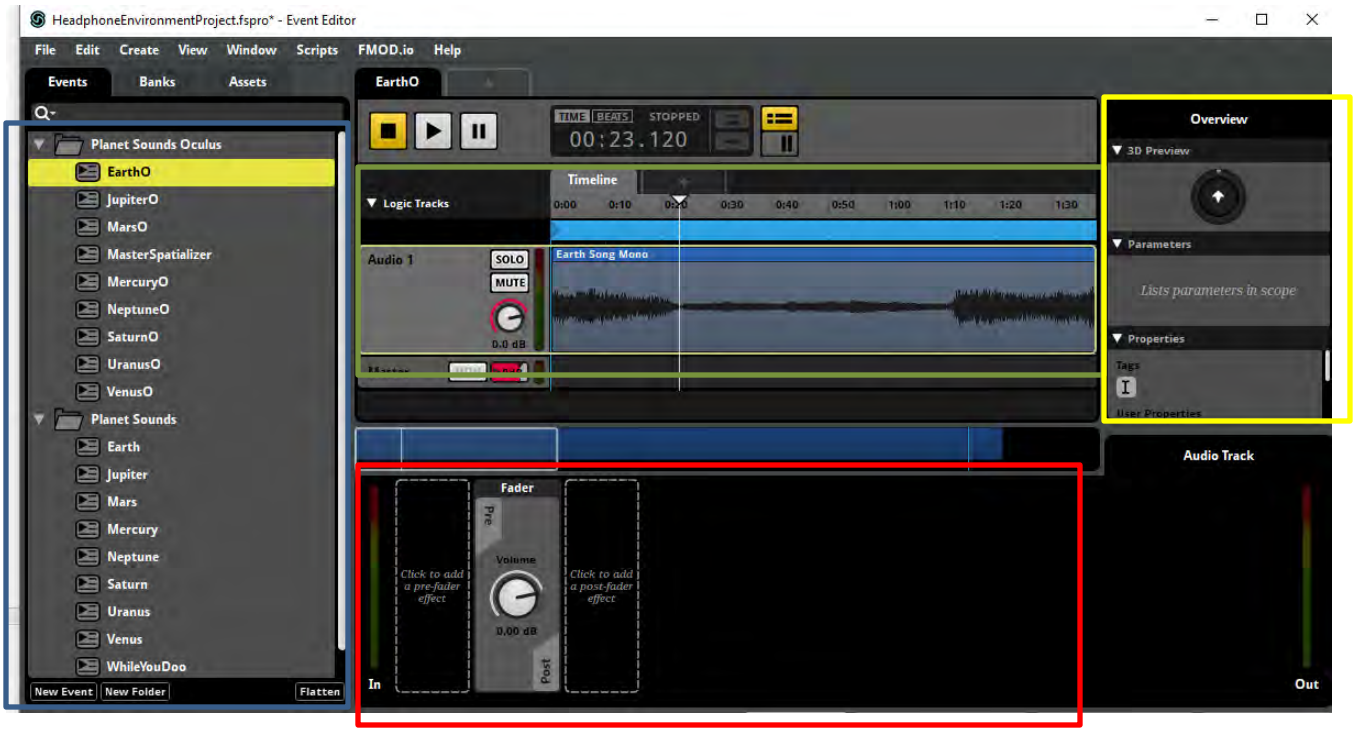

Figure 62: In contrast to the preceding screenshot, this figure indicates an FMOD Audio Project that Is populated by FMOD audio events.

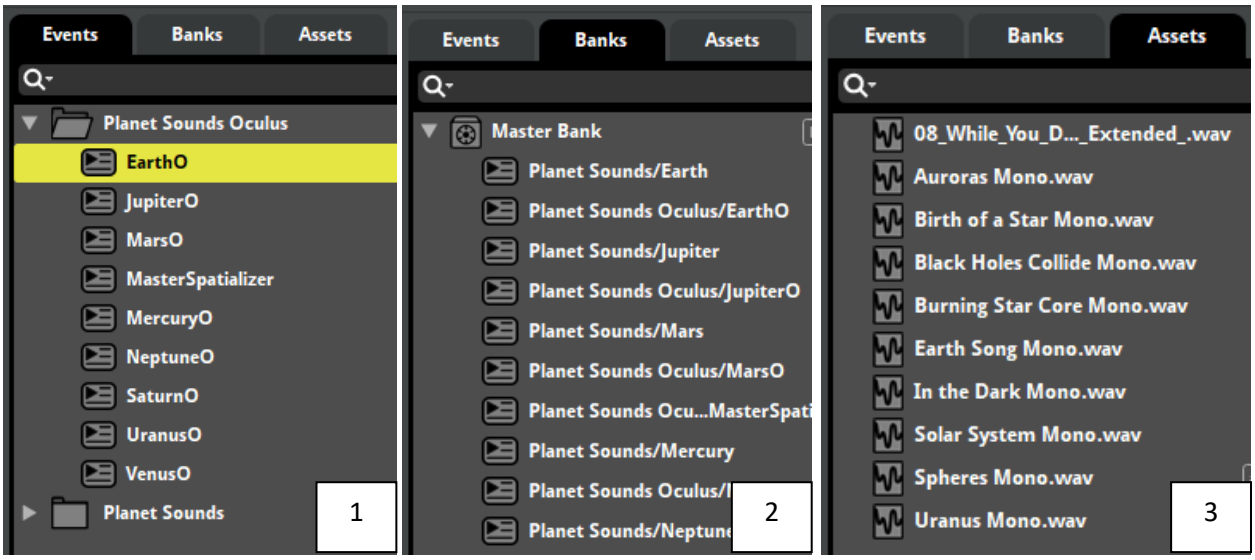

Figure 63: Screenshots of The Event, Bank, and Asset windows are all indicated in the left-hand portion of the FMOD Studio Application.

Figure 63 indicates the windows of the FMOD Studio Application that provide designers with the ability to construct and author FMOD audio events (see label 1), populate these events with audio tracks (*Assets,* see label 3), and finally assign these authored events to particular sound banks of the project (see label 2). Sound designers will import audio that has been created for the digital game or VR experience from a DAW, or another type of system. After which, they create FMOD audio events, which they assign to sound banks that are accessible to the game programmer from within the Unity engine.

4.3.2 The Integration of Unity and FMOD

This section will describe the integration processes of the FMOD audio middleware and Unity game engine frameworks. Figure 64 illustrates how the two software frameworks function in unison. The left side of the diagram shows the workflow of game audio production when using FMOD, while the right side provides an abstract view of the workflow of the Unity game engine. The dotted line provides a useful designation of content generation tools, i.e., the *Editor* applications for each framework (which are located above this line), while everything below this line indicates the core logic that is employed by both application *Editors* and also at runtime, once the game has been compiled into an executable application.

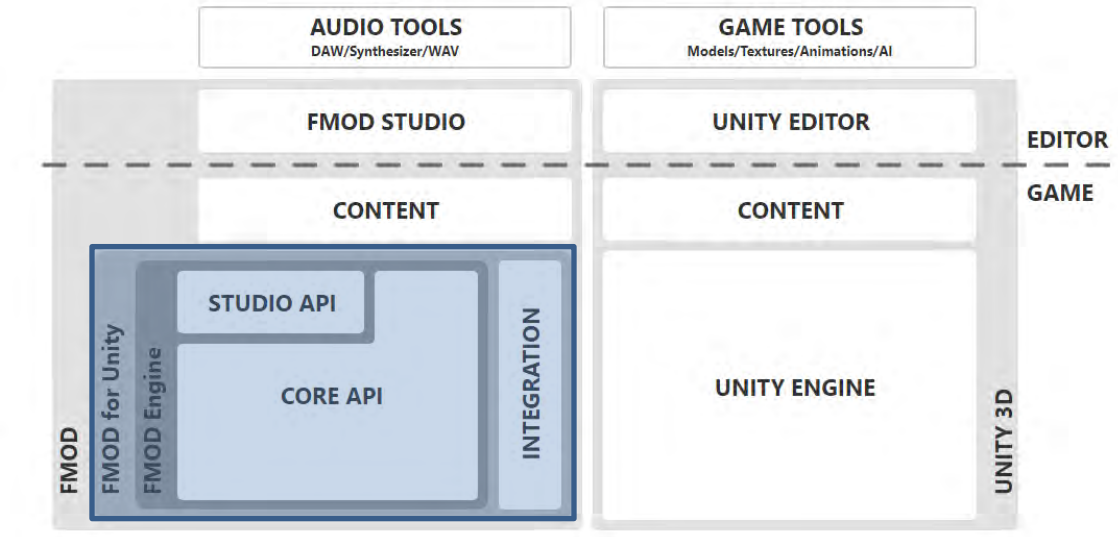

Figure 64: A diagram that indicates the two software frameworks used in the game development production pipeline (Firelight Technologies Pty, Ltd, 2021).

The process by which FMOD and Unity are integrated is the same as that through which the Oculus SDK is integrated into the game engine. A developer needs to find the 'FMOD for Unity' package through the Asset Store window in the Unity editor and download and import the package into their project. The integration package should be viewed as a wrapper or interface that exposes the core functionality of the audio middleware solution from within the Unity game engine framework. Therefore, the most important function of the integration package is to enable audio that is produced in the FMOD Studio Application (by sound designers) to be accessible to game programmers from within Unity. As such, a developer needs to provide a reference to the 'built' sound bank for a particular FMOD studio project. This bank would include all the relevant audio events that the sound designer has authored for the game. As indicated earlier on in this chapter, the game programmer then has the responsibility to correctly implement and trigger these audio events within the virtual world.

Upon the successful integration of FMOD into a Unity project, an FMOD tab will be available for navigation via the menu bar in the Unity Editor. This tab allows a developer to easily browse through FMOD audio events and enables FMOD settings such as referencing the correct FMOD Studio Project and FMOD sound banks through the Inspector window (see Figure 65). Via the FMOD settings and *Inspector* window, various debugging and troubleshooting tools for the Unity developer, as well as platform specific playback settings such as the sample rate, and speaker mode, can be altered. These

playback settings are defaulted to the settings that the sound designer decides on within the FMOD Studio Application, but the game programmer can change these parameters should they need to.

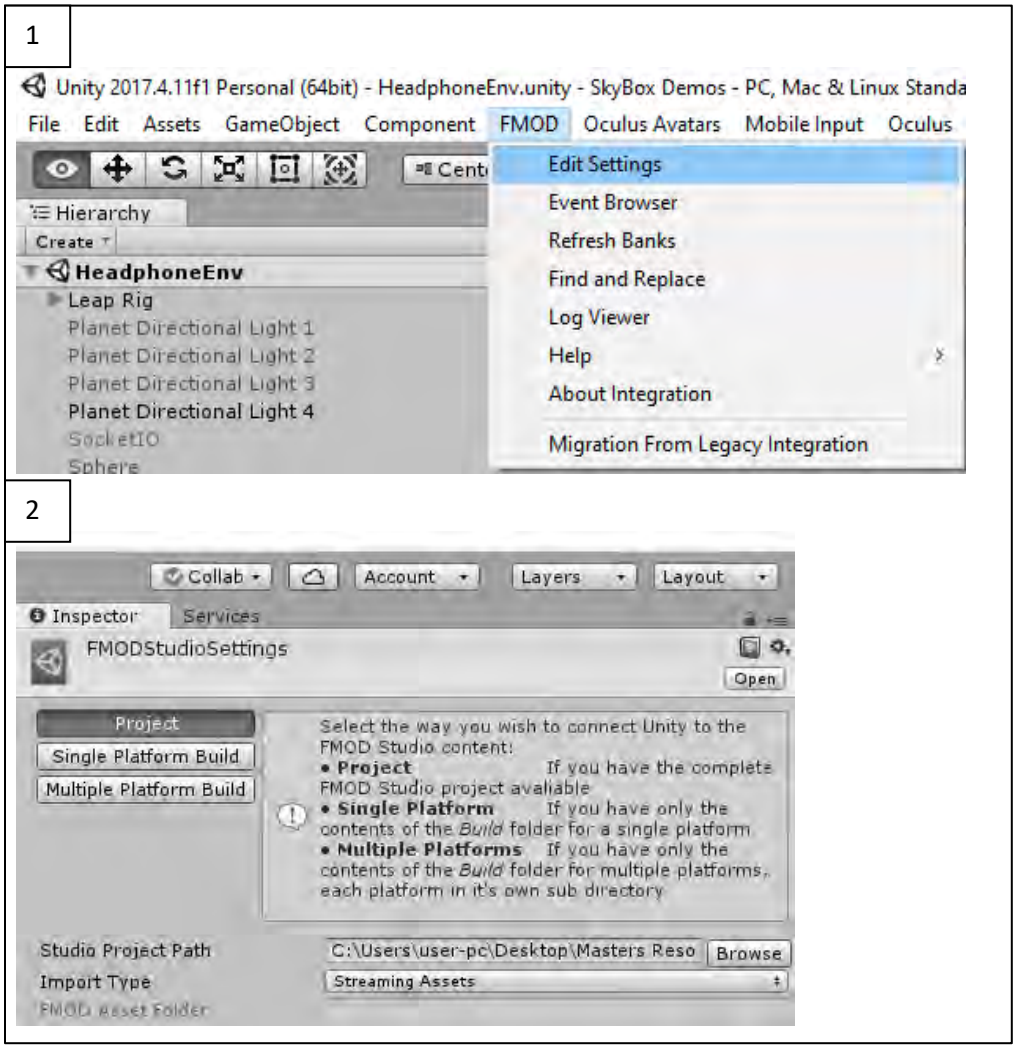

Figure 65: 1) A screenshot of the FMOD tab within the Unity editor. 2) A screenshot of the Inspector window of the editable FMOD settings that are exposed to the Unity game programmer.

There is one initialization requirement when using FMOD audio in Unity game development, namely that Unity audio needs to be disabled so that the application bypasses the default audio engine provided by Unity and uses the FMOD audio engine instead. Once this has been accomplished, the audio for a virtual environment or scene requires an observer or listener. Traditionally, in first-person game audio development, the primary camera in the scene is this 'listener'. This concept mirrors the visual requirements of a game—the perspective of the virtual scene that is being rendered to the output display is from the point of view of the user. Similarly, the audio 'listener' is the equivalent observer for the game audio.

In accordance with this requirement, the FMOD integration package provides a *FMOD Studio Listener* script that the developer should attach to a game object as a Unity *Component*.[51] This script

[51] https://www.fmod.com/resources/documentation-unity?version=2.02&page=integration-tutorial.html

communicates with the underlying FMOD audio engine and provides this audio engine with the three-dimensional position of the listener in real time through the Unity *Update* execution function (see 4.2.2.3.1). The listener position is determined by the *RigidBody* of the Unity game object of which the script is a component. In certain scenarios it is desirable for multiple listeners to exist within a virtual scene, and in these instances, the developer should provide each listener in the scene with a unique identifier (or index). The figure below indicates the *FMOD Studio Listener* script attached to the primary camera in the scene (see Figure 66).

Figure 66: A screenshot of the Inspector window of the 'MainCamera' GameObject of which the FMOD Studio Listener is a component with an index value of 0.

Of course, for the audio listener to detect audio in the scene, the virtual scene requires virtual audio *sources*. When using the FMOD Audio Middleware solution, a virtual audio source is implemented in the scene by attaching the *FMOD Studio Event Emitter* script as a *Component* to the Unity game object. This script uses of specific FMOD types that enable a developer to select an FMOD audio event from the appropriate sound bank of the project, while also determining when the event should start playing, and when it should stop (i.e., when it should be triggered).

Figure 67 provides the *Inspector* window for the *Studio Event Emitter* before an event has been selected, and afterwards. The lower image indicates the Play trigger (or in-game event) being called when the object to which the FMOD audio event is attached, executes its *Start* function. The Stop trigger (event) is set to occur when the Game Object is destroyed. The reader should realize that there are a whole host of different types of triggers. Common FMOD audio event triggers could be collisions between Game Objects, or when a Game Object inhabits a particular location in the scene. Alternatively, the triggers can be time-based, i.e., at the beginning of a game, a sound is generally triggered after a certain amount of time elapses.

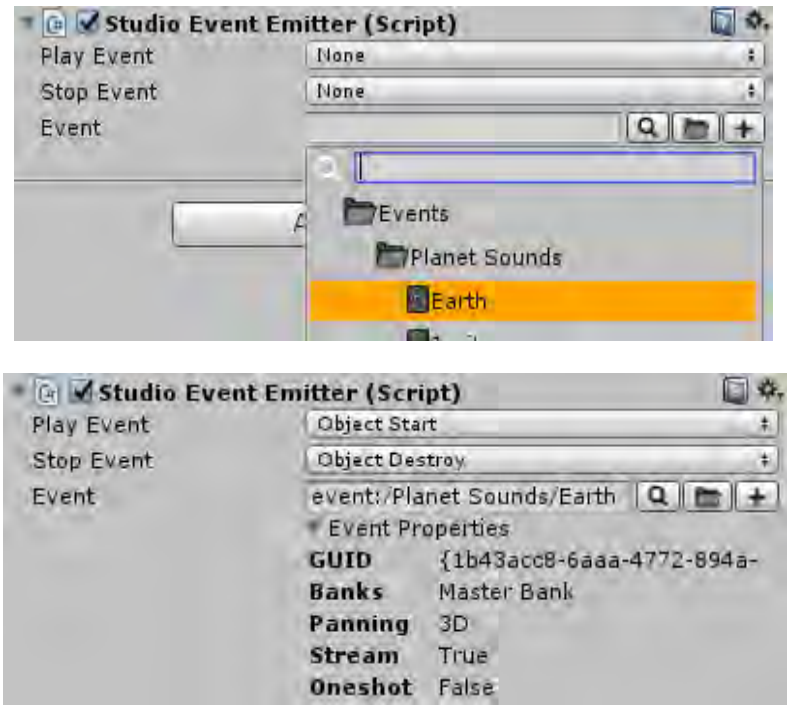

Figure 67: Two screenshots that indicate the same Studio Event Emitter Component attached to a game object. In the top image, an FMOD audio event in the sound bank is being selected the 'Event' variable of the component. The playback event conditions (playing and stopping) have not been provided. The bottom image shows an audio event has been selected, and the playback conditions have also been provided.

While the previous paragraphs indicate the default and native support that the *FMODUnity* integration provides game developers, more importantly, the integration package also allows developers to reference and utilize both the Low Level and Studio APIs in their custom scripts. A developer should simply include the *FMODUnity* using directive at the top of their script. An example of this is provided in the following code block.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Runtime.InteropServices;
using FMODUnity;
```

Code Block 2: An example of the Using directives in a Unity script. As can be seen, the FMODUnity directive is included in this instance.

Figure 64 indicates this namespace that Unity scripts can reference. In the diagram, this is indicated by encompassing the *FMODUnity* integration in a blue box. As can be seen, both FMOD APIs exist within it. The *Listener* and *Emitter* scripts provide the general functionality that most programmers would require in their projects. However, without direct references to specific API functions, specific audio control would not be possible. The employment of the *FMODUnity* namespace enables access to these references and accompanying audio control.

The breadth of the control and customizability that the APIs enable is beyond the scope of this thesis. However, the specific audio control requirements and implementation decisions that were made

according to the goals and objectives of this research are detailed in the following two chapters, i.e., triggering FMOD audio events directly from code and, more importantly, binding custom DSP effect plugins to FMOD audio events. The custom FMOD compliant DSP plugin that was implemented was used to route audio out of Unity and to an external ASIO playback server.

5 CONSTRUCTS TO ENABLE FMOD, UNITY, AND IMMERGO COLLABORATION

"I was taught that the way of progress was neither swift nor easy."

— Marie Curie

This chapter discuss how the implementation of an immersive audio capability was developed on and for Windows computers. In so doing, it will elaborate on how this capability realized one of the secondary goals of this research: to enable speaker-based immersive audio to be utilized within Unity game engine development. This therefore entailed the integration of both the Unity game engine and the ImmerGo spatialization system frameworks. The Unity framework does not natively support speaker-based immersive audio, and thus the need for such a capability arises. Indeed, Audio Middleware solutions require third-party modules and software to facilitate speaker-based immersive audio for irregular speaker configurations (arrays consisting of more than twelve speakers). The capability that has been developed represents one such third-party module, and therefore provides means to augment the Unity game engine with speaker-based immersive audio functionality, whilst also enabling the advanced audio signal processing that Audio Middleware provides. The realization of this capability, and its utilization in an immersive VR application (detailed in chapter six), enabled the attainment of the primary goal of this research.

Since the requirements and desirability of the immersive audio capability have been introduced, it is now possible to proceed to a discussion on its implementation context and how it achieves its functionality. The capability exists as three discrete components that communicate together to realize its needs. Each of these components will be introduced in the following paragraphs. The capability's needs are to enable immersive audio processing to be performed by the ImmerGo spatialization system from audio that is routed out of the Unity game engine framework. This enhances the potential for immersive audio to be used within the Unity environment for digital games and immersive VR applications, as the ImmerGo spatialization system supports a variety of immersive audio rendering techniques (which are reviewed in chapter three). As alluded to, the capability was also developed to operate in tandem with a particular Audio Middleware solution, namely the FMOD Audio Middleware software suite. This capability sought to combine specific audio processing functionality of the FMOD middleware solution with the localization functionality that the ImmerGo spatialization system provides. The ability to provide speaker-based immersive audio functionality necessitated overcoming a few fundamental problems, as the intent of this project was to spatialize audio events to speakers unconstrained both in terms of number and configuration.

Firstly, the audio (housed within Unity) needed to be transmitted to an Ethernet AVB network containing miniDSP NDAC-8 devices, through which the ImmerGo spatialization system functions. The NDAC-8s provide matrix mixing functionality (as mentioned in chapter two), which can be modified through its Firmware, and enables advanced control of the mixing of inputs and outputs according to the requirements of a developer, i.e., it can be used to facilitate more efficient implementations of audio rendering algorithms. The utilization of the advanced matrix mixer of the NDAC-8s were

requirements since FMOD by default uses its own mixer to spatialize audio events to a standard and constrained set of speakers.[52]

When considering this aspect, the ASIO SDK by Steinberg [53] was used to provide Ethernet AVB compliant audio streaming/playback. This encompasses the capture of audio being routed out of Unity, and then a streaming server (the ASIO playback server) to pick up the audio streams and forward these streams to an ASIO compliant driver. This playback server represents the first component of the immersive audio capability. The audio that is transmitted to the playback server needs to be provided by FMOD audio events attached to Unity game objects. Accordingly, FMOD audio middleware enables advanced audio processing functionality of audio events (tracks) and facilitates the development of bespoke Digital Signal Processing (DSP) plugins. This facility that FMOD provides was the primary reason it was selected as opposed to alternative middleware solutions. Accordingly, a plugin was developed that was able to slot into the DSP chain of FMOD audio events and transmit the audio samples of the events to the playback server, thereby rendering a solution to streaming audio for playback from the Unity game engine. Indeed, this ability to stream audio out of Unity represents the second component of the immersive audio capability.

In addition to the previous requirements, the localization of audio sources within a Unity game engine project needed to be accurately enabled. This entailed providing the ImmerGo spatialization system with the coordinates of game objects (to which the FMOD audio events, containing the bespoke DSP plugin, are bound) within the virtual scene, so that the locality of these objects could be correctly calculated using appropriate audio rendering algorithms. This final phase of the implementation required the positional values of the audio emitting game objects (GOs) to be transmitted via a socket implementation. To accomplish this, the primary functionality of the mobile ImmerGo client needed to be adapted to a modified version of this client embedded within a Unity scene, which represents the third and final component of the capability. A further requirement was to ensure that the unit measurements and cartesian coordinate systems in the game engine matched those provided by the ImmerGo spatialization system, so as to ensure virtual sound sources were within the bounds of the speaker array and were correctly spatialized. Figure 68 indicates the three components that the capability consists of, and how the capability serves to integrate both the Unity game engine and ImmerGo spatialization system.

[52] https://documentation.help/FMOD-Studio-API/FMOD_SPEAKERMODE.html
[53] https://www.steinberg.net/developers/

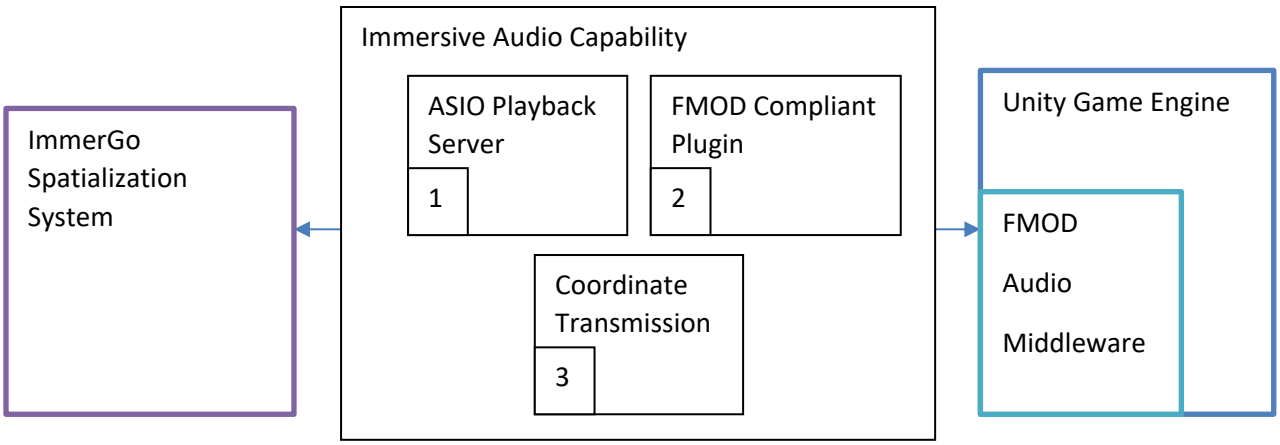

Figure 68: A diagram that represents the three components that make up the immersive audio capability. The components of the capability are numbered according to the order in which they have been introduced. Both frameworks are indicated on either side of the capability.

Having briefly introduced the capability and its components, it is now necessary to present the structure for the remainder of this chapter.

The chapter is comprised of four sections that indicate the implementation process behind the various components of the capability. The first section will provide a brief overview of the functioning of the capability when all the components are combined, thereby providing a holistic view of the core mechanisms. The second section will provide further information regarding the development of the three components that comprise the implementation. The third section will detail the mechanisms of the system, and how they operate within the contexts discussed in the first section. The final section will provide some context to the following chapter and how the immersive capability has been used in the construction of an immersive VR application.

This chapter assumes the reader is familiar with the functioning of game engines, and how audio functionality is provided to these engines. The previous chapter details how audio middleware is utilized from within game engines to provide audio processing requirements with which conventional digital audio workstations (DAWs) provide sound designers. Of course, audio middleware also enables specific "game" functionality that a DAW does not. Hence, the workflow of audio middleware and how it can be integrated into a game engine is also assumed knowledge. In addition, the reasons for the importance of spatial and immersive audio in digital game and VR application development have been outlined and emphasized across the opening four chapters.

5.1 A Holistic View of The Capability

This section aims to indicate how the three components that inform the immersive audio capability operate in tandem. Of course, the preceding section, which details the mechanisms of each component, should have provided the reader with a relatively comprehensive understanding of how they cooperate. However, the following discussion provides more clarity on this matter. During this discussion, figures that have been presented earlier in this chapter are elucidated from this holistic perspective.

Before detailing the communication and the dependencies that the components exhibit, it is sensible to recognize the core function of the immersive audio capability. The capability seeks to augment the Unity game engine with the ability to utilize speaker-based 3D/immersive audio in the development of digital

games and immersive VR application and experiences. Figure 69 provides a simple diagram that indicates the relation of the three components of the capability to the Unity framework. This critical function encompassed the integration of the ImmerGo spatialization system, the FMOD Audio Middleware solution, and the Unity game engine. As such, the capability should be viewed as a bridge that facilitates this integration. Indeed, there are other competitive solutions that provide similar functionality, but the resultant implementation provides a novel use-case for the ImmerGo spatialization system in the creation of immersive VR experiences that emphasize and benefit from speaker-based spatial audio. In addition to facilitating the development of immersive VR experiences that utilize both the Unity and ImmerGo frameworks, the capability enabled a platform by which the primary goal of this research to be achieved: that is, to analyze and determine the differences between immersive spatial audio implemented with a speaker and headphone-based approach in the context of VR applications.

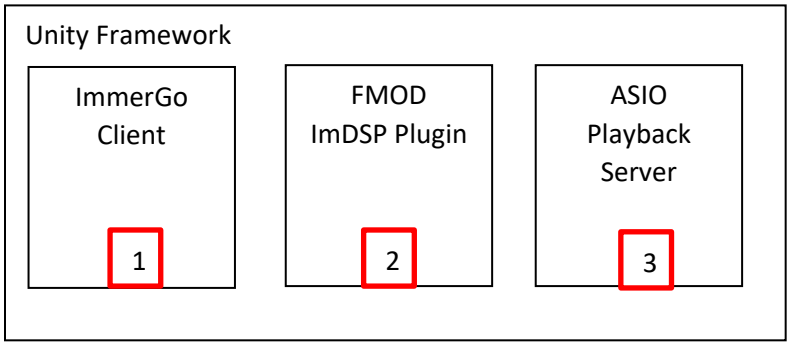

Figure 69: A simple diagram showing the three components of the developed capability. In this representation, the components have been enumerated according to how closely connected they are to the Unity framework: i.e., the ImmerGo client is embedded within the framework, while the ASIO playback server is discrete.

In providing an overview of the immersive audio capability, it is worthwhile to abstract the core functionality of the three components, and how each achieves the core function mentioned above. The core function itself can be broken into two separate functions. The first of these is for the capability to provide positional data for FMOD audio events, which are attached to game objects, to the ImmerGo spatialization server. In tandem, the second function of the capability needs to stream out the audio data (samples) pertaining to these FMOD audio events through an ASIO playback server, which, in turn, forwards the audio data to an Ethernet AVB network. The *Coordinate Transmission* component of the capability performs the first function, whilst the FMOD *ImDSP* and *ASIO Ethernet AVB Streaming Server* components provide the second function. This abstraction proves useful if one considers how it relates to the OBA paradigm (detailed in chapter three). The metadata of an audio source/object is transmitted to the ImmerGo server, whilst the actual audio sample data is streamed to the playback server via the *ImDSP* plugin. Figure 70 visualizes this OBA-based perspective.

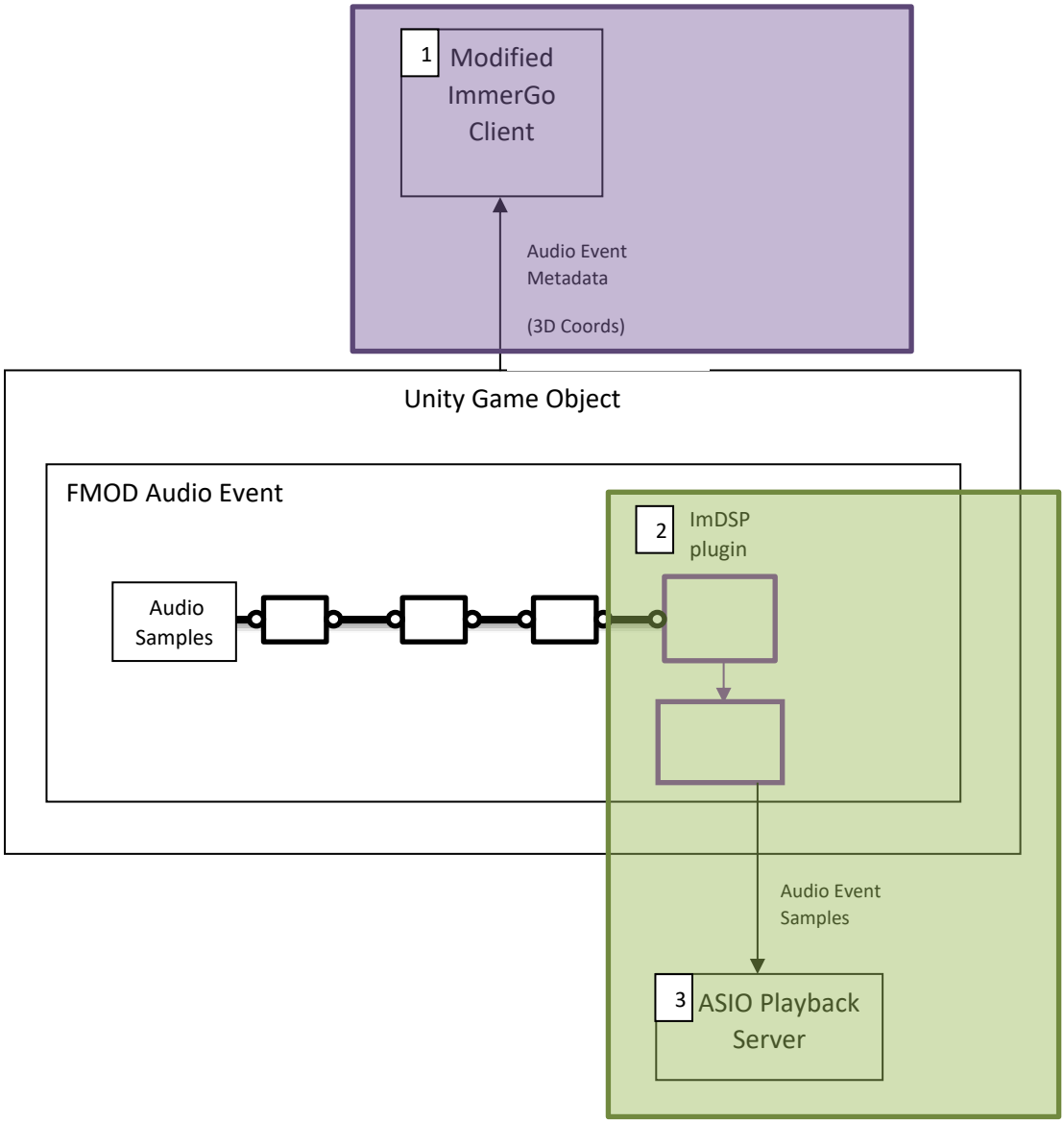

Figure 70: This diagram indicates how the primary function of the immersive capability can be abstracted into two subsequent functions: namely, the transmission of metadata pertaining to an audio source (illustrated by the purple area), and the streaming of the audio source's audio samples to the playback server (the green area).

In the proceeding diagram, the *Coordinate Transmission* component is represented by the number "1", while the *ImDSP*, and *ASIO Playback Server* are represented by the numbers "2" and "3" respectively.

If one is to consider the *ImDSP* and *ASIO Ethernet AVB Streaming Server* components, a brief overview of how they communicate using the multicast functionality of Winsock is pertinent. Figure 71 shows a representation of this communication model. In the diagram, four ImDSP plugins at the end of FMOD audio event DSP chains behave as multicast clients that all belong to a specific client group. The multicast server is contained within the ASIO Playback server and broadcasts a request for audio samples to the group. Upon reception of this request, each of the clients replies to the server with a set of samples that also additionally contain an identifier, which enables the server to recognize which client has sent the packet, and thereby fill the appropriate circular buffer

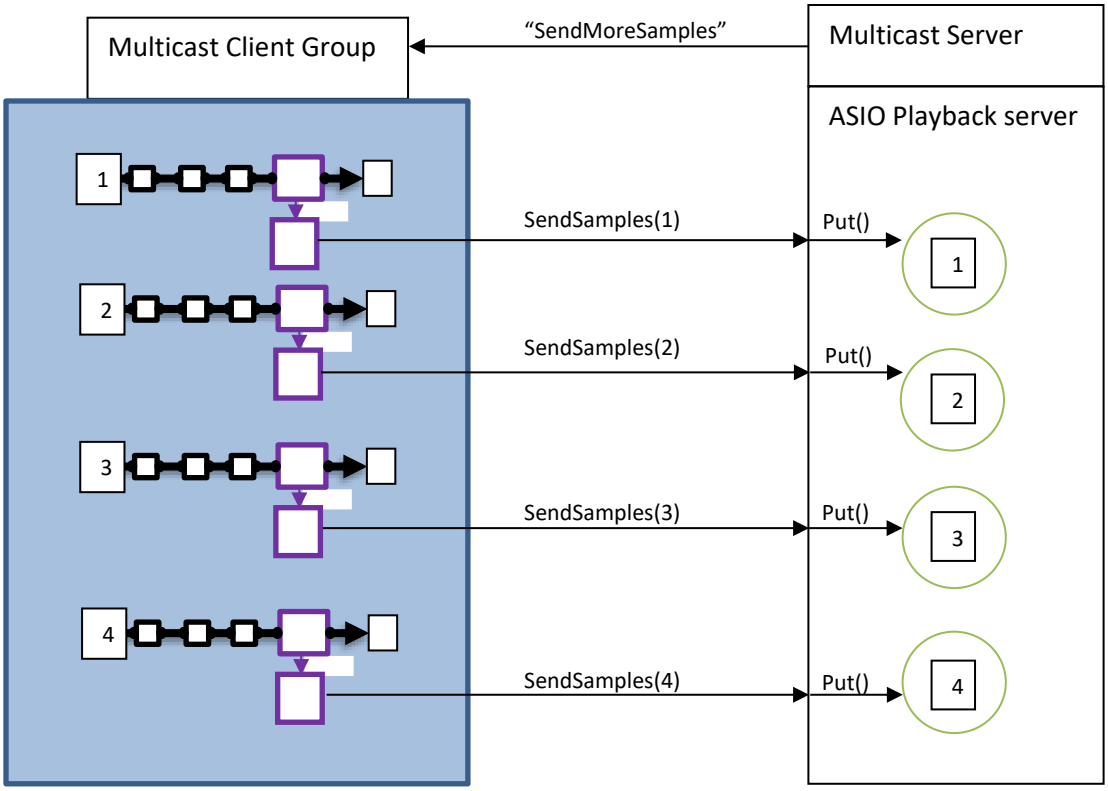

Figure 71: A diagram that indicates the multicast implementation that was used for the playback components of the capability.

Another aspect of the capability that should be examined is the requirement of keeping the identifiers across all components of the capability consistent. This forms a crucial element of the communicative functionality of the capability since it enables each component to be aware of which audio event is being positioned and played back. As such, it should be recognized that the association of an FMOD audio event to an instance of an ImDSP plugin is the crucial aspect in determining the unique identifier of this event. The numerical identifier of a specific audio event/object is provided to the instance of an ImDSP plugin at Unity application runtime. The use of the *setUserData*() function that custom FMOD DSP plugins provide is the way in which this identifier is assigned to the ImDSP instance (see section 5.2.2). This identifier is made consistent with the index of the array comprising of game objects (to which the FMOD audio events are attached) that the *SocketIOClient* script assigns. Accordingly, the ImmerGo server is aware of which output channel's mix levels need to be calculated. In addition, the identifier is also used to populate the requisite circular buffer on the Playback server, which, in turn, provides audio samples to the audio output channel of the ASIO AVB Driver pertaining to this buffer. It follows that the identifier of the FMOD audio event from within Unity remains consistent across all three components (see Figure 72).

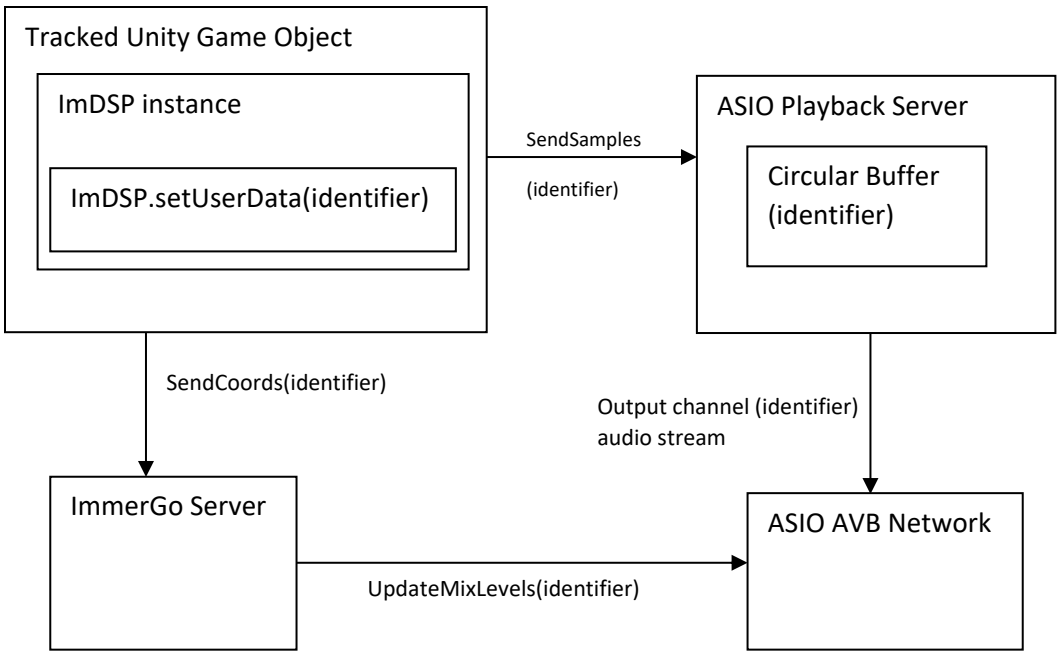

Figure 72: A representation of how the identifier of an FMOD Audio Event is kept consistent through all stages of the Capability

Having explained of how the capability and its components exist from an OBA perspective, and having discussed aspects of the component's communication requirements, it would now be useful to analyze in which framework (Unity or ImmerGo) each component exists. The schematic diagram presented below (Figure 73), which was also used in the Introductory chapter, provides a general representation of the capability, and its integration into both the ImmerGo and Unity frameworks. In other words, it provides a viewpoint of the capability bridging the two frameworks, and where the components of the capability exist in relation to the suite of applications used in their development. Of course, it should be noted that the Playback server exists externally from both ImmerGo and Unity frameworks, as it was developed using the ASIO SDK as a standalone console application. It could therefore be argued that the capability integrates three frameworks (the ASIO SDK, Unity and ImmerGo frameworks). Figure 73 indicates these different frameworks by color coding them.

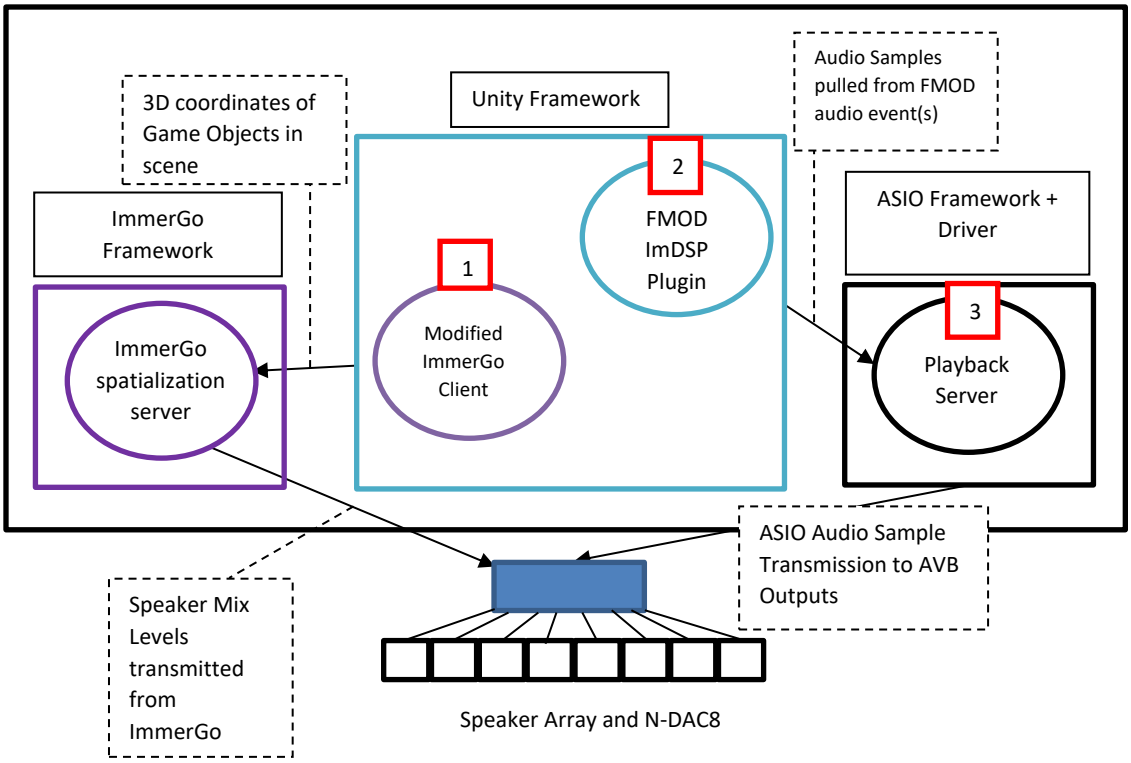

Figure 73: The diagram provides an overview of the capability and how its three components operate and communicate from within the ImmerGo (purple), Unity (blue), and ASIO SDK (black) frameworks.

5.2 Background Providing Context

As mentioned, this section will attempt to outline a few key aspects of the frameworks that were utilized in the developmental process. Section 5.2.1 provides readers with more detail on the platforms used to create the ASIO Ethernet AVB compliant playback server. Section 5.2.2 specifies the processes and structures needed to create an FMOD compliant DSP plugin, and finally, sections 5.2.3 and 5.2.4 indicate how Game Object coordinates function in Unity, and how the ImmerGo spatialization framework was tailored to the specific needs of the capability.

5.2.1 ASIO SDK and its Ethernet AVB Interface

To develop the ASIO Playback server, the ASIO SDK by Steinberg was utilized. [54] The SDK enables developers to utilize the audio streaming protocol for their specific needs. Developers use the SDK to construct an executable C++ console application that adheres to the ASIO API requirements and streams out audio to an appropriate ASIO audio driver—in our case the ASIO driver provided for the Echo Audio Ethernet AVB Streamware PCIe card. [55] As stated in the ASIO Output specification, "ASIO requires that the hardware manufactures provide a driver, which abstracts the audio hardware in a way ASIO can deal with" (Steinberg, 2019). The API employs a few constructs and an interface for initialization. Since not all these constructs and interfaces were needed in this implementation, only the crucial ones that were employed will be detailed to provide context to the following sections. The reason for the utilization of this SDK is that it provides "low-latency [audio streaming] and solves synchronization issues between the

[54] https://www.steinberg.net/developers/
[55] https://www.gearjunkies.com/2013/09/introducing-echo-audio-nic-1-pcie-network-adapter-and-streamware-sdk/

different input and output channels" (Steinberg, 2019). The developed playback server is Ethernet AVB compliant by virtue of using the Streamware ASIO driver and its associated Ethernet AVB Network Interface Card (NIC).

5.2.1.1 Implementation Guidelines

The ASIO specification outlines a few important implementation conditions that the developed application needs to adhere to. These conditions can be categorized as *Instantiation*, *Operation*, *Driver Query*, *Host Query*, *Audio Streaming*, *Media Synchronization*, and *Driver Notification* conditions that the developed application needs to implement (Steinberg, 2019). Before discussing these areas in more detail, it is useful to recognize that the return code for successful function in the ASIO SDK implementation is *ASE_OK* and *ASE_SUCCESS*. There is also a list of error codes that indicate unsuccessful calls and the reasons behind the failures.

5.2.1.1.1 ASIO Operation

A Finite State Machine (FSM) diagram of an ASIO SDK application is provided below (see Figure 74). An ASIO SDK application (or host application) exists in four states: *Loaded, Initialized, Prepared,* and *Running*. The *Loaded* state indicates that the ASIO audio driver has been recognized and is accessible from the host application for queries. The *Initialized* state indicates that the driver has been correctly instantiated and that the host application can accept queries. The *Prepared* state denotes that the audio buffers have been allocated correctly, and that the driver is ready to start receiving audio samples for playback. Finally, the *Running* state indicates that the application is executing and that audio streaming to the driver is taking place.

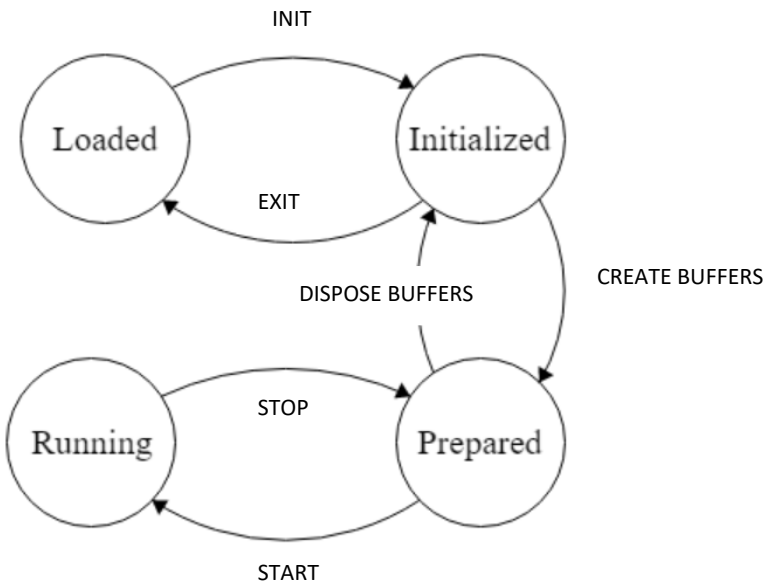

Figure 74: A Finite State Machine of the ASIO Driver that shows the states it inhabits and the functions that shift the system into the various states (Steinberg, 2019).

In accordance with FSM principles, various commands are used to shift the application from one state to another. In this instance, the commands used to shift the ASIO application's states are:

- *Init*: "Initialize the driver for use by the application. Optionally acquire hardware or load additional driver components. **ASIOInit()**". This command shifts the application from the *Loaded* state to the *Initialized* state.
- *CreateBuffers:* "Allocate memory for the audio buffers and allocate hardware resources for the audio channels. **ASIOCreateBuffers()**". The command is used to shift the application from the *Initialized* state to the *Prepared* state.
- *Start:* "Starts the audio streaming process. **ASIOStart()**". As expected, this command shifts the application from the *Prepared* state to the *Running* state.
- *Stop:* "Stops the streaming process. **ASIOStop()**". This command changes the application state from the *Running* state to the *Prepared* state.
- *DisposeBuffers:* "Deallocates hardware resources for the used channels and disposes the memory for the audio buffers. **ASIODisposeBuffers()**". This command shifts the application from a *Prepared* state to an *Initialized* state.
- *Exit:* "Deallocates any remaining resources and puts the driver back into the uninitialized state. **ASIOExit()**". Finally, the *Exit* command is used to return the application into the *Loaded* state from the *Initialized* state (Steinberg, 2019).

The diagrammatic representation of the FSM of an ASIO SDK application (see Figure 74) should be referred to with the following paragraphs. This will offer a less abstract interpretation of what is discussed subsequently. The following paragraphs detail how the constructs and interfaces of the ASIO SDK are utilized from a programmatic perspective (see Listing C1).

5.2.1.1.2 Instantiation (and Initialization)

The initialization of the ASIO audio driver is successful if the return value of the ASIOInit function is *ASE_OK*. This therefore denotes that the host application has moved into the *Initialized* state. The host application then indicates to the user all the information that pertains to the ASIO driver, such as the name of the driver and the version of ASIO it employs. After this, a handle to the ASIO driver from the OS's registry is presented as a structure that can be referenced by the developer for specific information. Crucially, this structure specifies the addresses in memory that the OS has assigned the audio driver for its input and output channels (which are the locations in memory to which channel specific audio samples should be streamed).

Having created the ASIO driver object, the ASIO callback structure is subsequently initialized, so that the application can appropriately respond to the queries that the ASIO driver will provide. This involves directing the "asioCallbacks" constructs' parameters to functions provided within the application. The constructs' *bufferSwitch, sampleRateDidChange, asioMessage* and, *bufferSwitchTimeInfo* callback parameters should each be provided with an appropriate response should the audio driver require them. In the context of this implementation, the *bufferSwitchTimeInfo* is the most important callback, as it is used by the ASIO driver to demand audio samples for streaming (as detailed below).

The flow of control of the application then moves to a *create_asio_buffers(&asioDriverInfo)* command that, if successful, shifts the application from the *Initialized* state to the *Prepared* state. This command takes the handle of the driver as a parameter and sets up all the input and output buffers for the application according to the information provided by the handle (i.e., the number of input/output buffers it supports, and its preferred buffer size).

5.2.1.1.3 Driver Queries, Host Queries, and Driver Notifications

The *driver query* and *driver notifications* aspect of the implementation refers to the ability of the host application and audio driver to communicate information between one another (see Figure 75 below). The *query* suite of communication tools refers to queries sent from the host application and responded to by the audio driver. These include inquiries about how many channels (both input and output) the driver supports, the buffer sizes that it supports, and the sample rate of the driver. It is worthwhile to note that buffer size can be defined as the amount of time allowed for the host application to process the audio before supplying it to an audio interface.[56] Therefore, a reduction in buffer size reduces latency, but increases processing requirements, as less samples are throughput to the audio driver in each buffer. Thus, the implementation should find some middle ground that best suits its needs (low latency vs processing cost).

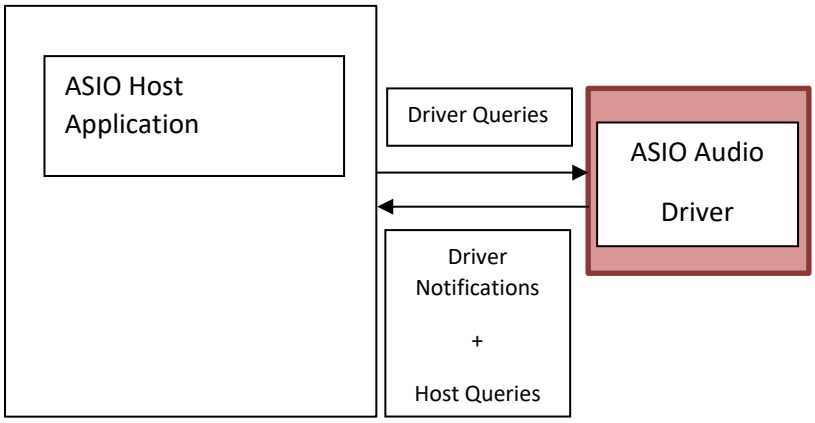

Figure 75: A diagram indicating the communication techniques employed between the ASIO Host application and the ASIO Audio Driver.

The audio *Driver Notifications* and *Host Queries* are messages that inform the host application about some or other issue that the ASIO driver may have encountered. For example, the driver might query the host application about whether the application supports *asioMessage* functionality, or which version of ASIO is being implemented. This information is determined for the driver through *Host Queries.* The audio driver might need to reset or change the buffer size and this is communicated through *Driver Notifications*. The queries are performed once the application is in the *Prepared* state, while the notifications are provided to the host application when it is in the *Running* state and streaming audio.

5.2.1.1.4 Audio Streaming

Audio streaming begins after the "ASIOStart()" function is called and the application moves into its *Running* state from the *Prepared* state. At this point, the flow of control enters a loop that executes while the ASIO application remains in its *Running* state, and breaks should the state change. This loop is where the developer should provide the input and output buffers of the ASIO driver with audio sample data to stream/playback. The ASIO driver maintains the application's execution by calling the *bufferSwitchTimeInfo* callback when it requires more audio samples for a particular output channel. Once the callback has been received, the developer is required to have created a system to supply the sample buffers for each of the active output channels. Of course, the loop requires a stop condition (i.e., after audio has finished streaming and the application should be closed). When this condition is met, a

[56] https://support.focusrite.com/hc/en-gb/articles/115004120965-Sample-Rate-Bit-Depth-Buffer-Size-Explained

set of state-transition functions should be created that call the “ASIOStop()” function, which shifts the application from the *Running* state back into the *Prepared* state. If the application needs to return to the *Initialized* state from the *Prepared* state, the functions should dispose of the allocated buffers (using the “ASIODisposeBuffers()” function. If the application is required to move from the *Initialized* state to the *Loaded* state, these state-transition functions should call the “ASIOExit()” function to shift the application back to its starting state (*Loaded*).

It is important to recognize that the audio driver requires its first set of samples before the application moves into the *Running* state. This is so that the audio driver can begin streaming as soon as the start function has been called. Generally, this first set of samples should be populated with silence, as the host application has yet to provide any samples from the input streams (audio sources).

Hopefully, this section has provided a concise understanding of how the ASIO SDK is used by developers to implement ASIO Host Applications that can stream audio out to compatible ASIO audio drivers. This framework obviously informs the playback server that was developed and which is detailed in the ASIO Playback Server section.

5.2.2 FMOD DSP Plugin Creation

Having introduced the ASIO SDK, the discussion will now look at the framework that enabled the creation of a custom FMOD compliant plugin. The FMOD Audio Middleware provides an API to developers that enables the integration of the FMOD audio processing functionality into whatever project and software environment they choose (and that FMOD supports). While aspects of this API are detailed in the previous chapter in the discussion of the FMOD Audio Middleware software solution, it is necessary to recapitulate the information on how this API is structured. Essentially, the API is comprised of two separate systems that function in unison. The Low-Level API is used to trigger FMOD audio tracks and provides a host of additional functionality, whereas the Studio API is a set of functions that are used to manage FMOD Studio banks and trigger events that have been authored within the FMOD Studio Application. This division of the APIs allows developers to separate their development according to what layer of the FMOD model their development lies (both APIs are written in C++, although the APIs can both be referred to using C#, or JavaScript (JS) from within the Unity framework). The Studio API can be seen to exist on top of the Low-Level API, as mentioned in the previous chapter.

In the context of the implemented capability, the FMOD Low Level API was of particular importance. This API enables developers to construct FMOD compliant plugins that can be augmented into the FMOD middleware software to provide specific and unique functionality that a developer may require. The plugins are compiled in C++ as Dynamically Linked Libraries (DLLs) that can then be imported into the FMOD Studio Application, as well as whatever game engine a developer is using for their projects. As such, two main types of FMOD plugins exist: namely, Effect Modules and Sound Modules (FMOD, 2021).

Effect Modules (EMs) are used to apply effects to an audio signal and have an input and an output. These effect plugins can be used for a variety of reasons. e.g., to perform down-mixing, compression, and distortion of audio signals. In the remainder of this section, the term FMOD DSP plugin will be synonymous with this type of plugin/module. Sound Modules (SM) are used to produce sounds and, as such, do not have an audio input and are thus not relevant to this research.

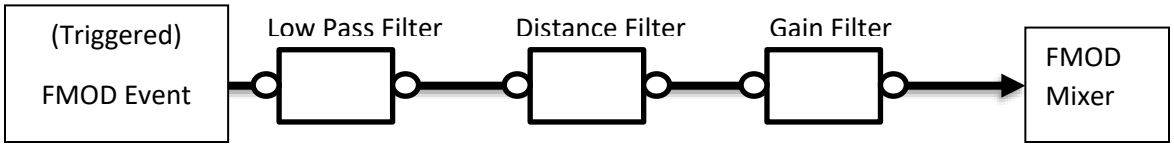

Figure 76: A diagram that indicates a signal processing chain of FMOD DSP plugins/Effect Modules

Figure 76 provides an example of a signal processing chain (of FMOD DSP plugins) that could be applied to an FMOD audio event. In this instance, a Low Pass Filter, Distance Filter, and Gain Filter are all applied to the signal before it is rendered by the FMOD Mixer. It should be recognized that there would be an effect chain for each FMOD audio event in an actual project that uses FMOD for signal processing. Due to the nature of an EM, this proved to be a perfect means to enter the chain at a specific point and inspect the audio samples as they passed through to the FMOD Mixer (i.e., inserting an Effect into the chain). For this reason, the following information applies to a plugin of this nature.

When constructing an FMOD plugin, the developer is required to adhere to a data descriptor in the form of a struct, entitled FMOD_DSP_DESCRIPTION. This descriptor "describes the capabilities of the plug-in and contains function pointers for all callbacks needed to communicate with FMOD", [57] and thus provides developers with the ability to construct functions within their plugins that house the signal processing logic that they require. The code block below indicates the various member variables and callback requirements that the FMOD_DSP_DESCRIPTION struct requires.

```
typedef struct {
  unsigned int pluginsdkversion;
  char name[32];
  unsigned int version;
  int numinputbuffers;
  int numoutputbuffers;
  FMOD_DSP_CREATE_CALLBACK create;
  FMOD_DSP_RELEASE_CALLBACK release;
  FMOD_DSP_RESET_CALLBACK reset;
  FMOD_DSP_READ_CALLBACK read;
  FMOD_DSP_PROCESS_CALLBACK process;
  FMOD_DSP_SETPOSITION_CALLBACK setposition;
  int numparameters;
  FMOD_DSP_PARAMETER_DESC **paramdesc;
  FMOD_DSP_SETPARAM_FLOAT_CALLBACK setparameterfloat;
  FMOD_DSP_SETPARAM_INT_CALLBACK setparameterint;
  FMOD_DSP_SETPARAM_BOOL_CALLBACK setparameterbool;
  FMOD_DSP_SETPARAM_DATA_CALLBACK setparameterdata;
  FMOD_DSP_GETPARAM_FLOAT_CALLBACK getparameterfloat;
  FMOD_DSP_GETPARAM_INT_CALLBACK getparameterint;
  FMOD_DSP_GETPARAM_BOOL_CALLBACK getparameterbool;
  FMOD_DSP_GETPARAM_DATA_CALLBACK getparameterdata;
  FMOD_DSP_SHOULDIPROCESS_CALLBACK shouldiprocess;
  void *userdata;
  FMOD_DSP_SYSTEM_REGISTER_CALLBACK sys_register;
  FMOD_DSP_SYSTEM_DEREGISTER_CALLBACK sys_deregister;
  FMOD_DSP_SYSTEM_MIX_CALLBACK sys_mix;
} FMOD_DSP_DESCRIPTION;
```

Code Block 2: The FMOD_DSP_DESCRIPTION struct.

[57] https://documentation.help/FMOD-API/FMOD_DSP_DESCRIPTION.html

To begin the construction of an FMOD DSP plugin, a developer needs to create a FMOD_DSP_DESCRIPTION struct object. This entails providing values for each of the members contained within the struct, while also providing references to callback functions for all requisite callback members within the struct. If the developer does not require all the member callbacks of the struct, they should set these instances of the unnecessary callbacks to null values.

The callback functions that were required for the implementation of the customized FMOD DSP will now be described. If the reader wishes more information on the callbacks that are not described in this section, they should refer to the FMOD API documentation (FMOD, 2021). [58] The developer is required to provide their plugin with a name, a version for the plugin, as well as the number of input and output buffers that it will process (this pertains to the type of audio; monophonic, stereophonic, quadraphonic). In addition, callback references need to be provided.

The *create* callback is used for the allocation of memory for the struct object that the developer defines. This callback is only called once by the FMOD Low Level system when all DSP plugins are attached to FMOD audio events. This then provides an ideal location for the allocation of variables and parameters that one might need in the functionality that the custom plugin provides.

The *release* callback is called when the FMOD Low Level system is deallocating resources after an audio event (to which the plugin is attached), has finished playback, or when the game application is closed. It is here that the memory that was allocated in the *create* callback should be deallocated. This then also entails the deallocation of all the variables that have been initialized. The *reset* callback is used to restore any data structures or buffers used within the plugin to their default state.

The *read* and *process* callback perform the same task, except that the *process* callback should be used by developers if there are any channel format changes in processing input channel data to output channel data (either upmixing or downmixing). These are the most important of the callbacks in the FMOD_DSP_DESCRIPTION struct, as they are called every time the FMOD mixer updates and are used to manipulate the digital signal of the audio event for every batch of samples (the sample block size is defined by the FMOD Low Level system). A representation of the *read* callback function and its parameters is provided below.

```
FMOD_DSP_READ(FMOD_DSP_STATE *dsp_state, float *inbuffer, float *outbuffer, unsigned int
length, int inchannels, int * /*outchannels*/)
```

- `*dsp_state` - The struct defined object.
- `*inbuffer` - A pointer to the location of the inbuffer (samples to be processed).
- `*outbuffer` - A pointer to the outbuffer (the location in memory where the processed inbuffer samples should be placed).
- `length` - The length of the incoming and outgoing buffer in samples.
- `inchannels` - The number of channels encoded in the PCM data (i.e., one for mono signals, two for stereo signals) located at the inbuffer pointer.
- `*outchannels` - The number of channels encoded in the outgoing PCM data, located at the outbuffer pointer.

[58] https://fmod.com/resources/documentation-api?version=2.01&page=plugin-api-dsp.html#fmod_dsp_description

The developer should perform their signal processing logic from within this callback—if one were constructing a low pass filter, the low pass filter logic would be applied to samples pointed to at the inbuffer pointer, and then, after processing, would be written to the outbuffer pointer.

As mentioned, the *process* callback encompasses the same functionality as the *read* callback, except that it receives references to the FMOD_DSP_BUFFER_ARRAY objects directly, and not just pointers to memory for the in and out buffers. Each of these objects, one for each plugin, contains the array of audio samples. The use of the FMOD_DSP_BUFFER_ARRAY objects requires developers to reference struct members (one of which is a pointer to the array of audio samples). An additional member of the struct is a reference to the channel format of the audio data. In comparison, the *read* callback only provides direct references to audio sample arrays and does not include information on the channel format of the audio data. A developer should decide, according to their needs, which of these to employ for their plugin. They can apply a macro definition to determine which of the callbacks should be used, such as:

```
#define FMOD_IMDSP_USEPROCESSCALLBACK
```

As one might guess, the DSP signal chain of an FMOD audio event functions in such a way that the first DSP effect plugin's inbuffer points to the location in memory of the actual FMOD event (the samples of the audio track housed within the event), which is provided by the FMOD mixer. However, every DSP effect plugin that follows the first will have an inbuffer pointer that points to the outbuffer of the preceding plugin in the chain. This trend continues all the way down the chain until the final DSP effect passes the audio samples (with all the effects applied to them) back to the mixer for output. The figure below better illustrates this process for an FMOD audio event having two DSP effects applied to it.

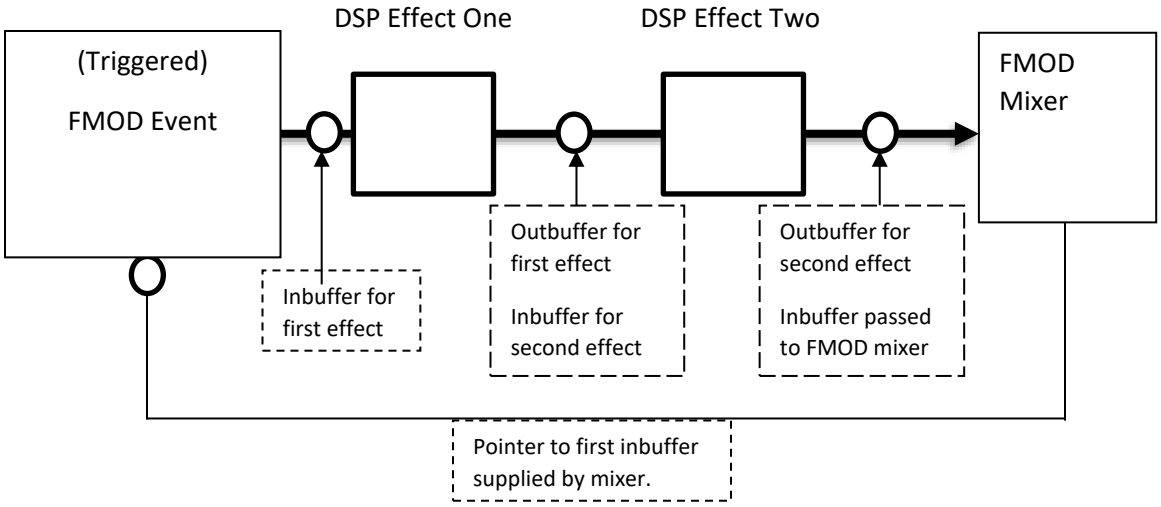

Figure 77: A DSP effect chain that indicates the buffer reference scheme. The FMOD Mixer is shown to provide the initial pointer in memory for an audio event that the 'DSP Effect One' initially references.

The number of parameters that a DSP plugin requires to perform its function needs to be supplied—for instance, in a distance filter plugin, the distance from the source, as well as the frequency of the source could both be parameters of the plugin. The FMOD_DSP_PARAMETER_DESC pointer points to an additional struct employed within the plugin that has a member for each of the parameters that the plugin requires (i.e., two in the case of the previous example, distance, and frequency). Intuitively, the

FMOD_DSP_SETPARAM_FLOAT/INT/BOOL/DATA callbacks are used to set the various parameter types that the plugin may contain.

This concludes the discussion on authoring custom FMOD DSP plugins, and should be seen as a description of the framework that was used in the construction of a bespoke FMOD DSP plugin, that is, the second component of the immersive audio capability (the details of which are described in section 5.3.4).

5.2.3 Game Object Coordinates in Unity

As discussed in the previous chapter, game engines are essentially a collection of libraries that are used together to create applications that encompass the rendering of interactive 2D or 3D environments. Of course, other functionality can be provided to a developed application by employing libraries that facilitate this functionality. These additional libraries can enhance and augment aspects of a game engine application, such as its physics system or networking capabilities of the application. Indeed, Audio Middleware could be seen as an additional library that aims to facilitate better audio control than that provided by the game engine's default/native audio engine. The Unity framework challenges conventional notions of proprietary game engines. In many ways, it is more than just a game engine and could just as easily be defined as a community driven collection of libraries that has been used to create thousands of applications (some of which are more game-like than others). While a description of Unity can be reviewed in the previous chapter, it is worth reiterating this aspect of the framework.

In the context of the capability that has been developed, Unity could indeed be viewed as a central point that unifies all the other libraries (SDKs and APIs) already mentioned in this section. An understanding of how Unity's coordinate system functions is needed to better understand how the localization and spatialization of audio through the third component of the immersive audio capability was achieved. This section will briefly identify some of the fundamentals of Unity's cartesian implementation, and how Game Objects (GOs) exist within this context. As with the previous chapter, Game Objects will also be referred to as GameObjects when discussing them in the context of Unity.

As defined by Unity, a GameObject "is the base class for all entities within Unity scenes" (Unity Technologies, 2021). [59] To briefly reiterate, the properties of a GameObject are called *Components*, which are the base class for everything attached to GameObjects. Therefore, *Components* can be anything from scripts (containing game logic) to custom shaders. In the context of positional data of a GameObject, the *transform* of the GameObject is a component that defines the position, rotation, and scale of an object. The *transform* of an object can therefore be manipulated to adjust an object's position, rotation, and scale within a Unity scene. Additionally, *transforms* are hierarchical. In other words, a *transform* can have a parent object (and therefore parent *transform*). This allows developers to apply changes to the *transform* of multiple objects if they are children of the *transform* being changed. In Unity, objects' *transforms* have both a *localPosition* and a (global) Position property. The *localPosition* refers to the position of an object relative to a parent *transform*. However, the position property refers to the location of the GameObject in world space. In the context of game engines, world space can be defined as the coordinate system for the whole scene. The world space has an origin at the center of the scene.

[59] https://docs.unity3d.com/ScriptReference/GameObject.html

The Unity coordinate system is a Left-Handed Coordinate system. This means that the system is a three-dimensional coordinate system in which the axes do not satisfy the right-hand rule. Three-dimensional coordinate systems have two possible orientations (see Figure 78 below).

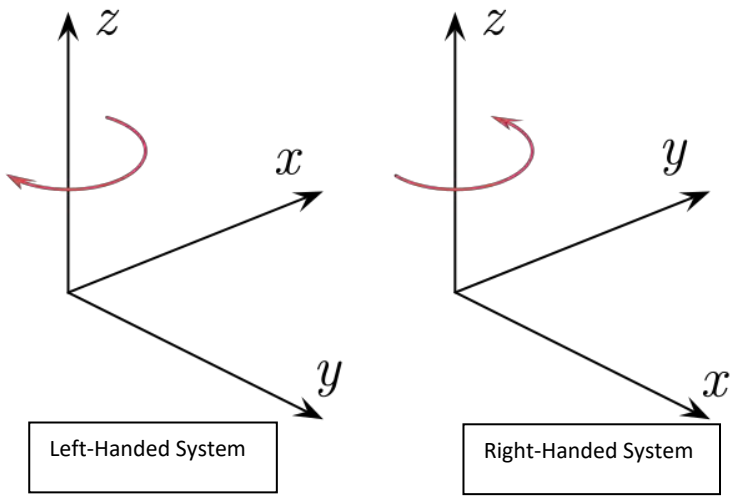

Figure 78: This diagram shows a comparison of the handedness of cartesian coordinate systems.

A further property of the Unity engine's coordinate system is that the x and z axes are horizontal axes, while the y axis is the vertical axis of the system (see Figure 79). You can therefore describe the Unity coordinate system as a left-handed, Y-Up coordinate system.

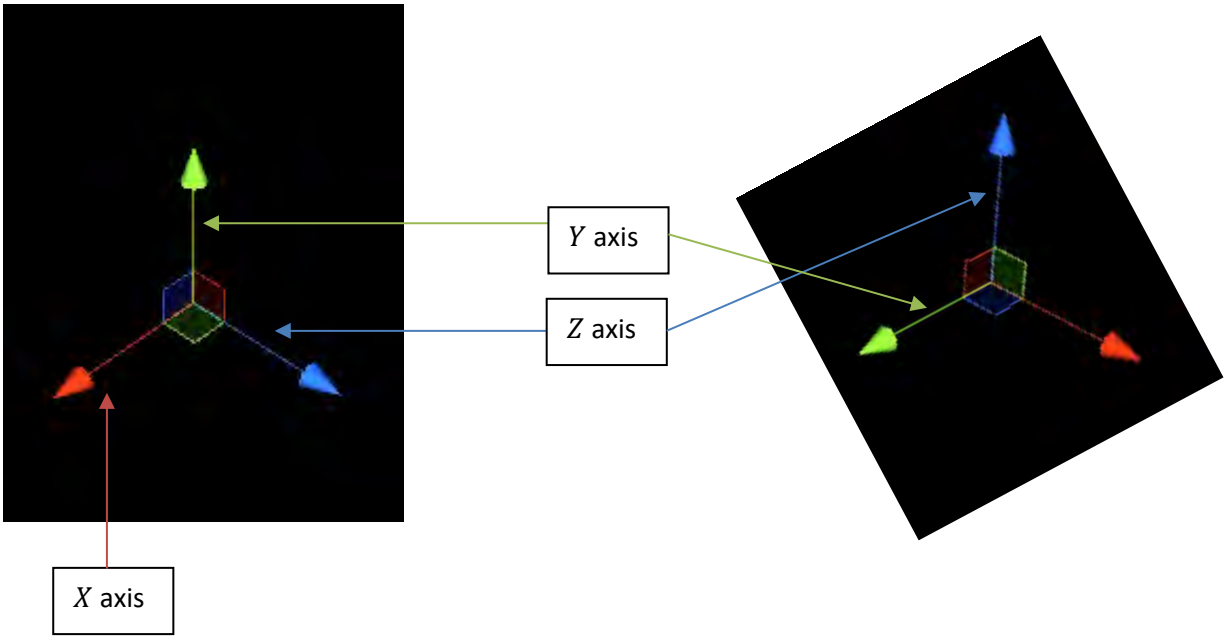

Figure 79: A screenshot indicating the axes used in Unity. The right images show that if you rotate the image so that the z axis is the vertical axis, the system is indeed Left-handed.

Since the spatialization system that was employed in this project uses a Left-handed Z-Up coordinate system, the data needed to be curated, to remedy this difference. The exact means of this curation are detailed further on in this chapter. As mentioned in the opening paragraphs of the chapter, one of the core mechanisms and components of the developed capability is the ability to transmit coordinates of virtual audio sources (i.e., GameObjects with FMOD audio events attached) to the ImmerGo spatialization system. To this end, a script was created that populates an array with GameObjects that have FMOD audio events attached. When the position of the game objects in the array is changed, this change is detected by the script, and the new position of the object is communicated to the ImmerGo system.

5.2.4 ImmerGo Localization of Audio Channels using 3D coordinates

The ImmerGo spatialization system was detailed and described in the third chapter of this thesis. However, before discussing how the system receives coordinates and then applies audio rendering algorithms to audio channels using these coordinates, it is worth briefly reiterating some of the fundamentals of the system and how it functions, but from a perspective that identifies the areas of the system that needed modification for its implementation in facilitating the immersive audio capability that has been developed. This revised system will therefore be called the modified ImmerGo system in the remainder of this section, to avoid confusion.

5.2.4.1 Description of the ImmerGo Spatialization Framework

ImmerGo is a speaker-based audio spatialization system that can position audio sources in three dimensions. As has already been noted, the system is implemented using a client-server-based model. Importantly, the system also employs Object Based Audio (OBA), which attributes metadata to an audio source. This metadata can be used to convey positional information amongst other parameters. The system has been developed to function in parallel with audio software that can output multichannel audio, for example a DAW, and enables the spatialization of the audio tracks housed within the DAW (review Figure 80 for an illustration of how the native system functions).

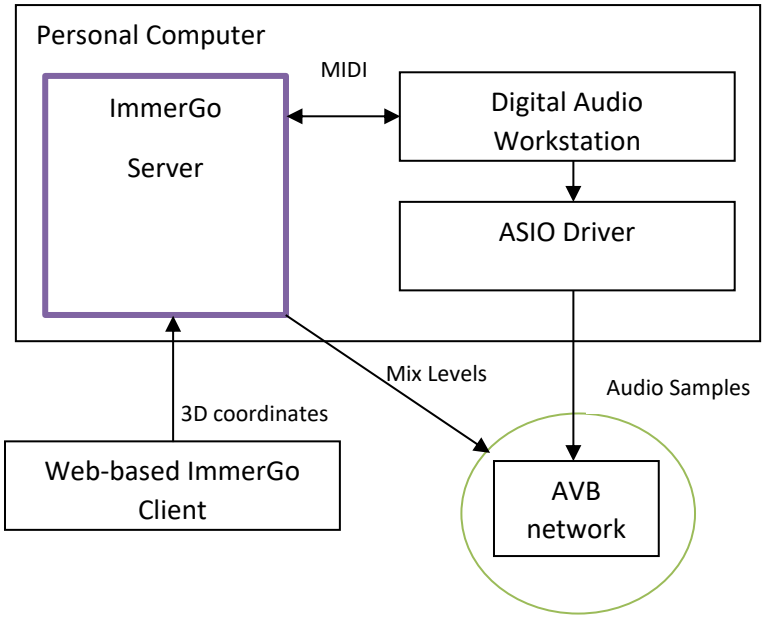

Figure 80: A diagram that schematically represents the native ImmerGo spatialization system and how it achieves its functionality through the communication of its components.

5.2.4.2 ImmerGo Server

The ImmerGo server connects to and communicates with speaker endpoints that are in turn connected to an Ethernet Audio Video Bridging (AVB) network. The server receives positional data from a client, and then employs spatial audio rendering algorithms to determine the mix-levels of the AVB endpoints (speakers). As alluded to in the chapter three, the exact location and distance of each speaker is required for the server to accurately perform its rendering calculations: that is, to ensure that the virtual audio object is correctly positioned relative to the speakers within the speaker array. The positional data that the server expects is provided from a web-based client that communicates to the server using User Datagram Protocol (UDP) messaging.

In addition to rendering mix levels for the AVB endpoints, the server also controls audio playback. The client sends a transport request to the ImmerGo server, which then translates the request to a Musical Instrument Digital Interface (MIDI) command that is received by the relevant DAW housing the audio tracks. To maintain synchronization with audio playback, the ImmerGo server also receives MIDI Timecode (MTC) information from the DAW, which it uses to determine when to localize channels on playback and timestamp localization on recording.

5.2.4.3 ImmerGo Client

The traditional client is a web-based one (generally a mobile device) with a GUI that enables a user to pan an audio source in three dimensions. The interface also enables users to control audio playback from the DAW and the real-time positioning of audio tracks in three dimensions. In addition, the client can record these positional changes, which can facilitate the creation of a three-dimensional mix for the selected audio track. As mentioned, while the user positions the audio, the client forwards these changes to the ImmerGo server in real time (see Figure 80).

5.2.4.4 Modifications to Framework

Having provided a concise description of how the ImmerGo spatialization (which consists of two primary components: the client and the server) operates, the discussion that follows will now indicate some of the areas of the system that needed altering. To utilize this immersive sound system from within the context of the Unity game engine and framework, certain modifications to both components needed to be made. As such, the modifications to the client will be initially presented, followed by an examination of how the server was altered. These changes can be broadly defined by a few key requirements detailed below.

i. The modified client would need to mimic the behavior of the native client.

ii. The ImmerGo server would need to accept three-dimensional information from a system utilizing its own, separate cartesian coordinate system.

iii. All functionality with the DAW through MIDI would need to be removed, as the audio tracks are housed within the Unity engine, which also dictates audio track playback control.

These three requirements are contextualized in Figure 81, which provides a comparison between the components and communication that comprise the modified ImmerGo system and the native ImmerGo system (see Figure 80). The diagram also contextualizes the three components that comprise the immersive audio capability and how they, in effect, interface with the ImmerGo system.

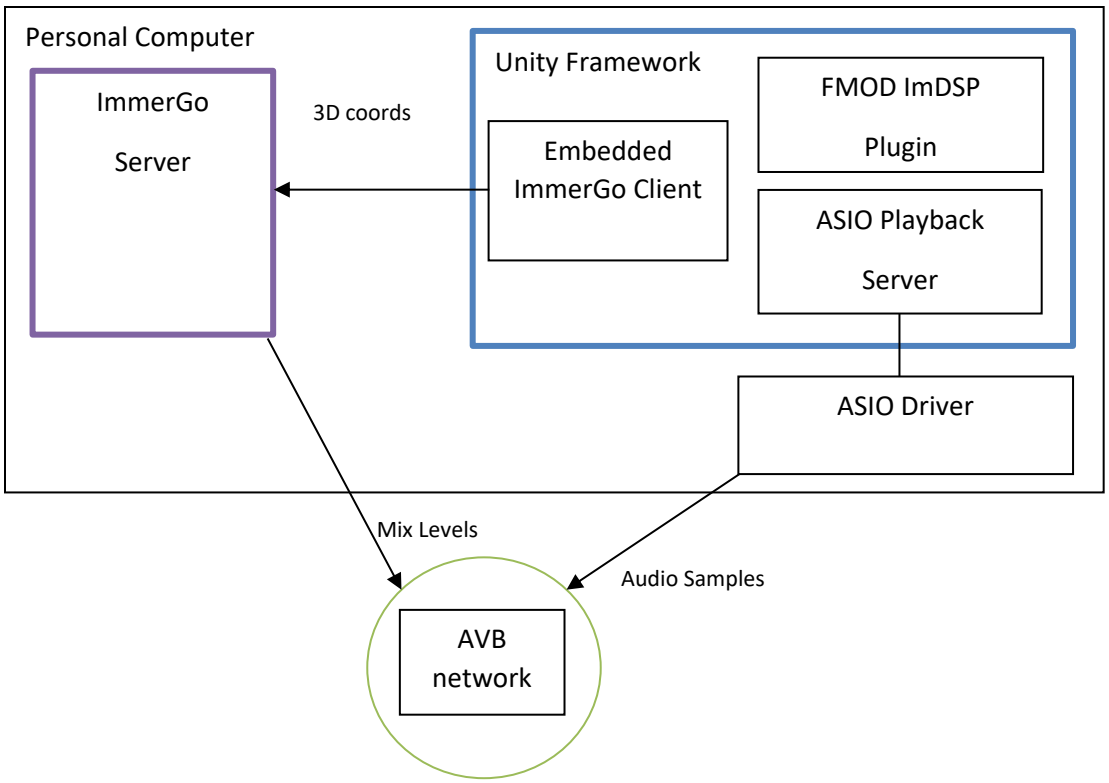

Figure 81:This figure should be viewed in conjunction with Figure 80, as it indicates the modifications to the native ImmerGo spatialization framework that enable 3D coordinate transmission.

5.2.4.4.1 The Modified ImmerGo Client

The client of the modified ImmerGo system needed to exist within the Unity framework in contrast to the native ImmerGo system using web-based clients. This "modified" client imitates the positional functionality of the web-based client by transmitting coordinates pertaining to audio objects using a socket asset that was openly available through the Unity store.

The native web based ImmerGo client also enabled the control of audio playback. Since the mechanisms of audio playback control are different to the native solution (see following section), the modified client only provides positional data of Unity game objects, to which FMOD audio events are attached, along with the audio game object identifier, to an instance of the ImmerGo server. The modified ImmerGo client is the crucial element of the Coordinate Transmission component of the developed capability, and its core functionality is detailed in section 5.3.5.

5.2.4.4.2 The Modified ImmerGo Server

The server was also altered to no longer require MTC information, and to only expect socket messages containing positional data for a relevant audio source in a Unity game engine scene. Section 5.2.3 alluded to the fact that the coordinate system of Unity is left-handed, and that whilst ImmerGo's is also left-handed, both operated with different vertical axes. To overcome this difference, the server was altered to use a Y-Up left-handed system.

In addition, the implementation ensured that the game objects and audio sources existed within the physical bounds of the speaker array. This involved requiring Unity game objects to occupy maximum ranges on each axis that adhere to the physical restrictions of the speaker array. A further mechanism was implemented to ensure that the unit measurements across both the Unity application and the ImmerGo system remain consistent. This mechanism is detailed in section 5.3.5.1.

5.2.4.4.3 Playback Control

Finally, as mentioned in the previous section, the playback functionality of the DAW and the MIDI transport control that the native ImmerGo system facilitated needed to be replaced. This was necessary for the capability to interface with the custom FMOD DSP plugin and the ASIO Playback server. Playback control over FMOD audio events is enabled using gameplay logic implemented via script. The details of this process are provided in chapter six, which indicates how audio playback control was achieved in the VR application. In addition, the workflow of the FMOD Audio Middleware solution provided in chapter four indicates how playback of audio events is controlled from within the Unity framework.

5.3 MECHANISMS OF THE IMPLEMENTATION

This section details the actual implementation of the capability that has been developed. The previous sections of the chapter have provided contextual information that expands on fundamental principles and software solutions that were used to develop the capability. Before discussing the implementation of each of the mechanisms (components) of the capability, it is worth reiterating the functionality of each. The first mechanism of the capability is the ASIO playback server. This component enables the receipt of audio samples being transmitted from the Unity game engine, and the subsequent streaming of this audio via output channels to an AVB ASIO driver. The second mechanism employs the FMOD audio middleware software solution to provide audio channels and data to the playback server, via the construction of a custom FMOD DSP Plugin. The third and final mechanism is a combination of custom scripts within the Unity framework that enable the transmission of positional data and audio event identifiers to the ImmerGo spatialization system. As detailed, this mechanism therefore behaves as a modified ImmerGo client. The following paragraphs and subsections will better define how each of these mechanisms functions in isolation.

5.3.1 The Limitations of FMOD in the Context of Immersive Sound

The FMOD software suite provides basic 3D panning capabilities that function optimally with predetermined speaker configurations (mono, stereo, quad, surround, 5.1, 7.1, 7.1.4). While the software does allow for irregular speaker configurations for developers, it requires them to manually configure a mix matrix, which involves mapping speakers to outputs. Users of a speaker-based immersive sound application developed using FMOD would have to find a way to configure their mix matrix according to their speaker configuration: the number of speakers and their precise positions. However, this is not possible without access to the source-code or precompiled version of the application—things that most users are unable to access.

The default configurations limit the audio output channel count to twelve (for 7.1.4 surround sound systems: 11 speakers and one subwoofer). As such, the middleware does not natively support immersive sound development for irregular speaker configurations with high (more than eight) output channels and large speaker numbers. When using predetermined speaker configurations for output, the default FMOD 3D panner and ‘FMOD Spatializer’ utilizes the FMOD mixer to determine the correct amplitude gains for each of the speakers in the specific configuration (see Figure 82 below). When using headphones for audio output, this Spatializer determines which generalized HRTF best describes the position of the audio source relative to the user’s head position. More specific detail on these mechanisms is presented in the previous chapter on Audio Middleware.

Figure 82: A screenshot of the FMOD Studio Application that indicates the native FMOD Spatializer on the left, and the preview of the 3D Audio event on the right.

5.3.2 Enhancements to FMOD to Overcome These Limitations

The ImmerGo spatialization system can support thirty-two output channels and an 'unlimited' number of speakers. To employ this functionality and overcome the output channel and speaker limits, while retaining the DSP power that FMOD possesses, an FMOD compliant DSP plugin was developed to stream out audio from an audio event to an external ASIO audio driver. Each audio event has an instance of the DSP plugin bound to it, and an identifier that enables the ASIO playback server to allocate the event to a specific output channel. By doing this, the eight-channel limitation mentioned in the preceding section is overcome. If there are enough output channels to provide one-to-one relationships with audio events, the system will be able to stream out audio from each audio event to each output channel. This solution bypasses the FMOD mixer, and therefore only provides playback functionality and does not enable spatialization from the onset. The spatialization of the audio sources is subsequently enabled by immersive audio capability operating within the ImmerGo system.

5.3.3 ASIO Playback Server

This section details how the ASIO SDK was used in the construction of an ASIO (Ethernet AVB Streaming) playback server with the ability to receive audio samples from socket-based clients, while acting as a multicast server. The section will also detail how these audio samples are subsequently assigned to an output channel for audio playback, and how this output channel is determined by the identifier of a particular client (ImDSP). The Windows Socket API (Winsock) was used for the socket communication between the playback server and the ImDSP instances (socket-based clients). As detailed in the official Microsoft Winsock documentation, "Winsock enables programmers to create advanced Internet, intranet, and other network-capable applications to transmit application data across the wire, independent of the network protocol being used. With Winsock, programmers are provided access to advanced Microsoft® Windows® networking capabilities such as multicast and Quality of Service (QoS)" (Microsoft, 2021). Appendix C indicates some of the functions that were used in the ASIO Playback server that have been omitted from the following paragraphs. In addition, the codebase for this project contains the main.cpp file for the ASIO playback server.

The Playback server operates using two threads that execute in parallel. After initialization, the first thread enters a receiver loop that constantly requests more samples from all the ImDSP plugins. Upon receipt of an audio event packet from an ImDSP plugin (bound to an FMOD audio event), the server determines which ImDSP plugin has sent the samples and then populates a corresponding circular buffer for its associated audio event (Listing C5 provides the class for this circular buffer implementation). For example, if ImDSP plugin number one sends a packet to the server, it analyzes the packet, recognizes it is from client number one, and then populates the circular buffer reserved for client number one (Figure 83 depicts this interaction). An instance of the ImDSP plugin associated with audio event one sends

across samples to the server, which then populates the respective circular buffer–the red circle in the diagram. Should ImDSP plugin client number two also send a packet, the server will perform the same process, although the circular buffer that will be populated will be the one reserved for output channel two.

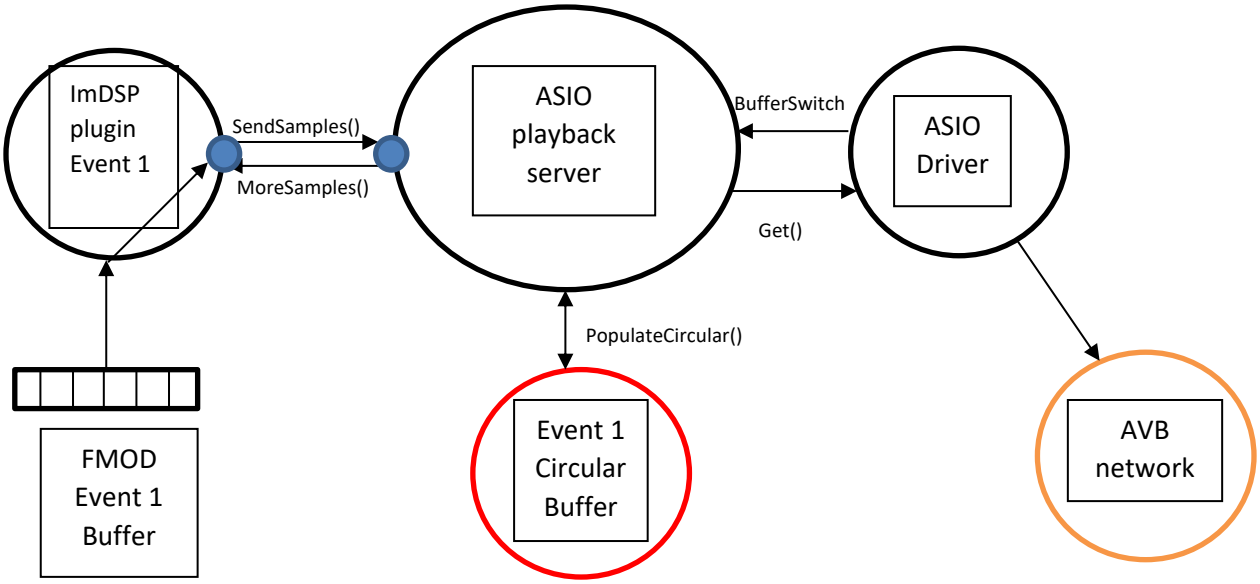

Figure 83: A diagrammatic representation of how an instance of the ImDSP plugin sends audio samples to the ASIO Playback server, which in turn supplies the samples to the AVB ASIO Audio Driver. This process is indicated for the first "event" and would therefore be replicated for each FMOD audio event in the Unity scene.

The second thread is used by the ASIO system to handle queries from the 'Streamware ASIO driver', which is designed to interface with a specific Ethernet AVB NIC that was used in this implementation called the 'Echo Streamware AVB compliant Network Interface Card (NIC)' or *Streamware NIC-1.* [60] When the driver requires more data for audio playback, it is forwarded samples for a client and thus ASIO output. As indicated in Figure 83 , the ASIO driver requests samples from the server. The server then retrieves the samples from the appropriate circular buffer and provides them to the driver, which then streams out audio across the AVB network (through a *get* function, which is indicated in the previous diagram).

[60] https://www.gearjunkies.com/2013/09/introducing-echo-audio-nic-1-pcie-network-adapter-and-streamware-sdk/

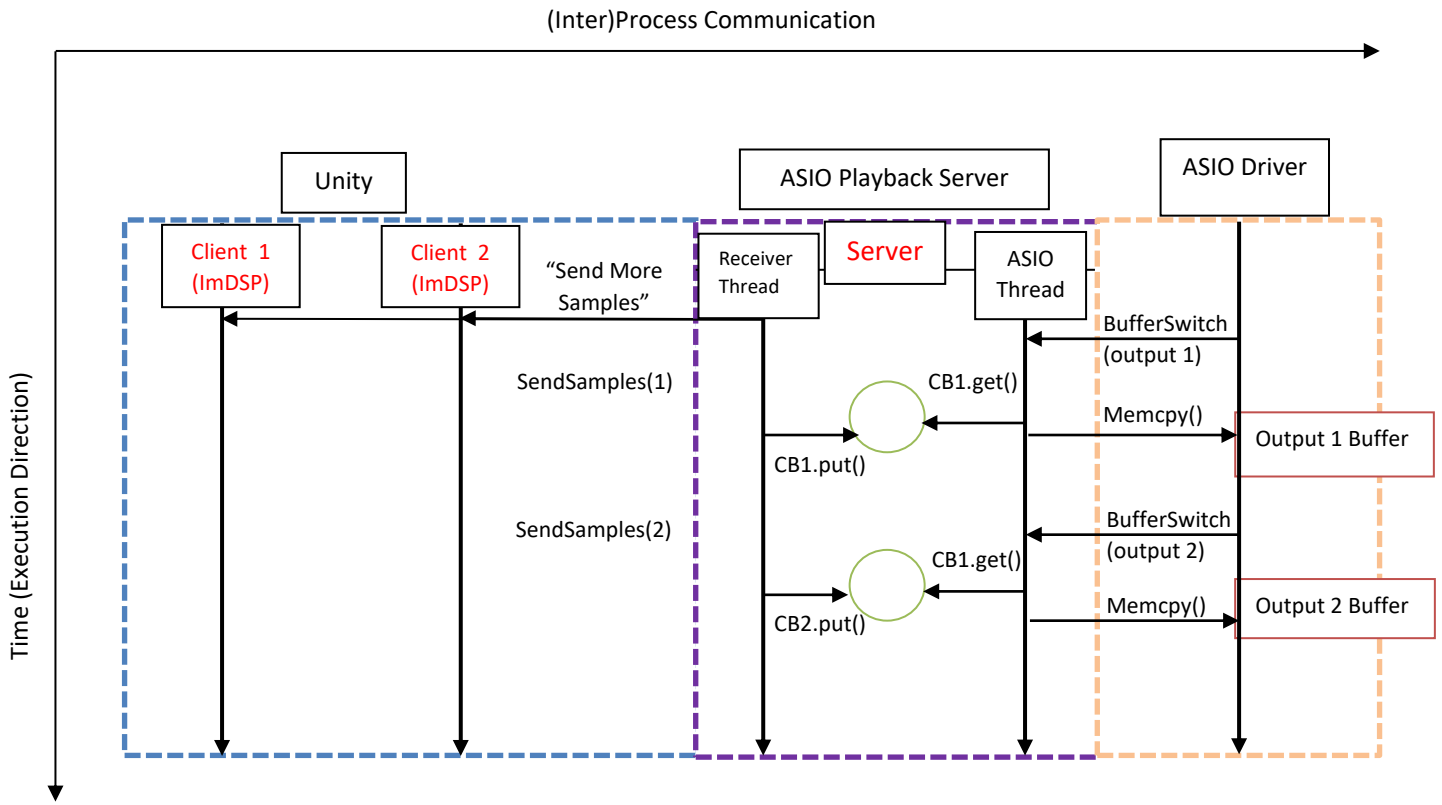

Figure 84: A UML sequence diagram indicating the threads of execution and the communication between the two FMOD ImDSP plugins and the ASIO playback server.

Figure 84 schematically represents the communication between two ImDSP plugins which contain multicast Winsock clients, and the ASIO playback server. The y-axis in this diagram indicates time elapsed, and therefore the direction of execution of the three processes (which are indicated in red). The process communication is represented along the x-axis. It should be noted that this communication operates bidirectionally. For example, the ImDSP clients transmit audio samples to the playback server, whilst the server communicates a request for more audio samples to both the clients.

As the driver requests more samples for a specific output channel, the receiver thread in the server application requests more samples from the ImDSP plugin clients embedded in Unity. The clients then send their respective samples across to the server, which places them in a corresponding circular buffer (represented as green circles in the figure). The ASIO thread is then able to pull samples from the circular buffer and populate the respective output channel's buffer.

Amongst the parameters supplied to the *SendSamples* functions in the figure are the specific client, and therefore FMOD audio event, identifier value. The three process domains (Unity, Playback Server, ASIO Driver) are color coded in the diagram according to the location of the communicating elements. As such, the horizontal arrows indicate communication between and within these processes. The reader should recognize that although the diagram only illustrates communication between two ImDSP plugins and the playback server, this overview would apply to all ImDSP clients, and therefore audio events, that are present and active in the Unity scene. It should also be noted that the diagram above represents a communication loop that is repeated until the FMOD audio events in the Unity scene are no longer active.

The previous paragraphs provide an overview of how the ASIO Playback component functions in tandem with the ImDSP component. The following paragraphs will now provide a more in-depth understanding of the implementation of the mechanisms that inform the playback server, and also a discussion of the circular buffer mechanism, the receiver thread, and the ASIO thread within the playback server.

Figure 86 shows a UML representation of a circular buffer used by the ASIO Playback server. The *head* variable is an integer that keeps track of the position in the buffer (*buf[ ]*) to which the audio samples should be written. Similarly, the *tail* variable indicates the location in the buffer where the next set of audio samples should be consumed by the ASIO driver. The *size_* variable represents the number of blocks per circular buffer that can be filled with batches/blocks of audio samples. In other words, this variable represents the size of the circular array. This size is determined by a macro definition (NUM_BLOCKS_PER_CIRCULAR).

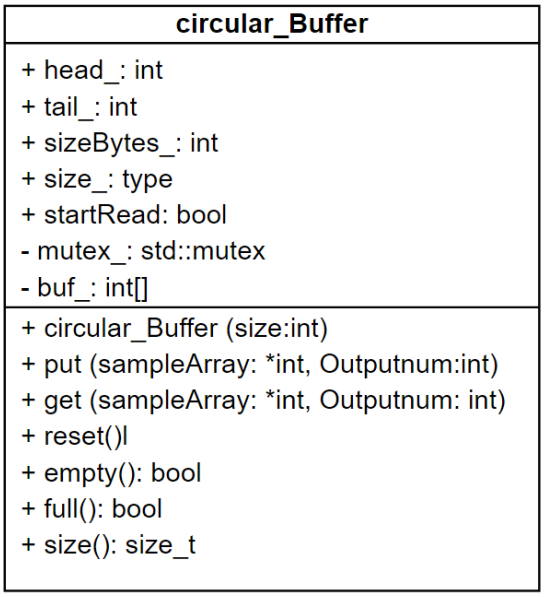

Figure 85: A UML diagram of the circular buffer class that was implemented to firstly store audio samples being received from the ImDSP plugins, and to secondly provide these audio samples to the ASIO streamware driver.

The important functions of this class are the *put* and *get* functions, which represent the provision and consumption operations respectively. The *put* function is used by the circular buffer objects to populate the head (front) of the buffer array. As such, the first parameter of the function is an integer pointer that provides a reference to the first sample in a set (block) of audio samples that have been provided by the relevant ImDSP plugin. The size of these sample sets was defined by the developer and adhered to a buffer size that is supported by the ASIO driver (see section 5.2.1.1.3).

In this ASIO playback server implementation, the buffer size is set to 512 samples, which is achieved by defining the DEFAULT_BUFLEN macro to 512 samples at the head of the server file. Indeed, this is a buffer size that the ASIO Streamware driver supports. The second parameter of the *put* function is the output channel number that applies to the audio source transmitting the samples, and therefore is the identifier of the client (ImDSP instance) that provides the server with audio samples.

Both the *put* and *get* functions utilize a locking function (a mutex object; the private variable in the UML diagram of the class), so that simultaneous *put* and *get* calls will not result in samples being overwritten. In other words, this ensures that while a *put* function is being executed, a *get* function needs to wait for the former to finish its execution and vice versa.

If the lock has been lifted, the *put* method can perform its core function, which is to copy the audio samples pointed to by the first parameter to the position in the circular array to which the *head_* index points. To reiterate, these audio samples are provided by the ImDSP clients. Once they have been copied, in accordance with the principles of circular buffer implementations, the *head_* index is incremented to point to the next element (block) in the circular array. The following code block indicates the contents of the *put* function.

```
void put(int *sampleArray, int OutputNum)
{
        std::lock_guard<std::mutex> lock(mutex_);
        memcpy(&buf_[head_*DEFAULT_BUFLEN], sampleArray, DEFAULT_BUFLEN * sizeof(int));
        head_ = (head_ + 1) % size_;
}
```

Code Block 3: the circular buffer class put() function.

The *get* function represents the consuming operation of the circular buffer. As such, the function's first parameter points to the audio sample array of the output channel (provided by the second parameter), whose location is referenced by the ASIO Driver.

Due to a problem whereby the audio samples of the FMOD audio events were being requested by the ASIO driver for playback before any samples had been transmitted to the ASIO playback server, a *startRead* condition was implemented to ensure that a certain number of sample blocks had been received by the server before any consumption of audio samples could begin (any *read* function operations). This also mitigates against situations whereby the rate at which audio samples are consumed is faster than their transmission. Of course, this countermeasure did produce certain playback overheads such as additional latency. The sample rate of both the FMOD mixer and the ASIO driver was set to 48000 (see Listing C3 as to how this is achieved from the playback server). If one excludes the network latency, the number of sample blocks used to "pad" the circular buffer dictates the expected latency from the audio samples leaving Unity and being played back by the ASIO driver—a padding of 4 blocks would result in a latency overhead described by the following calculations:

$$\begin{aligned} 4b \times 512xpb &= 2048x \\ 2048x \div 48xpms &= 42.67ms \end{aligned} \tag{12}$$

where b represents the sample blocks, xpb is the samples per block (buffer size), x is the number of samples, $xpms$ is the samples per millisecond, and ms is milliseconds. In other words, the initial playback latency experienced would be 42.67 milliseconds.

This therefore necessitated reflection on how to reduce this latency so that it is not perceptible, whilst also providing the necessary padding to reduce the chance of the circular buffers being emptied too quickly. One possible solution for this scenario was to sufficiently increase the number of blocks per circular buffer and defaulting the *tail_* index to point to an index further on in the array. This aspect of

the implementation is discussed in chapter eight. The following code block indicates the content of the *get* function, and the padding preventative measure.

```
void get(int* sampleArr, int OutputNum)
{
	std::lock_guard<std::mutex> lock(mutex_);
	if (empty())
	{
		//if the circular buffer is empty, provide the ASIO Output with silence.
		for (int i = 0; i < DEFAULT_BUFLEN; i++)
		{
			sampleArr[i] = 0;
		}
		return;
	}
	if (head_ > NUM_BLOCKS_PER_CIRCULAR-Padding)
	{
		startRead = true;
	}
	if (startRead == true)
	{
		memcpy(sampleArr, &buf_[tail_*DEFAULT_BUFLEN], DEFAULT_BUFLEN * sizeof(int));
		tail_ = (tail_ + 1) % size_;
		return;
	}else
	{
		return;
	}
}
```

Code Block 4: The circular buffer class get() function

The *reset* function locks the mutex object and sets the block number pointed to by both the *_head* and *_tail* to the same value, thereby resetting the buffer to its initial state, albeit that the buffer is not cleared of samples. The *empty* function indicates whether or not the circular buffer is empty. The *full* function determines whether the buffer is full. And finally, the *size* function is a helper function that determines how many blocks there are in the circular buffer.

```
//C1
circular_Buffer RecCli1(DEFAULT_BUFLEN * NUM_BLOCKS_PER_CIRCULAR);
```

Code Block 5: An example of how the circular buffer for the first ImDSP client is instantiated.

5.3.3.1 Receiver Thread

At the entry point of the playback server, the Winsock multicast server is allocated and initialized before the handshaking requirements to get a handle on the ASIO AVB driver are performed. This was done so that the server is ready to receive samples before the ASIO implementation requires audio samples for playback. A Winsock multicast server is set up using a few primary steps, as well as certain commands to enable multicast and therefore multiple client communication. Following this, the playback server is then able to transmit sample requests via the bound multicast send socket to all ImDSP plugin clients. It can also receive samples from the ImDSP plugins on a bound receive socket.

Having conducted the handshaking requirements of the Winsock multicast group set up, the server performs the requisite procedure to launch and provide a handle to the ASIO AVB Driver (as detailed in the ASIO AVB Interface subsection earlier on in the chapter). Following this, the playback server enters the *Running* state and executes a loop:

```
while (!asioDriverInfo.stopped)
```

Code Block 6: The condition that determines whether the ASIO application moves from the Running state and back to the Prepared state.

The body of this loop represents the "receiver loop" that was mentioned at the beginning of this section. It is in the body of this loop that the first "Send More Samples" request is transmitted to the clients of the multicast group. The following code segment indicates the command that is used for this request.

```
iResult = sendto(MultiSocket, requestBuf, requestbufflen, 0, (SOCKADDR*)&recvSock, sizeof(recvSock));
```

The second argument in the *sendto* function is an array of characters with the message: "Send More Samples". The third parameter is the length of this character array. *recvSock* is a socket object interface to use for transmission requests. Within the loop, the playback server waits to receive two packets and stores each of their contents in a structure shown in Figure 86.

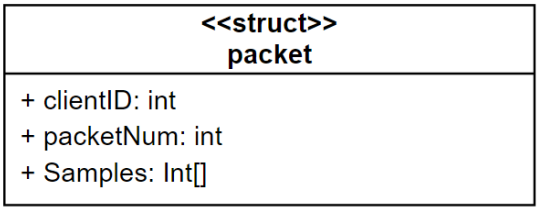

Figure 86: A UML representation of the struct that was created to aid the processing of audio samples and to identify the client to which the samples apply.

As can be seen, the packet struct has three members: an identifier member, the number of the packet, and an integer array used to store the actual audio samples being transmitted. The packet number is used to determine the sequence of the received packets, as each packet contains half the buffer size, which in this case is 256 samples, since the default buffer size is 512. This was a necessary implementation as they payload of a UDP packet is not big enough to transmit 512 samples at once.

Once the samples have been acquired, the loop then proceeds to determine whether the packets are from the same client and uses the client identifier to populate the corresponding circular buffer. The code block below indicates this process for client 1.

```
switch (clientID)
{
case 1:
    PopulateOnceCircular(clientID, recvPacketOne.Samples,recvPacketTwo.Samples, tempor);
    RecCli1.put(tempor, clientID);
    break;
```

Code Block 7: illustrates the population of circular buffer 1 to store the audio samples.

The *PopulateOnceCircular* function simply combines the two packets payloads and provides a temporary array which contains a single block (set) of 512 audio samples (see Listing C4). After this temporary array has been returned, the circular buffer class uses its *put* function to copy the audio samples to the current head of the circular array.

The reader should realize that the switch statement provided above will have cases for each of the ImDSP clients transmitting audio.

One additional condition of this receiver loop is to guarantee that audio sample packets are from the same client. If the packets are from different clients, the receiver loop is simply repeated, and the packets are dropped. If the packets are from the same client, and the appropriate circular buffer is provided with audio samples, the receiver loop iterates once again, and provides the multicast clients with additional "Send More Samples" requests.

5.3.3.2 ASIO Thread

The ASIO thread executes in conjunction with the receiver thread. Once the playback server is in its *Running* state, the ASIO AVB driver begins requesting audio samples for streaming. Every time the driver requires more samples, the *bufferSwitchTime* function is executed.

The *bufferSwitchTimeInfo* function operates by looping through all the available channels (both input and output channels) of the ASIO AVB Driver. As this ASIO implementation did not consider input channels, an identifier determines whether a particular channel is an output channel before proceeding. Having determined whether the channel is an output channel, the data type, and format (i.e., 32-bit integer type, 64-bit floating-point type) that the ASIO Driver supports is selected. In this instance, the *ASIOSTInt32LSB* type is selected. Once the correct data type is identified, audio samples are provided to the specific output channel. In this implementation, a switch statement dictates which circular buffer should be used in the consumption of a block (set) of audio samples. Once the correct circular buffer has been determined, the *get* function of this buffer populates a sample array that these samples to the output channel. The following code block shows functions that have been described. The cases for additional output channels are excluded for the sake of brevity.

```
for (int i = 0; i < asioDriverInfo.inputBuffers + asioDriverInfo.outputBuffers; i++)
        {
                if (asioDriverInfo.bufferInfos[i].isInput == false)  //is it an output buffer?
                {
                        // OK do processing for the outputs only
                        switch (asioDriverInfo.channelInfos[i].type)
                        {
                        case ASIOSTInt32LSB:        //buffer type for ASIO AVB outputs.
                                switch (i)
                                {
                                case 1:                                 //Output 1
                                  RecCli1.get(tempAr, 1);
                                   indicator = true;
                                   memcpy(asioDriverInfo.bufferInfos[i].buffers[index], &tempAr, Sizeof(tempAr)));
```

```
                    break;
```

Code Block 8: The ASIO Thread.

Once the first output channel has been provided with audio samples, the loop iterates for each output channel, and provides them with the requisite audio samples from their circular buffers. This process thereby results in this audio being streamed out over the Ethernet AVB Network to speaker endpoints. After all the output channels have been provided data, the ASIO thread will hang until the ASIO AVB driver once again requires samples. At which point, the *bufferSwitchTimeInfo* callback will begin executing again. Figure 86 indicates diagrammatically how the receiver thread functions. The vertical black arrow indicates the direction in which the receiver thread executes. The infinity sign at the top of this arrow indicates that the thread repeats this process infinitely or until the ASIO driver no longer requires audio samples for its outputs.

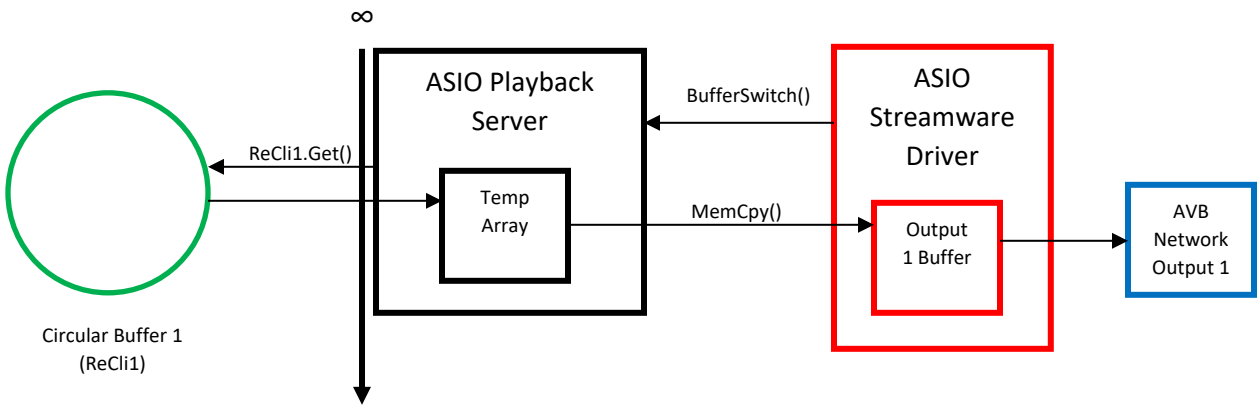

Figure 87: A simple schematic diagram of how the Receiver thread functions. In the diagram, the function calls closer to the top are performed before those at the bottom. In this instance, only output one requires samples, which are therefore being taken from the appropriate ReCli1 circular buffer.

5.3.4 FMOD ImDSP Plugin

This section outlines the procedure that was followed when constructing an FMOD DSP plugin which transmits audio samples to the ASIO playback server from FMOD audio events housed within a Unity project scene. As mentioned, FMOD DSP Plugin developers need to construct a C++ Dynamically Linked Library (DLL). This DLL exports the FMOD_DSP_DESCRIPTION (plugin description) struct that enables the FMOD Studio application, as well as Unity, to register the DSP plugin and use it within the Studio Application, or through a Unity project that is integrated with the FMOD Low Level system. The authored DSP is added to the signal chain of a FMOD audio event, in the same way any conventional DSP effect would be added to any other FMOD event.
As indicated in section 5.2.2, the plugin description also enables the FMOD Programmer API to make callbacks to the plugin to perform a specific function or to report what state the plugin is in, as well as to provide the FMOD system with a means to hook into and perform the DSP processing on an audio signal.

The DSP plugin that was created was called the “ImDSP” plugin (ImmerGo plugin). A developer need merely bind an instance of the “ImDSP” plugin to an FMOD audio event to provide the audio transmitting functionality that is detailed below. Figure 88 provides a diagram that indicates where the

ImDSP plugin should be placed on the DSP signal chain of an FMOD audio event. In this instance, the chain consists of three additional DSP plugins that should be placed before the ImDSP plugin, so that the signal can benefit from these effects being applied to it before being transmitted to the playback server. This figure can be contrasted with Figure 76, in which the FMOD audio event does not have the ImDSP plugin attached to its chain. The FMOD mixer has a dotted line in this figure to indicate that samples passed to it from the ImDSP signal are muted, i.e., zero value arrays.

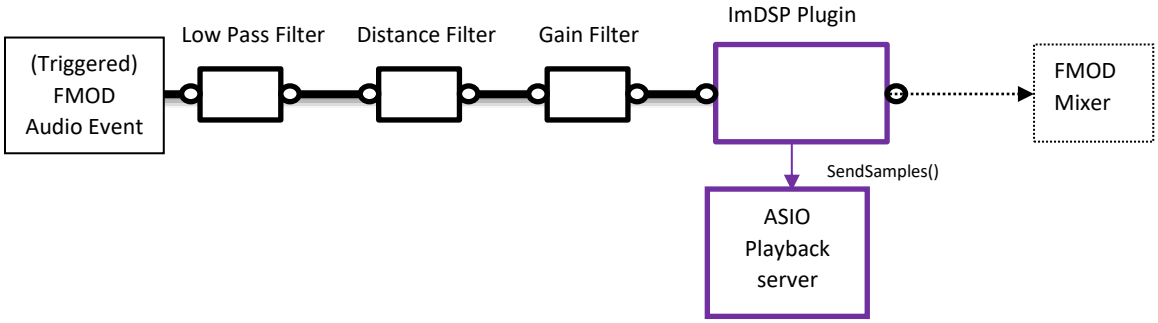

Figure 88: An FMOD audio event signal processing chain containing an instance of the ImDSP plugin.

The following paragraphs will describe the functionality and logic that was employed in the creation of this FMOD DSP plugin. As such, the reader is encouraged to cross-refer to section 5.2.2, as it outlines the fundamentals of the construction of an FMOD compliant DSP plugin. In addition, Figure 84, which was provided in the previous section, provides an overview of how the ImDSP plugins communicate with the ASIO Playback server, which in turn provides the ASIO driver with audio samples to stream over the Ethernet AVB network.

To transmit the audio samples of an FMOD event to the playback server, the ImDSP plugin required to have some sort of transmission functionality. After evaluating various Inter-Process Communication (IPC) mechanisms, it was decided that the plugin should employ socket communication to achieve data transmission from the Unity framework to the ASIO playback server. An important consideration that determined this socket-based approach was that it enables a distributed approach to providing spatial audio playback, which is congruent with distributed processing principles that the ImmerGo spatialization system enables (Devonport & Foss, 2019). The plugin supports Winsock functionality and is deployed to act as a multicast client to the ASIO playback server— the server requests samples, and multiple clients (each “ImDSP” instance on individual FMOD audio events) transmit their samples to the server for playback (Figure 88 contextualizes the ImDSP communication with the playback server). Essentially, the plugin retrieves FMOD event samples and transmits them across to the ASIO playback server upon request. The playback server consumes the samples and then sends more requests to the clients for additional audio samples. Each FMOD audio event has an instance of the ImDSP plugin bound to it, and a unique client identifier which the server then uses to assign an output channel.

While this discussion has provided the reader with a broad sense of how the bespoke ImDSP plugin functions, it is now necessary to examine the precise implementation of the plugin. The ASIO playback server expects to receive batches of 512 samples from its multicast group clients. As such, the FMOD System within Unity was set to operate with a sample buffer size of 512 to remain consistent with the playback server’s requirements. In addition, the multicast clients were initialized with the requisite multicast group address, and the receive and send ports.

In adhering to the FMOD_DSP_DESCRIPTION object specifications detailed earlier, a unique description object, called FMOD_ImDSP_Desc (of type FMOD_DSP_DESCRIPTION) requires initialization by the ImDSP plugin. The implementation of this object can be found in the Appendix BListing B1 of this thesis, which contains the variety of callbacks that the ImDSP DSP description struct requires. Code Block 2 indicates the callbacks that the description object provides. Once the description object has been initialized, the FMOD system executing in Unity needs to get a handle to this object if it is to use the DSP in an effect chain for an FMOD event. Accordingly, the object is exported, along with the parameter descriptor values, that are used in the FMOD Studio Application to adjust graphical representations of the plugin when it is used on signal chains in FMOD studio. After the object has been exported, an object of type *FMODImDSPState* within the plugin is instantiated. The *FMODImDSPState* class contains the requisite public functions to be executed when the corresponding callback function is called. This class, and member functions, can be seen in the code block below.

```
class FMODImDSPState
{
public:
        FMODImDSPState() {}

        void    init(FMOD_DSP_STATE *dsp_state);
        void    release(FMOD_DSP_STATE *dsp_state);
        FMOD_RESULT process(float *inbuffer, float *outbuffer, unsigned int length, int channels);
        void    reset();
        void    setClientID(int);
        int     clientID() const { return m_clientID; }


private:
        int     m_sample_rate;
        float    m_max_channels;
        int     m_clientID;
};
```

Code Block 9: The FMODImDSPState class

When a Unity application begins execution, and the FMOD System initializes an “ImDSP” plugin, the *init* or *create* callback for the plugin is called, and the client ID (FMOD event identifier) variable is initialized. This variable is used to identify which client the instance of the “ImDSP” plugin belongs to and is used in the process callback and data transmission. As has been consistently mentioned, the “ImDSP” plugin’s primary function is to function as a Winsock client that is part of a multicast group used in the transmission of audio samples to the ASIO playback server. In the create callback, a function performs all the necessary handshaking to create an instance of a socket that can be written to, and the enrollment of the client (this instance of the plugin), to a Multicast socket group (see Listing B2).

The *release* callback is set up to firstly release all memory allocations for the sockets. Additionally, the callback deallocates the memory used for the description object. The release callback is used once the FMOD audio event has finished playing, or when the Unity application is being closed and resource deallocation takes place.

In section 5.2.2 of this chapter, it was noted that the *read* and *process* callbacks are the most important callbacks with regard to manipulating an audio signal as it travels down a DSP effect chain. Both

callbacks perform the same essential function, except that the *process* callback provides greater specificity regarding input and output channel formats. For this reason, the *process* callback was selected. The code for callbacks that have been mentioned, including the *process* callback (detailed in Listing B4Appendix B), are all viewable in the codebase accompanying this thesis. The *process* callback is employed to determine the client identifier of the plugin, and subsequently populates a temporary audio sample buffer with the samples from the FMOD audio event. After this, the *SendSample* function is called (see Listing B5).

The temporary sample buffer is passed as a parameter to the *SendSamples* function, which performs the final step required in socket communication: namely, to send and receive data (see Listing B5). While the exact commands that achieve this are not provided in-text, they are viewable in the "ImDSP" code-listing. Essentially, the function waits to receive a request for samples from the ASIO playback server, and upon receipt of this request, takes in the samples from the *SampleBuffer* array and populates two packets with them (each with 256 samples of the 512). Thereafter, it transmits the packets to the ASIO playback server across the multicast socket that was initialized. These packets are objects generated from a struct that has the same components and types as the packet struct that the server uses to obtain the audio samples (see Figure 86). The mechanism by which the developer decides on the identifier for the instance of the ImDSP plugin is provided in the following chapter, although an overview of this mechanism is provided in the final section.

5.3.5 Coordinate Transmission

To enable spatialization of the audio events, the coordinates of the events need to be transmitted to the ImmerGo server. Each audio event is assigned an identifier (an integer value) that is also transmitted to the ImmerGo server along with the updated coordinates (see Figures Figure 89 and Figure 83). This identifier is the same one that is used when transmitting the audio samples to the external audio driver. Accordingly, each audio event is assigned an identifier that is used both to assign it a specific output channel (via the ASIO driver) and to notify ImmerGo about which event's 3D position has changed. The server receives the coordinates, and then applies an audio rendering algorithm with the them as input. The result of the rendering algorithm is then applied, via mix levels for speakers in the array, to the NDAC-8 devices on the AVB network.

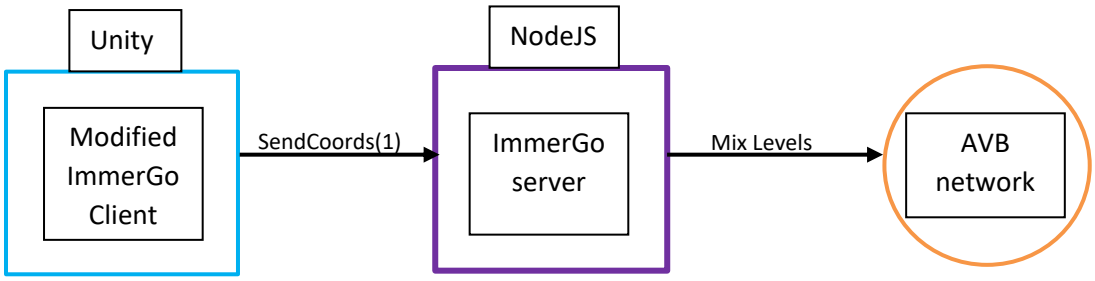

Figure 89: A simple diagram that was first used in the Introduction to this thesis. The diagram serves to remind the reader how the modified ImmerGo client communicates with the ImmerGo server, which, in turn, determines and transmits the mix levels for the NDAC-8 devices on the AVB network.

To facilitate this socket communication from within Unity to the ImmerGo server, a socket.io Unity asset was acquired, which enables developers to create, maintain, and close socket connections with applications that are external to Unity. Since the utilization of this asset, it has been deprecated by

Unity, and now exists as an open-source asset available through github.[61] A script, entitled *SocketIOClient*, was developed to act as the client to the ImmerGo server. It is bound to the camera object in the Unity scene and is launched at run-time with a list of the positions of each audio source in the scene. The previous positional values are regularly compared against the current positional values of the specific audio events. If there is a difference (i.e., a game object containing an audio event has moved), a socket message is sent to the ImmerGo server with the updated coordinates of the audio event. Figure 89 indicates the coordinates of a game object containing an audio event being transmitted to the server.

Now that an outline of this coordinate transmission mechanism has been provided, the logic employed within the *SocketIOClient* script will be supplied. Scripts that provide custom functionality to Unity projects are authored in either C# or JavaScript, and as such subscribe to a Unity constructed interface called *MonoBehaviour.* As defined by Unity, "*MonoBehaviour* is the base class from which every Unity script derives". This interface defines certain functions that are called at the various phases of a Unity application's execution. While the developer need not utilize any of these functions, they provide a variety of ways in which game logic can be implemented within a script. In the context of the script that was developed, only one of these functions was utilized: the Start() function. This function is called when a script is enabled and, is called before any of the Update() functions are called. It is only called once in the lifetime of the script, and is generally utilized to perform variable initialization, and allocation of resources that are used in the script's lifetime. To briefly provide some context, the suite of Update() functions also defined by the *MonoBehaviour* interface are called for every frame in the application's execution. Although this Update() function could have been used to transmit coordinates, the biggest issue with it, as detailed in the previous chapter, is that it does not execute at a fixed interval, which is a requirement of the modified ImmerGo server.

The population of a GameObject array with the GameObjects that are audio sources in the scene is an important precursor to the *SocketIOClient* script's function. This is achieved by the creation of a public GameObject array that can be populated via the use of the Unity Inspector (a window that displays variables contained within the script and, if any, components that are attached).

The core logic of the script resides within a coroutine. In the Unity framework, a coroutine is a modified function "that can pause execution and return control to Unity but then continue where it left off on the following frame."[62] The Start function of the script is used to launch the coroutine that was implemented, while also finding a handle to the SocketIO object that enables socket communication. This coroutine is called *SendCoords* and was implemented to execute at 10ms intervals (which is the same interval that the native ImmerGo server expects to receive positional data from its native web-based client). *SendCoords* inspects the *localPosition* of each of the FMOD audio objects every 10 milliseconds (ms) and if a change in position is detected, it is transmitted to the spatialization server within this same interval.

On launch, the coroutine initializes a set of Vector type position variables (which is to say, a tuple with three fields of type *int* and can therefore be used to store the coordinate values of GameObjects for each of the axes). These variables are essentially placeholders that are used to store the position values of the public GameObject array that is initialized by the script at its start.

[61] https://awesomeopensource.com/project/floatinghotpot/socket.io-unity
[62] https://docs.unity3d.com/Manual/Coroutines.html

The following statement initializes the variable posa1(the position of the first GameObject) with the three int fields of the Vector type set to values of 0.

```
Vector3 posa1 = new Vector3(0, 0, 0);
```

Once the position variables have been initialized, the coroutine enters a loop that checks to see whether any of the GameObjects with FMOD audio events attached have changed position.

Code Block 10 indicates the position check for the GameObject housing the first FMOD audio event in the scene.

```
        //Audio Source 1 (source1 = Earth Game Object)
        Vector3 temp1 = posa1;

        posa1 = source1.transform.localPosition;

        if (posa1 != temp1)
        {
           Vector3 percentPos = PercentileConversion(posa1);


           Dictionary<string, string> data1 = new Dictionary<string, string>();
           data1["x"] = System.Convert.ToString(percentPos.x);//ax
           data1["y"] = System.Convert.ToString(percentPos.y);//ay
           data1["z"] = System.Convert.ToString(percentPos.z);
           data1["spread"] = System.Convert.ToString(0);
           data1["trackNo"] = System.Convert.ToString(0);


           JSONObject bob = new JSONObject(data1);


           socket.Emit("coordsvr", new JSONObject(data1));
        }
```

Code Block 10: A segment of the SendCoords Coroutine that determines whether a relevant GameObject's position has changed according to a previous (temporary) value.

If the GameObject's position has changed, a data curation function is applied (detailed below) to the posa1 variable, and the resultant vector is packaged, along with the event identifier, into a JSON object, and transmitted to the ImmerGo server. The *socket* object used in the emission of the JSON object is the SocketIO handle that is initialized in the *Start* function. This JSON object represents the metadata of the audio game object. A more comprehensive view of the *SendCoords* coroutine is provided in Listing A1.

The enumeration of the FMOD audio objects according to their position within the audio object array provides them with an identifier ("trackNo") that the spatialization system uses to recognize which object applies to what positional data (see previous chapter). Indeed, this enumeration identifier is consistent with the identifier of the ImDSP plugins and the output channels and circular buffers in the ASIO playback server.

5.3.5.1 Curation Requirements

As was noted earlier in the chapter, differences between the coordinate systems used by Unity and the ImmerGo spatialization resulted in the need for certain data curation functions to be implemented. For this reason, the normalization function (shown in Code Block 11) takes in a position variable as a parameter and returns a vector that has been converted into a value between 0 and 1 according to the ranges of each of the axes (which will be application specific). The following paragraphs will elucidate this feature.

This functionality was required due to a realization that Unity applications might utilize very small- or large-scale representations of virtual environments. To accommodate this possibility, this conversion enables the spatialization system to contain audio sources within the physical bounds of the speaker array.

```
    public Vector3 PercentileConversion(Vector3 position)
{
    Vector3 TransmissionVector = new Vector3();
    //convert ranges into percentiles
    TransmissionVector.x = ((position.x - ApplicationMinimums.x)/ApplicationRange.x);
    TransmissionVector.y = ((position.y - ApplicationMinimums.y) / ApplicationRange.y);
    TransmissionVector.z = ((position.z - ApplicationMinimums.z) / ApplicationRange.z);

    return TransmissionVector;
}
```

Code Block 11: The PercentileConversion function.

Code Block 11 above indicates the structure of the function that converts the coordinate values of the vector parameter provided into percentages based on the hard limit ranges (for each axis) that the GameObjects can occupy. The function assumes that the ImmerGo system will receive the percentage positional variables, and then convert the percentages to the appropriate values in each of its ranges for the axes. The server side of this data curation is available in the ImmerGo source code available in the codebase.

5.4 Summary

Hopefully, this chapter has sufficiently explained how the immersive audio capability has been implemented for this project. The codebase pertaining to this capability, and each of the comprising components, is provided in Appendix B of this thesis should the reader wish to review aspects of the implementation. The utilization of this capability from within a Unity application, and therefore the detailing of its workflow, is the domain of the following chapter. The authoring of FMOD audio events, the attachment of *ImDSP* plugins to these events, and the binding of the events to game objects within a Unity scene forms an important aspect of this implementation that is examined in chapter six.

6 A VR SYSTEM THAT UTILIZES IMMERGO

"Not just beautiful, though – the stars are like the trees in the forest, alive and breathing. And they're watching me."

— Haruki Murakami

The previous chapter provided an overview of the core functionality and implementation that facilitated the use of speaker-based spatial audio in VR and digital game development for Unity developers (i.e., the immersive audio capability). The next step of the research was to exhibit this capability through the development of an immersive VR experience. The immersive experience was created firstly to showcase the immersive audio capability in a real use-case, and secondly to analyze how the capability compared to the Oculus Spatializer, which is a headphone-based HRTF spatialization system. In so doing, the core aims of the project were realized, that is, to determine the differences between immersive spatial audio implemented with a speaker- and headphone-based approach in the context of (immersive) VR applications. Therefore, the hypotheses about spatial audio output systems could be corroborated or negated when reviewing relevant literature. However, another goal of the project was to provide the immersive VR application with high fidelity audiovisual elements and an intuitive mode of interaction. This foundation for the immersive experience thereby enables answers to the question as to whether speaker- or headphone-based immersive audio systems best complement VR experiences. Through qualitative analysis, the usability of the system, the comparative audio quality of both headphone- and speaker-based systems, and the localization accuracy of each system were evaluated to provide these answers. The testing procedures and results of this analysis are presented in chapter seven.

This chapter will therefore consist of a few sections that will firstly analyze the processes adopted to create the VR experience from within the Unity game engine and FMOD Audio Middleware frameworks. This will describe the creation of the scene, the mode of interaction that was used, and how specific gameplay logic was implemented to facilitate the various aspects and requirements of the experience. In addition, the workflow of the FMOD studio project to provide FMOD audio events for both headphone and speaker environments will be examined. A description of how the scene was kept consistent across both speaker- and headphone-based implementations will also be provided. Once the construction of the scene has been described, a section will examine how a developer could use the system for their own projects, i.e., the workflow of the capability being utilized within the Unity and FMOD development frameworks will be provided. Finally, a brief section will recognize some of the resources that were used in the developmental process, along with certain enhancements of the implementation. However, before discussing the above aspects, the goals of the developed application will be presented, in order to provide the reader with a reference point so that they might better see how the various elements combine to achieve these goals. Indeed, a brief description of the VR application is pertinent before reading further.

It was decided that an immersive planetary scene would be an intuitive and visually appealing use-case for the capability. The orbit of planets in our solar system provided a suitable scenario for the panning of

audio in three dimensions. Moreover, it was decided that the user would be positioned at the center of this solar system, and that eight planets would orbit around them emitting audio. As a result, the application was entitled “Immersive Planets”. Eight planets were selected due a hardware constraint of the Digital to Analogue Converters (DACs), which only facilitated eight unique audio channels. In addition, it was the researcher’s view that eight sources with different positional values, due to unique orbiting parameters, would provide a holistic representation of near and far audio sources in a normal VR/Unity scene. Theoretically, however, the immersive audio capability could enable a maximum of 128 audio channels in a scene. This maximum is consistent with the FMOD audio middleware upper limit of active channels.

6.1 THE GOALS OF THE APPLICATION

The following section is divided into two subsections which describe the creative and developmental goals of the application, respectively. The creative goal provides a context for the design and implementation decisions that were taken during the immersive VR application’s implementation. The developmental goals indicate decisions taken to promote simplicity and ease-of-use when employing the immersive audio capability in Unity and FMOD. They both, however, are relevant to a facilitation of the primary and secondary goals of the research that were detailed in the introductory chapter of this thesis.

6.1.1 The Creative Goal

The creative goal of the application was that **the application should conform to the requirements of a “highly” immersive VR application/experience**. These requirements are indicated in chapter two, and include that the audiovisual fidelity of the experience should be of a high standard (see Table 1). Additionally, the multimodal application should provide intuitive interaction. The relative success of this goal and the degree to which the application can be classified as “highly” immersive are discussed in the following two chapters.

6.1.2 The Developmental Goals

From the perspective of using the developed capability (chapter five), the goal in the construction of this VR demo was to enable intuitive use of this capability, and to therefore also produce a blueprint as to how the capability can be used in other immersive audio Unity projects. Consequently, the utilization of the capability tried to adhere to traditional FMOD Audio Middleware and Unity engine workflows. Of course, the complexity of integrating the various software frameworks created certain barriers that could not easily be overcome. However, as will be demonstrated, the only real requirements above and beyond those expected when using the audio middleware, ImmerGo spatialization system, and Unity game engine are relatively small.

Fundamentally, **the goal of the immersive audio capability is to stream audio out of Unity using ASIO, and to enable accurate speaker-based spatialization of the audio using the concept of OBA**. This then presupposes the usefulness of such a feature in a variety of different VR contexts. However, it should be noted that the developed capability has been hypothesized to better complement narrative driven immersive stories rather than other possible application domains. As a result, the overheads of the integrated frameworks assume certain application domains could benefit more from the system than others. Indeed, the accurate spatialization of fast-paced VR games is possible and desirable, but the

impracticality of having a large speaker configuration is often mitigated via the use of headphones. This issue is elucidated in the closing chapter of this thesis.

In addition, the **simplicity of the experience was prioritized** so that variables between the headphone- and speaker-based versions would remain consistent. This then allowed for the only independent variable in the testing procedure to be the different spatialization systems and output devices being deployed, which in turn emphasized the core objective of the VR application in enabling clear comparisons between both headphone and speaker-based spatialization systems while maintaining the core application operation processes.

6.2 The Creation and Implementation of Immersive Planets

This section details the steps taken in constructing the immersive VR application. It therefore informs the following section, which looks to abstract some of the processes used in the implementation of this VR application and formalize these processes to better illustrate how Unity developers would create their own speaker-based immersive audio applications using the developed capability and the ImmerGo spatialization system.

Before considering the various implementation aspects of the VR application, the first area to be examined will be steps taken in creating the celestial scene. Thus, the primary focus of this section will be an examination of the visual layer of the application and the hierarchy and population of the scene with Game Objects. Chapter four may be referred to if clarity is needed for aspects of the Unity and FMOD software frameworks.

6.2.1 Creating the Unity Scene

The VR application is a single scene (replicated for each spatialization system) application that places users in the center of the scene. As mentioned in the introduction, the origin of the scene is the center of a representation of our solar system. A suitable skybox was selected that provides the Milky Way as the backdrop to the scene (see Figure 92). A skybox, as defined by Unity, is "a 6-sided cube that is drawn behind all graphics in the game".[63] In this instance, the skybox was imported as an asset, and is an 8192 x 4096 image of the galaxy that has been converted into the cube map that is used for the texture of the skybox.

To enable Oculus VR development, the *OVRCameraRig* Prefab was imported into the scene. This object represents the First-Person Controller (or camera) and therefore provides the perspective of the scene to the user on application launch. The camera prefab was chosen over the controller prefab that Oculus provides, as the application did not require the user to move outside of the orbiting planets, or into another environment. However, it should be noted that this does not limit flexibility, as the headset still maps a user's real-world movements virtually, i.e., they can still maneuver around the virtual space, and are not forced to occupy a single location. As the experience is a stylized representation of our solar system, there are no other objects present in the scene other than the planets, e.g., comets, space stations. This also adhered to the creative goal of maintaining simplicity and limiting interactions within the virtual world.

[63] https://docs.unity3d.com/560/Documentation/Manual/HOWTO-UseSkybox.html

The planets were simply spherical Game Objects whose textures were created to approximate the colors of the planets in our Solar System. The scale of the planets in our solar system relative to one another was too difficult to accurately implement (see Figure 90). As a result, the planets in the virtual scene were scaled down so they could still be easily panned around the scene. However, some relative difference in size was still maintained (e.g., Jupiter is still the largest planet). Each of the planets were visually rendered according to their unique physical characteristics. This entailed the creation of unique *materials,* halos, and lights for each planet. In addition, the associated logic to facilitate orbiting, and some other critical requirements were also attached to the planets as *Components*. Figure 91 indicates all the components that were attached to each of the planets. In this instance, the planet is Jupiter, but the reader should recognize that each planet would have their associated components tailored to their unique visual characteristics, i.e., Neptune's material involved the use of blue colors for the base map, whilst Mars' material used red colors. Of course, one should not forget that the primary use for the planets was to bind FMOD audio events to them. This process is detailed in the following sections.

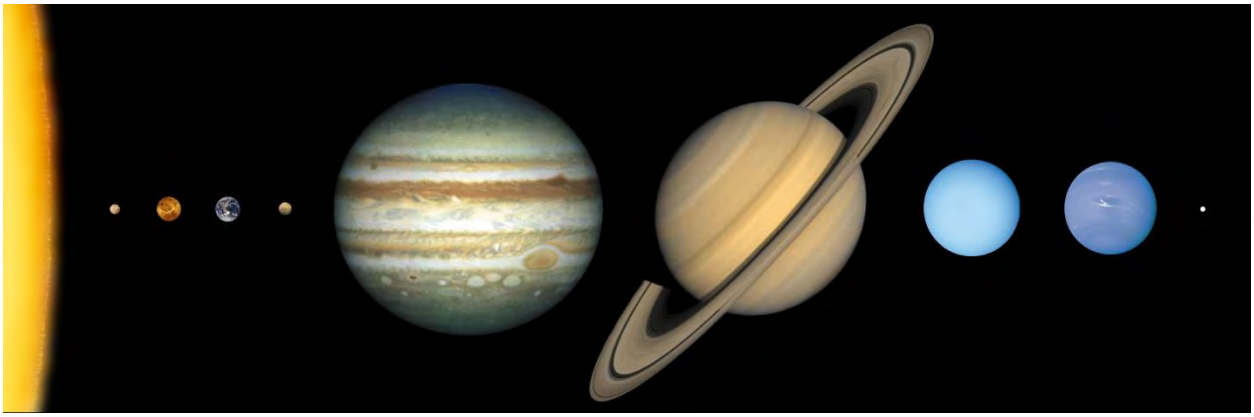

Figure 90: An illustration of the differences in scale between the nine planets in our solar system. This image was taken from NASA's website.[64]

Apart from the materials, the *Sphere Collider Component* enabled the planets to take part in physical collisions, which in turn enabled a user to select and pan a planet in three dimensions. Each of the custom scripts attached to the planet are discussed in the following Game Logic and Custom Scripts section. However, the name of these scripts should provide the reader with an intuitive understanding of their roles.

[64] https://solarsystem.nasa.gov/resources/686/solar-system-sizes/

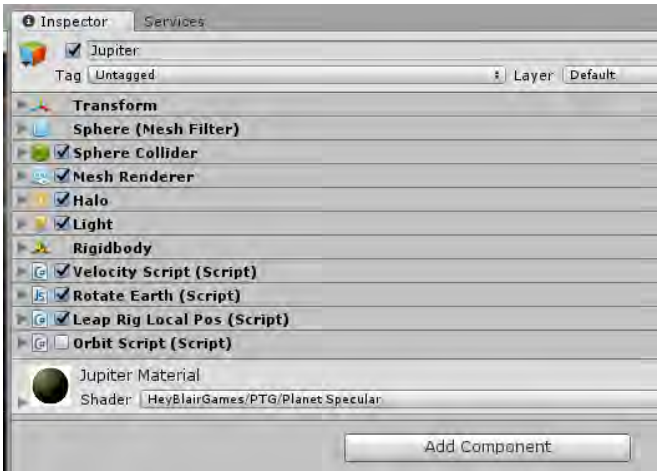

Figure 91: The inspector view for the planet Jupiter. Each planet possessed all of these critical components.

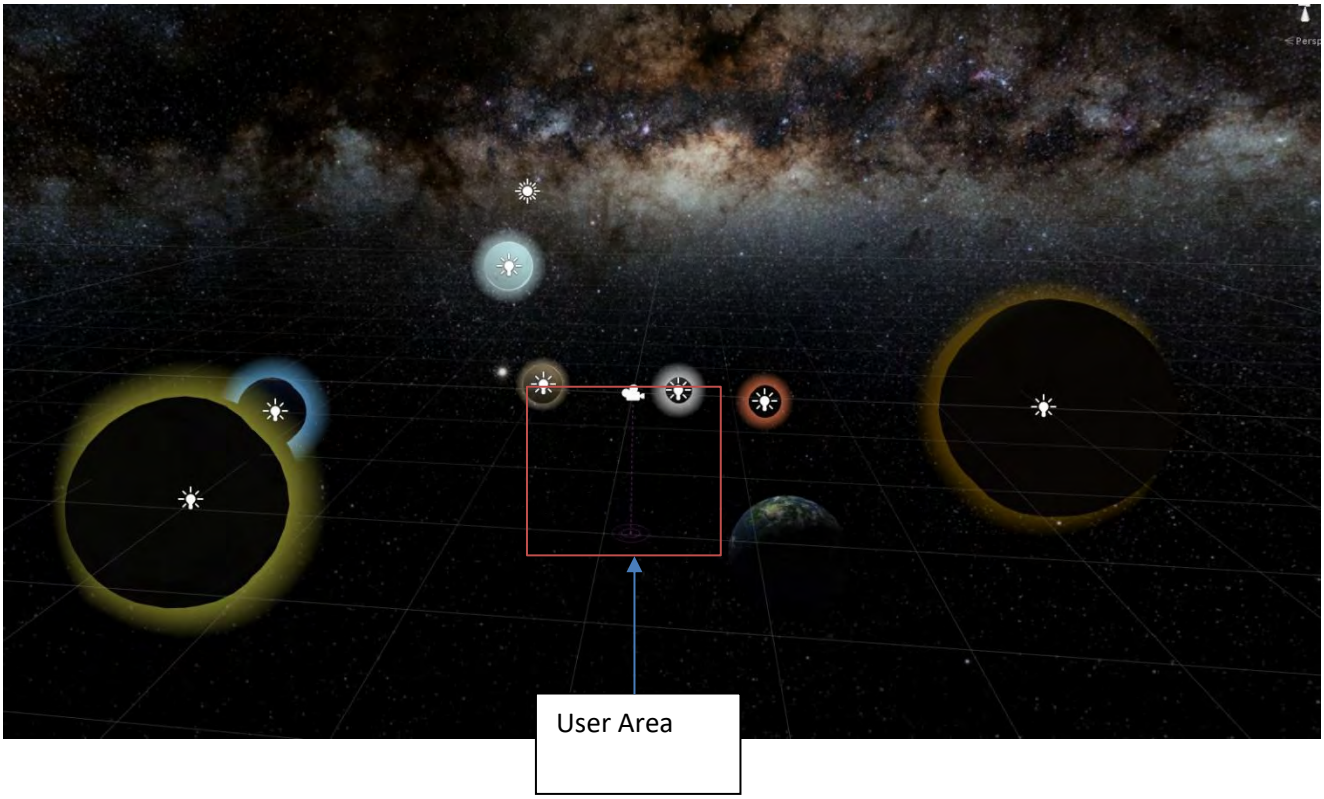

Figure 92: A screenshot from the Unity Editor that shows the scene layout. The camera icon indicates the position of the first-person controller (i.e., where the user is positioned; this area is also demarcated in red).

In Figure 92, the representations of each of the planets in the immersive scene are presented. From left to right, Saturn, Neptune, Uranus, Venus, Mercury, Mars, Earth, and Jupiter are all positioned around the origin. This screenshot is taken from the Editor window, which is the window into the game scene that allows developers to manipulate object transform parameters (i.e., their scale, rotation, and position) and, in real-time, see how these adjustments affect the layout of the scene. The lightbulbs that

are visible on each planet indicate that each planet is a light source, which is how a halo effect for each one is achieved.

Due to the simplicity of the populated scene and the relatively simple implementation requirements, the hierarchy of the scene was easily maintained. Figure 93 indicates the primary and secondary levels of the scene hierarchy. The hierarchy exhibited in this figure pertains to the headphone environment of the VR application, but this hierarchy is consistent across both Unity scenes. It should be noted that the planets were all children of the *Leap Rig* Game Object, which can be seen as the player/user object. This was a requirement of the scene, such that the *localPosition* of the planets is relative to the position of the user, and therefore the origin of the solar system. To recap, it is this position of each planet (game object) which is tracked and transmitted to the ImmerGo spatialization system. Whilst this requirement technically does not matter if the user is positioned at the origin of the scene in world space, it was implemented to add another level of robustness to the solution.

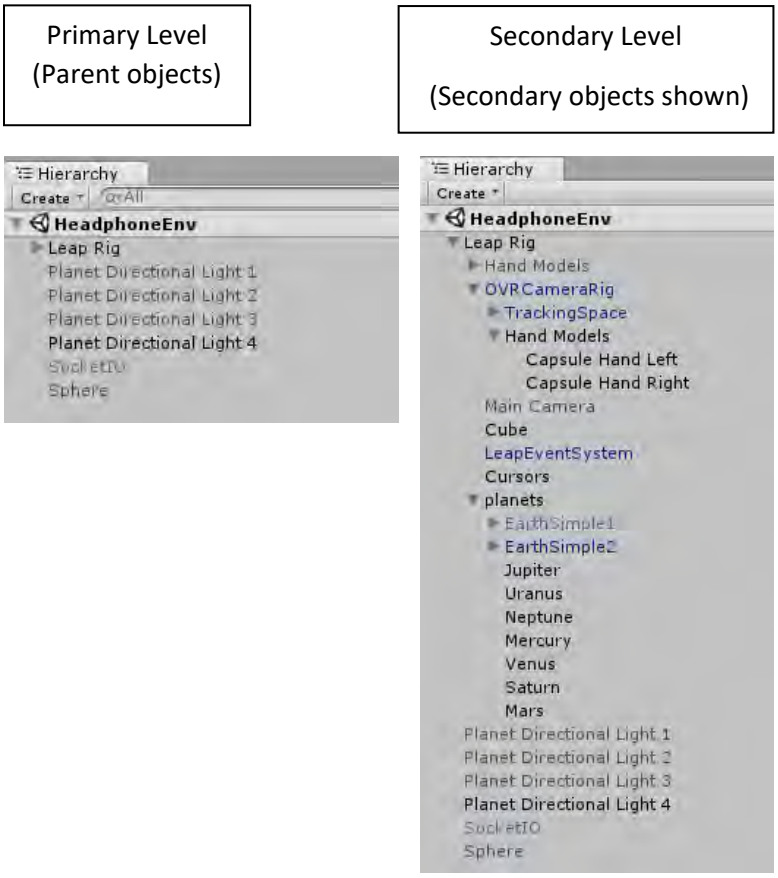

Figure 93: Two screenshots of the hierarchy window of the 'Immersive Planets' application. The left hierarchy indicates the primary layer of objects in the scene, whilst the right image indicates the expanded view of the hierarchy and indicates all the children objects to the Leap Rig object.

The *Leap Rig* object is a prefab provided by the Leap Motion (LM) Orion SDK and facilitates the use of the Leap Motion Controller (LMC) in VR development. It contains all the base functionality that is needed when the application references the underlying Leap Motion API, and service running on the machine. In addition, the *LeapEventSystem* is another prefab that translates information from the *Leap Rig* to the API providing hand and gesture tracking. The *Cursors* object was also provided by the LM

Orion SDK and was edited to better recognize and facilitate the use of three-dimensional cursors. This is expanded upon in the following section on how interaction was implemented for the VR application. The final object of importance in this hierarchy is the *SocketIO* object, which contains information about the network address and port for communication with the ImmerGo spatialization server. Figure 94 provides an overview of the information that was provided to this object. Indeed, this *SocketIO* object is referenced in the modified ImmerGo client script, and can be seen as the intermediary that receives and sends information between the modified ImmerGo client and the ImmerGo server.

The final objects viewable in this hierarchy are the *Directional Light* objects which are used to provide lighting to the Unity scene. Of course, lighting is a fundamental requirement in Unity to correctly indicate how a material should be rendered according to its position in the scene, and what light is incident on the object to which the material is attached.

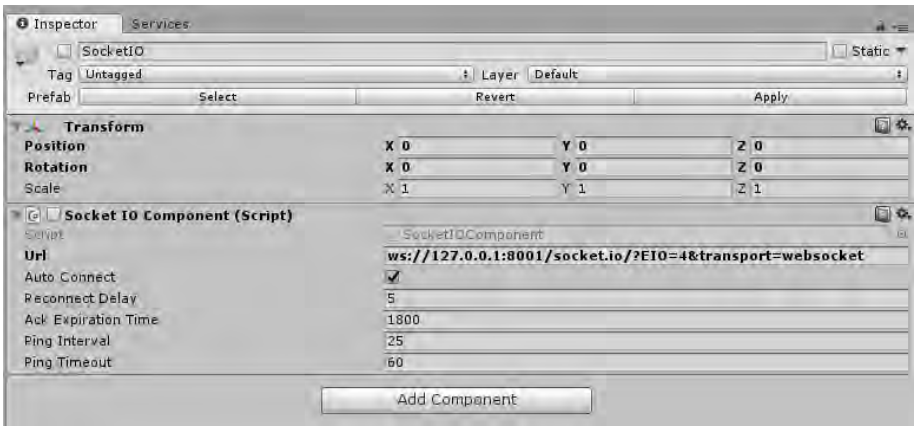

Figure 94: The inspector window of the SocketIO object, and the parameters used for its Unity implementation.

It is worthwhile to review a recording of the Immersive Planets application, to gain a better understanding of the application and how it functions. The actual recording is also attached to the codebase that accompanies this thesis, and should be viewed with headphones as the audio output device. Although significant research was conducted to determine which Unity rendering pipeline should be employed, the default pipeline for Unity 2017.4.11f1 was decided upon to provide adequate rendering parameters and visual fidelity. This pipeline was the lightweight rendering pipeline, which has subsequently become the Universal Rendering Pipeline (URP) in later editions of the Unity engine. A rendering pipeline refers to the "series of operations that take the contents of a Scene and displays them on a screen."[65] In other words, it encompasses the compilation of shaders, materials, and other elements of the scene, and outputting this visual information onto the output display, which in this case is a VR HMD.

6.2.2 Application Interaction

Having provided an overview of the VR application layout and Game Object hierarchy, together with a brief description of the visual components of the planet objects in the scene, it is now important to detail how the mode of interaction for the application was implemented. As has been alluded to, hand-tracking and gesture recognition were used as the primary means of interaction for the user. If the reader has watched the accompanying project video, they will have noticed how the three-dimensional

[65] https://docs.unity3d.com/Manual/render-pipelines.html

cursors were used to attach to objects and pan them about the virtual space. The details of this interaction are detailed below. However, before doing so, a brief analysis of the Leap Motion Orion SDK will be provided to enable some context on how the LMC was utilized within the development of this Unity VR application.

6.2.2.1 The Leap Motion Orion SDK

The desirability of more natural forms of interaction in VR were outlined in chapter two. Of course, the accuracy and precision of these forms of interaction need to compete with traditional interaction methods (controller-based interaction). The Leap Motion controller (LMC) provides hand-tracking and gesture recognition functionality. It was of particular importance to this project, as it facilitated the use of hand-tracking as the mode of interaction in the immersive VR application that was developed. The integration of the Orion SDK (the SDK provided by Leap Motion) into a Unity project is straightforward. Once the Orion API and LM Control Panel[66] had been installed onto the host machine, the Core (Unity) Assets (version 4.4.0) were then imported into the project in the same way Unity assets generally are.[67] These core assets facilitate the use of the LMC's functionality in a variety of different ways. However, the additional *UIInput* [68] modules were of particular interest for this implementation, as they facilitate interaction with Unity game objects and the utilization of a three-dimensional cursor (which is discussed in the following paragraph). Initially, it was thought that a user would be able to reach out and grab a particular planet, and then move it to any location they wished to. However, the limitations of this process were quickly realized, as a user would have to move a considerable distance during the placement of these planets, which was undesirable given that they would be located in a hemispherical speaker-array with physical limitations affecting how far a user could traverse within the array. As a result, the appeal of a form of interaction whereby the user could not only remain within the center of the virtual space, but also pan any of the planets to the outer limits imposed by the speaker-array, was obvious.

The three-dimensional cursor provided a means to circumnavigate these issues. Essentially, the cursor is a rendered sphere that extends from the user's wrist. It was positioned approximately one meter in front of the wrist. The cursor moves in direct relation to how the user moves their hand in three dimensions. Consequently, it enabled an intuitive and easily implementable way in which a user could position any of the planets around them from a single location. Due to the ease with which a pinch gesture can be recognized, this gesture is often employed by hand-tracking software that aims to provide surety when detecting a gesture. Indeed, the reliability of this gesture being detected is high, whilst still enabling medium usability and relatively high levels of comfort.[69] Figure 95 indicates the "pinch" gesture: in the context of this research, a pinch can be defined as a gesture that connects the tip of the index finger to the tip of the thumb. In the diagram, the tips of each finger are indicated by blue dots. As can be seen, when the tip of each finger meets, this can be defined as a successful "pinch". If one reviews chapter two, one should remember the discussion on the Orion SDK, and how, in its implementation, each abstract finger class provides a *TipPosition* and *Direction* vector as variables in the

[66] https://developer.leapmotion.com/releases/leap-motion-orion-400-ssafy-nfdl7
[67] https://developer.leapmotion.com/releases
[68] https://gallery.leapmotion.com/ui-input-module/
[69] https://docs.ultraleap.com/automotive-guidelines/interaction-types.html

class. Therefore, in this instance, a pinch is recognized if the *TipPosition* of the index finger matches that of the *TipPosition* of the thumb.

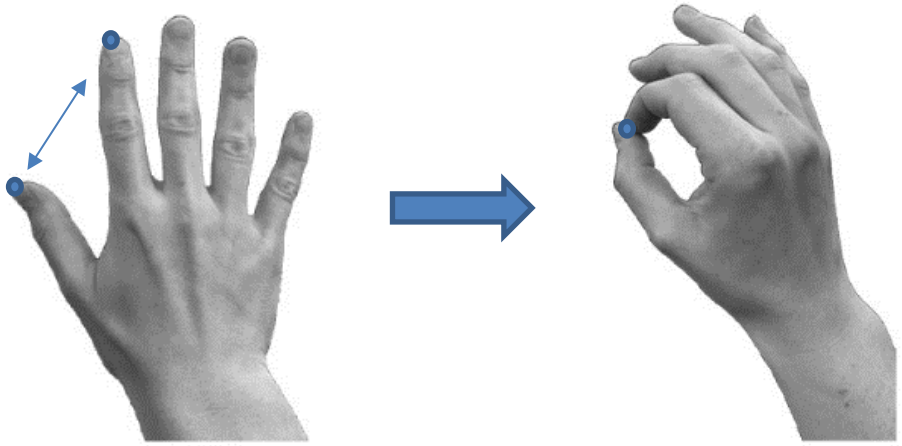

Figure 95: A diagram indicating the "pinch" gesture.

In the implementation of this gesture, a threshold value is compared to the *PinchDistance*, which can be defined as the distance between the two points/tips of the index and thumb fingers. If the distance between the two finger tips is less than the threshold value, then the pinch system detects the gesture as having occurred. This will result in the three-dimensional cursor position being determined and its interaction with other objects being processed. This *PinchDistance* variable is a public variable that can be modified by the developer to find a balance between detecting the gesture, and providing a robust enough solution that the detection algorithm is not too sensitive (i.e., a misclassification).

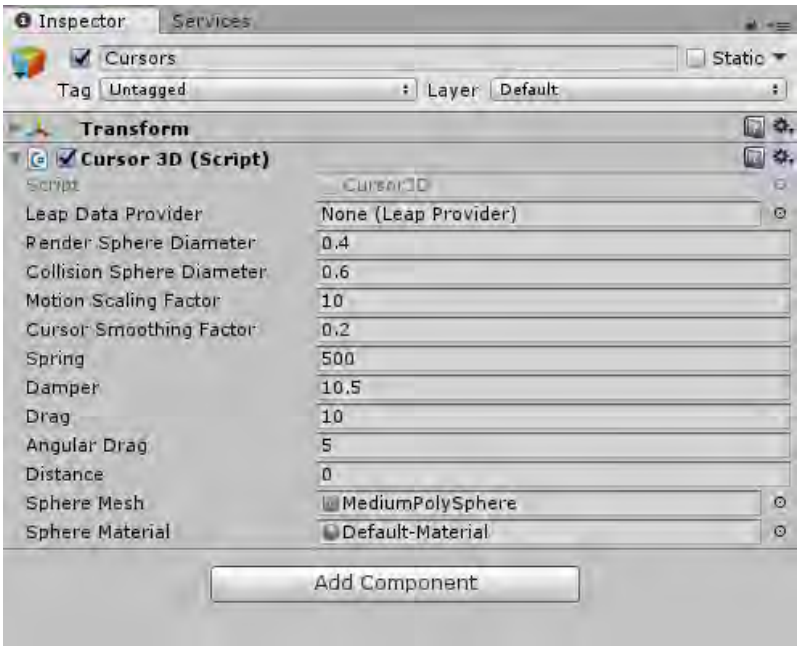

Figure 96: A screenshot of the inspector window that shows the public variables that are exposed to a developer when using the Cursor3D script.

Figure 96 indicates the other parameters that the developer can modify for the three dimensional cursor. They all apply to the size of the cursor, its motion through the virtual space, and how it affects the objects that it selects (the *Drag* variables).

In the three-dimensional cursor implementation, the cursor (sphere) turns green whenever a pinch is detected. This process is demonstrated in the three figures below. The first figure shows this interaction in a sample application that was developed to evaluate the means of interaction (Figure 97).

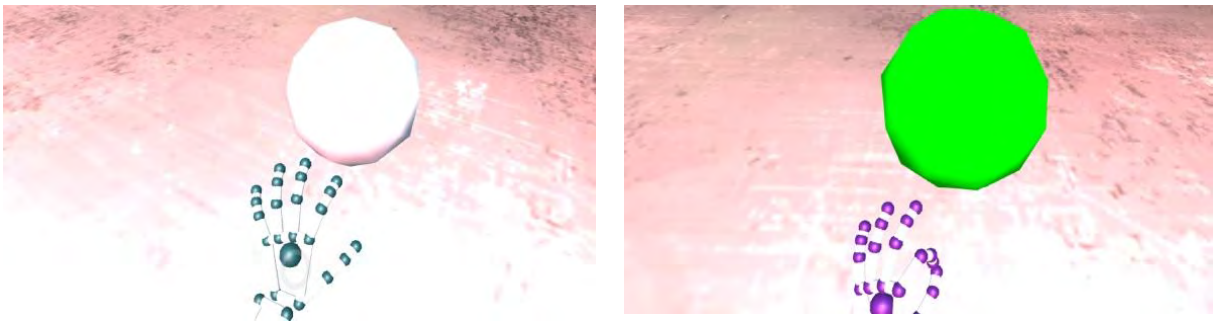

Figure 97: Two screenshots of a sample application in which the three-dimensional cursor was used and positioned about 20 centimeters from the user's hand. The left image indicates the inactive cursor and that the pinch has not been detected, while the right image indicates the active cursor when the pinch has been detected.

Figure 98 and 10 provide the same illustration as Figure 97, except they are provided from the context of the Immersive Planets application. It is worthwhile to mention that, although the LM Orion SDK enables a user's hands to be rendered in VR, it was decided to remove these models (see Figure 97) from this implementation, to promote more focus on the planetary objects and the three-dimensional cursor. Most users of the application had not experienced hand-tracking before, and the ability to see one's hands in VR can reduce the emphasis placed on the visual and auditory aspects of the immersive experience (FiguresFigure 137,Figure 138, andFigure 139 in Appendix D indicate how the planetary scene would look like with a user's hand models being rendered).

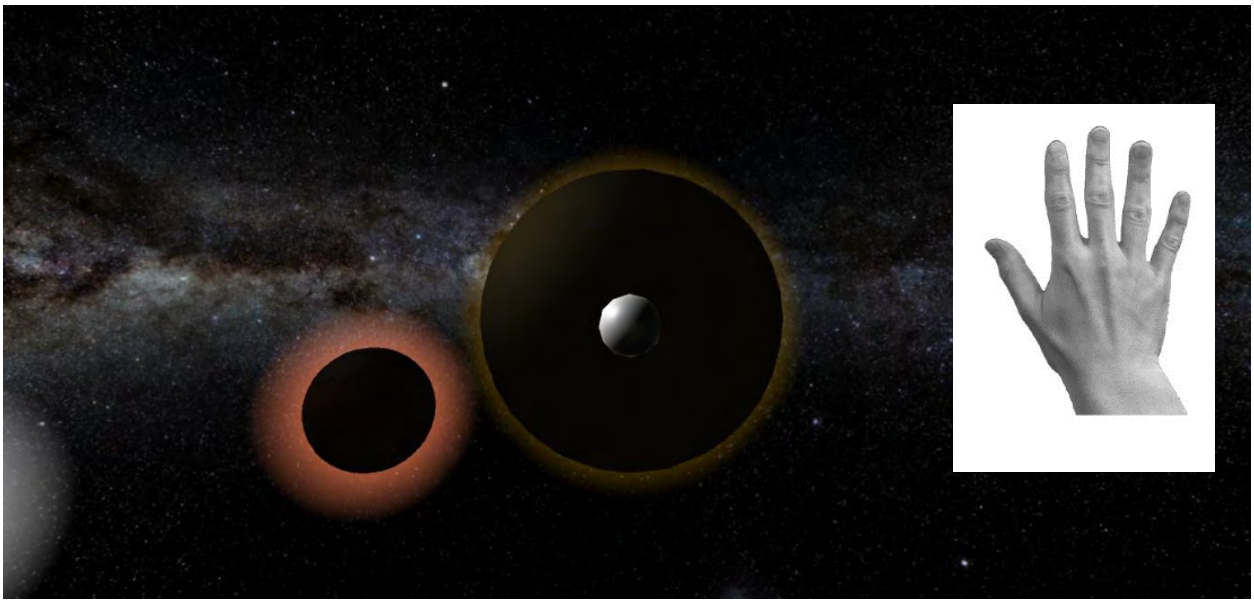

Figure 98: An inactive three-dimensional cursor seen in front of the planet Jupiter. The user's current gesture is indicated to the right of the image. As the gesture is not a "pinch", the cursor can be seen as inactive and white.

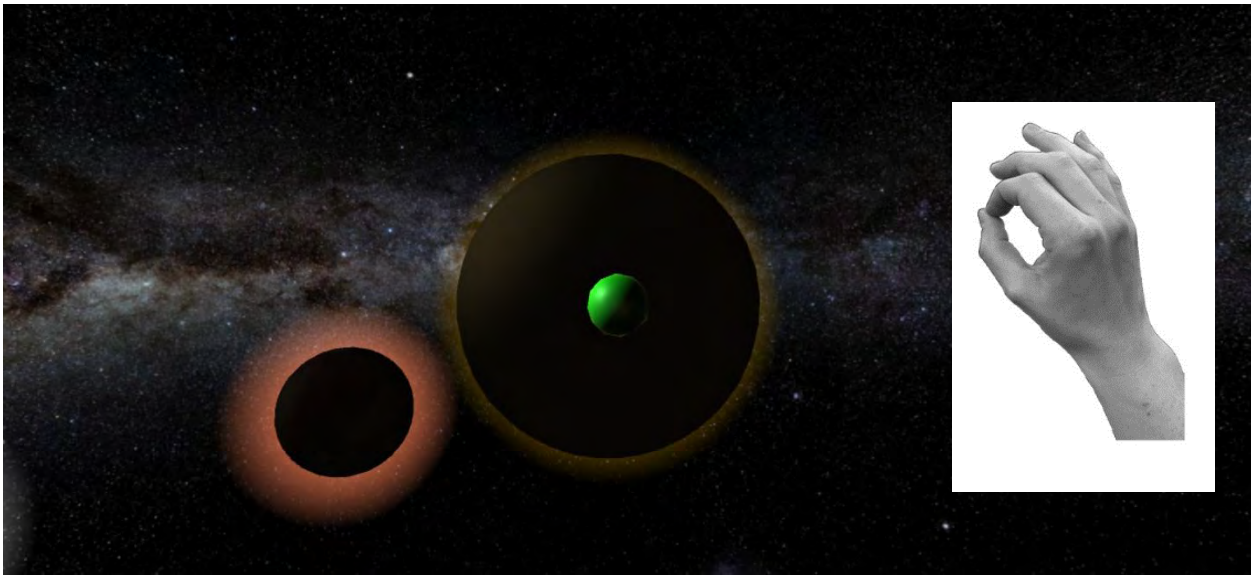

Figure 99: In contrast to the preceding figure, an active three-dimensional cursor is pictured in front of the planet Jupiter. The user's current gesture is once again indicated to the right of the image. In this instance, the "pinch" gesture is being detected, so the cursor is active and is green.

The mode of interaction used in this VR application has been termed "Selecting". This type of interaction was the only form of interaction that was implemented in the experience, and it refers to the ability of the user to select and pan the planets, and therefore audio sources, about the scene. If the three-dimensional cursor collided with one of the planets, and the user performed the requisite "pinch" gesture, the planet is "bound" to the cursor.

In terms of the actual implementation, a collision is recognized by performing a *Physics.OverlapSphere* function call that returns a list of all the *Colliders* which the three dimensional cursor is touching, or near

to. Indeed, this function is only called when a "pinch" gesture is detected. If one recalls, each of the planets have their own *SphereCollider* which will be returned by this function if they do indeed overlap with the cursor. Upon recognition of a planet's collision with the cursor, the planet is checked to see whether its *rigidbody* is Kinematic. A *rigidbody* is defined by Unity as the "control of an object's position through physics simulation."[70] The *isKinematic* condition determines whether the Unity physics system should affect the *rigidbody*. As this condition is false for all planets, they are affected by the Unity collision system, and they are successfully bound to the cursor. Susbequent *DragObject* function enables the planet's position to be altered according to the position of the cursor. The parameters of this function are dictated by the developer's decision to alter the public variables exposed by the *Cursor3D* script (Figure 96).

As soon as the user is no longer "pinching", the planet is detached from the cursor, and remains in the position at which it was detached. This process or attachment/"selection" is indicated in Figure 100. The planet being panned in three dimensions is indicated in Figure 101, and Figure 102 shows the planet being "deselected" by the cursor when it is detactivated, as the user is no longer pinching. In each of the referenced figures, a representation of the the user's gesture is indicated to the right of each image. As with the preceding section, a review of the accompanying application video will provide the reader with a useful insight into this mode of interaction.

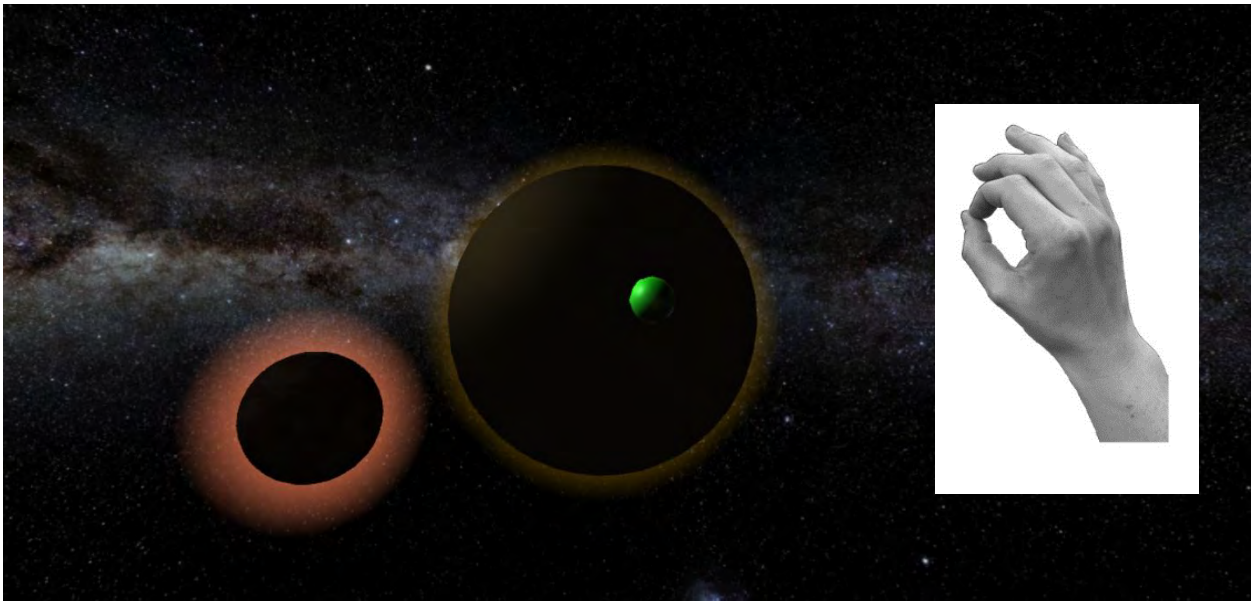

Figure 100: A screenshot from Immersive Planets that indicates the three-dimensional cursor colliding with the planet Jupiter and being activated. At this step in the interaction process, the planet is "bound" to the cursor and has been "selected". As a result, it will follow the cursor if the pinch gesture is maintained.

[70] https://docs.unity3d.com/ScriptReference/Rigidbody.html

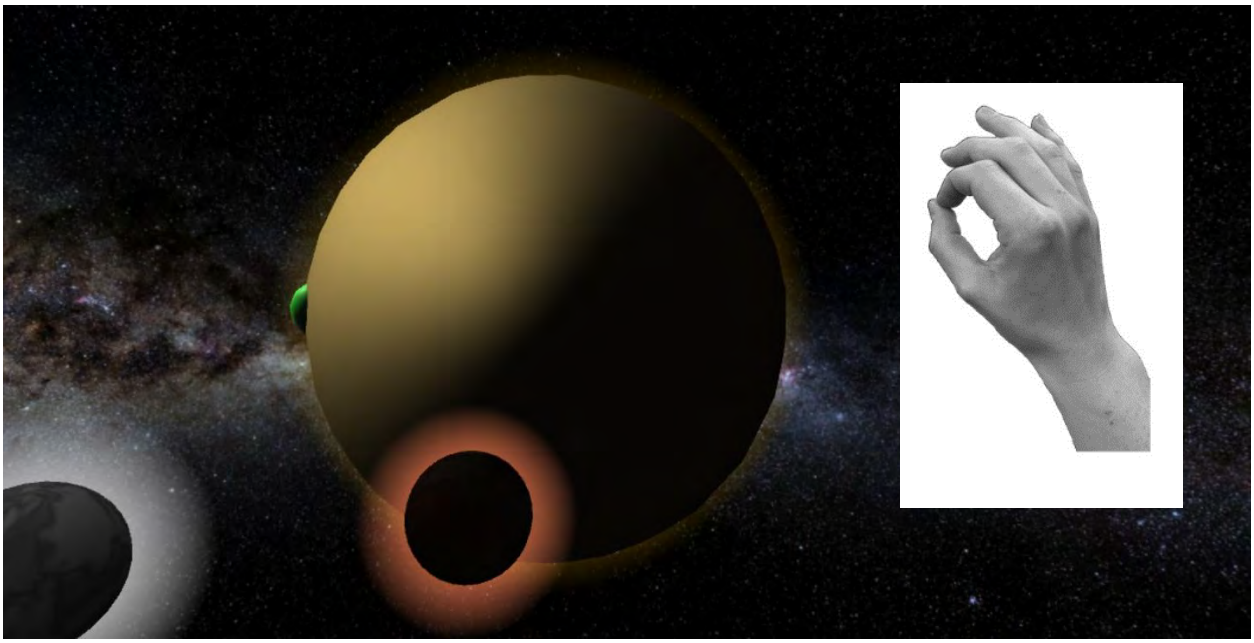

Figure 101: Another screenshot of the VR application that indicates Jupiter being bound to the cursor, and being positioned behind the planet Mars, whilst the three-dimensional cursor is active.

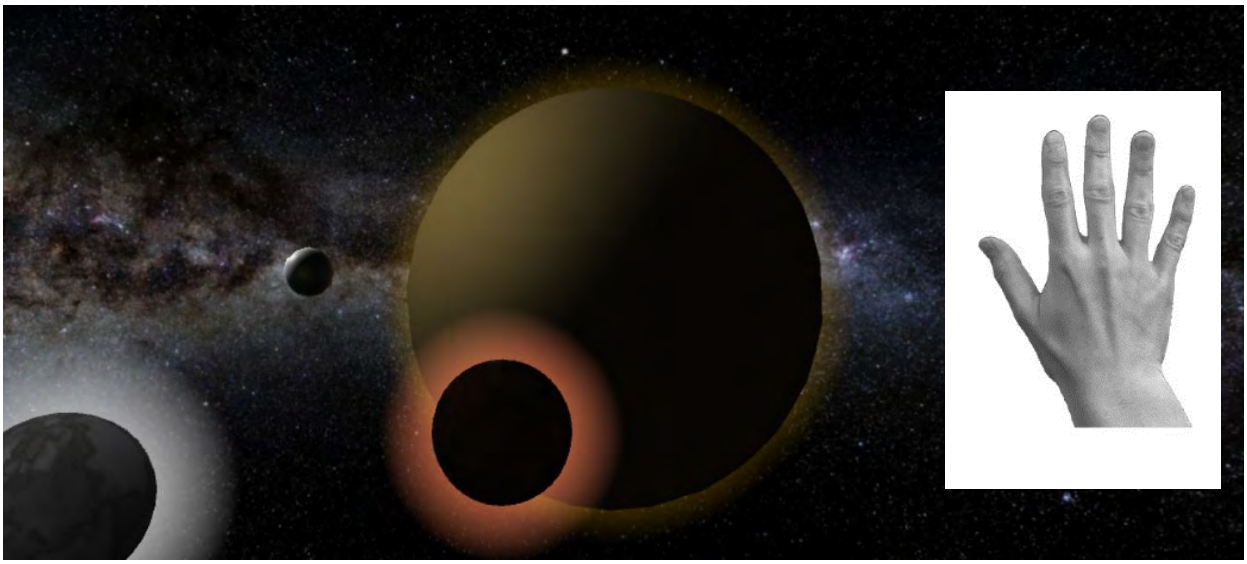

Figure 102: This final screenshot indicates how once the user has released their pinch, the planet is deselected, as the cursor is no longer active (white), and therefore the planet has been unbound from the cursor.

Of course, there are many uses for gesture and hand detection, as any number of gestures and therefore events can be triggered in a Unity scene. Although different gestures were implemented in iterations of the VR application, the robustness of the pinch gesture and the simplicity of the interaction needed by the user, made it the gesture implementation of choice.

6.2.3 Game Logic and Custom Scripts

Having indicated the blueprint of the Immersive Planets scene, together with the mode of interaction that was utilized for this application, the following section will provide some clarity on how the application's game(play) logic was implemented. Game logic can be defined as the core mechanisms of a game that enable the functionality of all the aspects of the game to synchronize and work together. The next section will therefore provide an overview of the custom C# scripts that were developed to provide the core functions of the VR experience. These core functions can be further grouped into two domains: project functionality and gameplay mechanics. Section 6.2.3.2 below will provide an overview of the essential logic that was needed to utilize the immersive audio capability for this VR experience, whilst the subsequent Gameplay Mechanics section will discuss the logic of the Immersive Planets application. As has been mentioned, most of this logic was implemented through the development and implementation of custom scripts that are employed within the Unity scene. However, due to the utilization of the FMOD Audio Middleware solution in the implementation of the immersive audio capability, the first of these sections will examine the construction of the FMOD project that housed the audio of the Immersive Planets VR application.

6.2.3.1 Integrating FMOD and its Workflow

Audio Middleware was outlined in chapter four, and the implementation of an FMOD Compliant plugin (*ImDSP*) was analysed in chapter five. This section will briefly discuss the authoring of the eight FMOD audio events that were bound to the eight planet game objects in the VR application. It is assumed that the reader knows how the project was integrated with the FMOD solution (as detailed in chapter four).

Due to the nature of the experience, a suitable score needed to be found, and the soundtrack for a similar immersive experience, entitled SPHERES,[71] was selected. The score was produced by Kyle Dixon and Michael Stein, who are predominantly known for their contributions to the Stranger Things series.[72] The atmospheric and moody ambience of the score made it an ideal candidate for the audio tracks used for each of the planets. The audio was imported into the FMOD audio middleware and downmixed to be monophonic tracks, which is a prerequisite for both the developed immersive capability and the Oculus Spatializer FMOD plugin. The screenshot below indicates the hierarchy of the FMOD project (see Figure 103). Once each of the FMOD events had been authored, the desktop platform was selected from the build platforms, and the sample rate was set to 48 kHz, to mirror the requirements of the ASIO Playback server detailed in chapter five. After this, the project was built into the FMOD sound bank, which is referenced from inside the Unity project. Figure 103 indicates that there are separate FMOD audio events for both speaker and headphone environments. The differences between them are detailed in section 6.3.

[71] https://www.oculus.com/experiences/rift/1859625197439973/

[72] It should be noted that this project was conceptualized in 2017, and the Immersive Planets VR application was developed in 2018, so the similarities between SPHERES and the developed application are purely coincidental. Indeed, as Immersive Planets was an application used solely for research, the developer did not profit from the development of the application in any way.

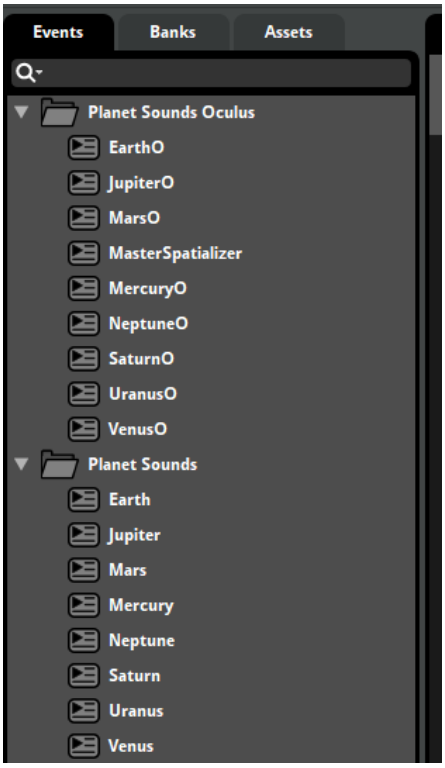

Figure 103: The hierarchy of the Immersive Planets FMOD project.

As previously mentioned in the previous chapter, the buffer size of the ASIO Streamware driver and ASIO Playback server need to be set to 512. Similarly, before the FMOD low level system is initialized, the DSP buffer size must be queried and set to 512 if necessary.

6.2.3.2 Project Functionality

This section will outline the three custom scripts that were developed to facilitate the utilization of the immersive audio capability in the application's implementation, and how they function.

All three of the custom scripts were developed to exist in the same life cycle of the *Leap Rig* Game Object. This therefore denotes that they were each attached to the *Leap Rig* object as *components* to the object (see Figure 104).

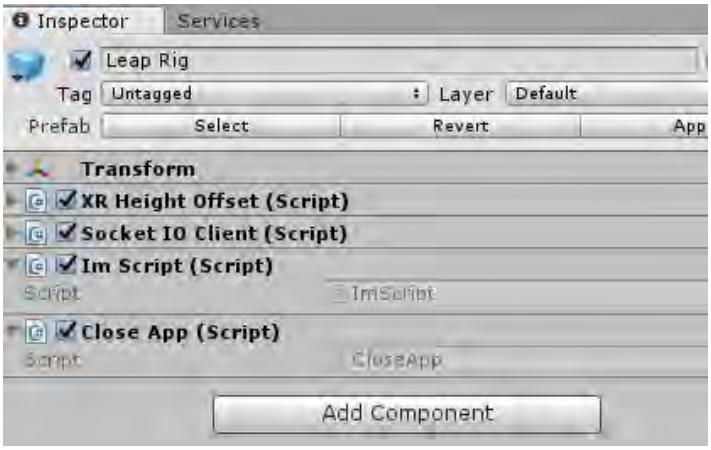

Figure 104: Leap Rig Game Object Components. The SocketIOClient, ImScript, and CloseApp custom scripts contained the fundamental logic that informed the integration of the Immersive Audio Capability into the Immersive Planets VR application.

6.2.3.2.1 The VR ImmerGo client (SocketIOClient script)

The previous chapters should have provided an understanding of how the ImmerGo spatialization system enables spatial audio for both a DAW and the FMOD Audio Middleware system. Indeed, chapter five specifically provided an analysis of the *SocketIOClient* script. This section will not reiterate these concepts, but rather indicate how this functionality manifests as a custom script that is deployed in the Immersive Planet VR application. Figure 105 thus shows the inspector view of the custom *SocketIOClient* script and indicates the population of a Game Object array with the eight planets. In the instance of the script on the left, the array of game objects is constructed by the *Start* function, but in other iterations of the script (the right image), the array is indicated to Unity developers through the Unity Editor Inspector window.

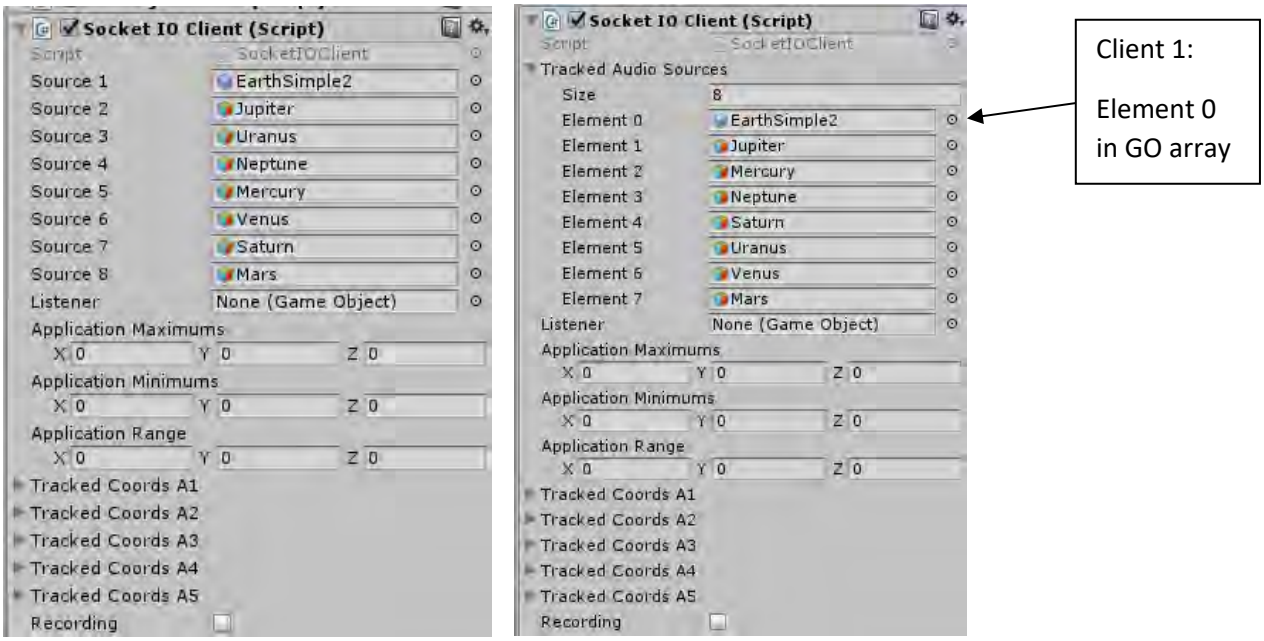

Figure 105: The inspector view of the SocketIOClient Script. The eight audio sources indicated by the developer represent the eight audio sources whose positions are tracked and transmitted to the ImmerGo spatialization server.

In addition to the audio sources whose positions need tracking, the application ranges should be supplied to the script, so that the curation methods (detailed in chapter five) can be correctly processed by the ImmerGo spatialization server. The curation functions were employed to enable the virtual audio sources to be correctly placed within the ImmerGo speaker array, regardless of the unit scale a developer might employ within their Unity application.

An additional requirement of this *SocketIOClient* script is that the *SocketIO* game object (see Figure 93) needs to be correctly configured, as it is referenced internally to facilitate the socket communication between the client script and the ImmerGo server. This entails providing the object with the correct URL for the ImmerGo server, and various other socket-based connection settings (see Figure 94).

6.2.3.2.2 ImScript script

The *ImScript* script was implemented to begin FMOD audio event playback. As was indicated in chapter four, FMOD events can be initialized by script, as opposed to attaching *FMOD Studio Event Emitter* scripts to Game Objects. Though this route of initializing FMOD audio events adds additional complexity to systems, it facilitates the fine control of the event, which in this implementation was a necessary aspect due to the implementation of a custom FMOD DSP plugin. The *ImScript* script therefore performed the following functions:

- Enabling FMOD audio event playback.
- Attaching *ImDSP* plugin (custom FMOD DSP plugin) instances to the audio events.
- Providing the client number/FMOD audio event identifier for each specific audio event, thereby initializing the identifier that is used by the playback server to assign audio events to output channels.
- And, finally, allowing the ImmerGo server to correctly identify which event should be positioned in real time.

The *ImDSP* plugins referred to above were detailed in the preceding chapter and facilitated the streaming of audio out of Unity and to the ASIO Playback Server. An account of the implementation of these tasks is available in the accompanying code-listing for this script.

Code Block 12 indicates the initialization steps for the Earth FMOD audio event. One of the important considerations in this implementation of the *ImScript* script was that FMOD audio events are created so that DSP effects apply to channel groups, and do not reference these events directly. This is a facility that the FMOD solution provides to enable DSP instances to apply to numerous different events that can belong to more than one group. In essence, this enables a single DSP instance to modify the signal of multiple events at once, which is a desirable feature as it reduces the overheads needed to initialize multiple DSP plugins for several FMOD audio events. Of course, in the context of *ImDSP* plugins, each event requires its own plugin. As a result, the audio events for each of the planets are unique groups to which the *ImDSP* instances are uniquely applied (see *EarthInstGrp* object in the following code block).

```
        FMOD.Studio.EventDescription EarthDesc;
        FMOD.Studio.EventInstance EarthInst;

        RuntimeManager.StudioSystem.getEvent("event:/Planet Sounds/Earth", out EarthDesc);
        EarthDesc.createInstance(out EarthInst);

        plugRes =RuntimeManager.LowlevelSystem.createDSPByPlugin(ImDSPHandle, out ImDSP);
        EarthInst.getChannelGroup(out EarthInstGrp);

        plugRes = EarthInstGrp.addDSP(0, ImDSP);            //cli 1
        plugRes =ImDSP.setActive(true);

        int trackID =1;                     //ClientNumber starts from 1 (on ASIO server Side)
        IntPtr trackIDPtr = Marshal.AllocCoTaskMem(Marshal.SizeOf(trackID));
        Marshal.WriteInt32(trackIDPtr, 1);
        ImDSP.setUserData(trackIDPtr);
        trackID++;

        EarthInst.start();
```

Code Block 12: The code block above indicates all the steps involved in firstly creating an instance of the Earth FMOD audio event, secondly binding an ImDSP instance to the event, and finally providing the ImDSP plugin with the correct identifier for the FMOD event.

The code block above pertains to the Earth Object and audio event. However, the reader should recognize that this process is applied to all event instances in the VR application. Once all the FMOD audio events are correctly initialized, each of the events' playbacks is initiated when the application is launched (by using the *Start* function).

6.2.3.2.3 CloseApp script

While the previous section detailed the steps taken to bind *ImDSP* instances to FMOD audio events, the *CloseApp* script was implemented to provide control over audio event playback. It was primarily developed for the experimental setup that was utilized in attaining the results that were necessary to realize the primary aim of this research. As such, a more comprehensive look into its functionality is provided in the chapter seven. Nevertheless, it is necessary to briefly describe its function.

The CloseApp script can play back all audio tracks at once or mute all audio tracks in the scene. As a result, it also provides:

- Playback control for specific audio events and does this by muting all the other active events in the scene, i.e., to solo a specific track.
- It controls the orbiting function of each of the planets.
- It is used to launch the ASIO playback server from within the Unity application after the application was built for testing, or to close the Unity application upon the experience's completion.

A byproduct of this playback implementation is that all the FMOD audio events were set to loop indefinitely, or until the application was closed. Of course, this does not model a direct use-case for events that are activated non-linearly, but the capability does encompass this potentiality, as the ASIO

server will simply stop receiving and supplying audio samples to the specific output channel for an FMOD audio event that has stopped playing. If an audio event is triggered non-linearly, then the developer needs only apply an *ImDSP* instance to this event (with an identifier that has been released from an event that has stopped playing).

At present, the ASIO playback server exists as a compiled console application (as per the ASIO SDK specification) that a developer needs to run before launching their project. The launching of this application was tied to the execution of the Immersive Planets application. This was accomplished using the *Process.Start* command that Unity supplies developers and required the server application to reside in the same file path as that of the compiled VR application. This code segment is indicated in Code Block 13, and was deployed from within the *CloseApp* script.

```
System.Diagnostics.Process.Start(Application.dataPath + "/Winsock Server2");
```

Code Block 13: The code segment indicates the command used to launch the ASIO Server Application ("Winsock Server2").

The *CloseApp* script functionality was hidden from the participant in the experiment and was developed to allow the experiments to be conducted without needing system input from the user in VR when determining the spatial accuracy of each spatialization technique. The control mentioned above was provided using the Unity *Input* module, which responds to keypresses on the keyboard of the computer from which the application is launched.

6.2.3.3 Gameplay Mechanics

Having outlined the three primary components that enabled the immersive audio capability to function within the Immersive Planets VR application, this section will briefly discuss how some of the gameplay mechanisms were implemented. The *Gameplay Mechanics* scripts (Unity *Components*) are custom scripts bound to each of the planets (see Figure 91), but this section excludes the interaction mechanism which informs the core gameplay mechanics of the immersive experience and is presented in section 6.2.2.

The primary gameplay mechanics of the developed application is the ability of the planets to orbit about the origin of the scene (the place at which the user is positioned). The following code block indicates the logic to enable this orbiting of each of the planets about the user. As the planets orbited around the user on the horizontal axes, the x and y values for each planet were determined according to the elapsed time, and updated on every *Update* call, i.e., once per frame.

```
        timeCounter += Time.deltaTime * speed;

        float x = Mathf.Cos(timeCounter) * width;
        float z = Mathf.Sin(timeCounter) * breadth;

        transform.position = new Vector3(x, y, z);
```

Code Block 14: The orbiting function that applies to each planet.

The speed, width, breadth, and height (y value) of each planet were public variables that were modified to provide different orbiting speeds and orbital paths for each planet. These variables can be seen in Figure 106 below.

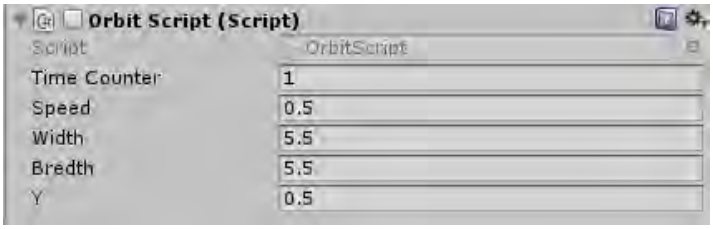

Figure 106: The inspector view indicating the different variables that affect the orbit of a particular planet.

Each instance of the *OrbitScript* on each planet was inactive upon application initialization, and its activation was controlled by the *CloseApp* script mentioned above. Another important attribute of the planets was that they should still be able to be "Selected" by the user even if they were orbiting. As a result, the three-dimensional cursor implementation was modified to deactivate the orbiting script, if a planet had been 'Selected" by the user. Upon "Deselection", the planet's *OrbitScript* remained disabled, thereby leaving the planet at the position the user had decided.

The *RotateEarth* script perfoms a similar role to that of the orbiting script, except that it modifies the rotation of the planets through time, as seen in the following code snippet. The script refers to the Earth game object, but its function was generalized to rotate whatever object it was attached to. The utilization of the *transform*.*Rotate()* function pertains to the *transform* of whatever Game Object the script is attached to.

```
transform.Rotate(Vector3(0, Time.deltaTime * speed, 0));
```

The *VelocityScript* was used to always set the planets' velocity to zero. This was implemented to prevent planets from floating out of reach when users were selecting them. Similarly, the *LeapRigLocalPos* script acted as a helper script to ensure that a planet's parent is the *Leap Rig* object. An issue emerged when it was found that, upon selection, the planets would sometimes disobey the primary hierarchy of the scene and would no longer be children to the *Leap Rig* object. Thus, this script was implemented to maintain the parent-child relationship of the *Leap Rig* and planet Game Objects.

6.3 Consistency Across Both Spatial Systems

Having discussed the primary project functionality and gameplay mechanics, some of the critical differences between the speaker- and headphone-based spatialization systems and their implementations in the immersive VR application will be discussed. Of course, as was indicated in section 6.1, the simplicity in the design of the application enabled the consistency across both systems to be maintained. Indeed, once all the gameplay and interaction mechanics had been realized when employing the speaker-based immersive audio plugin, the scene was simply duplicated and modified for the headphone equivalent (see Figure 107). The resulting scenes were therefore entitled the *SpeakerEnv* and the *HeadphoneEnv* to denote the difference in audio output system.

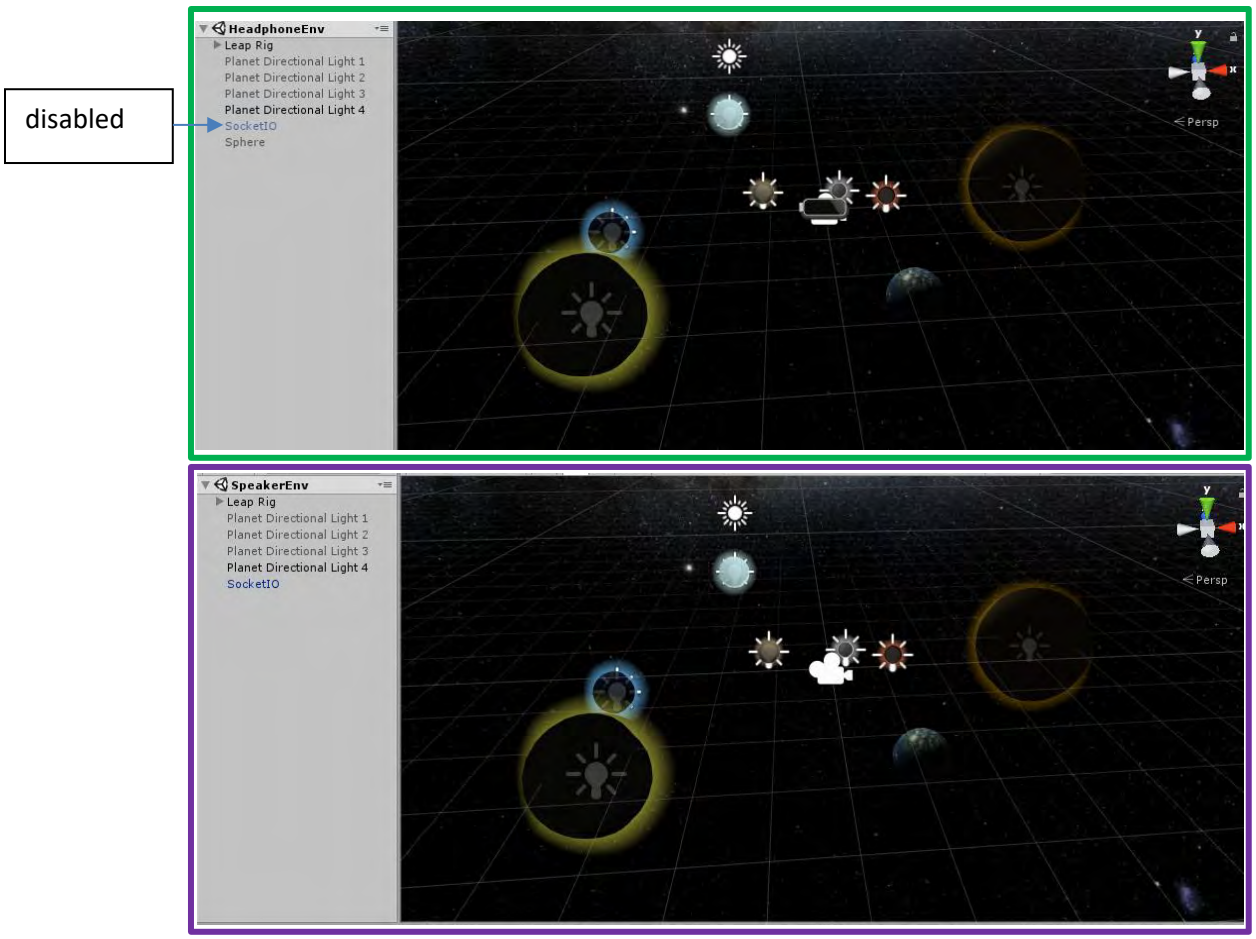

Figure 107: A comparison of the two Unity scene and hierarchy views for each of the spatial audio implementations. The green box represents the headphone environment, while the purple box represents the speaker environment. As can be seen, the SocketIO Game Object is disabled in the headphone environment. It was left in the environment (albeit inactive) to better differentiate the similarities and differences between both scenes.

The differences between the two scenes are visually indicated by the differences between the two *Leap Rig* objects, which are indicated in Figure 108. This figure should be referenced when reading the following two sections.

Headphone Environment

Speaker Environment

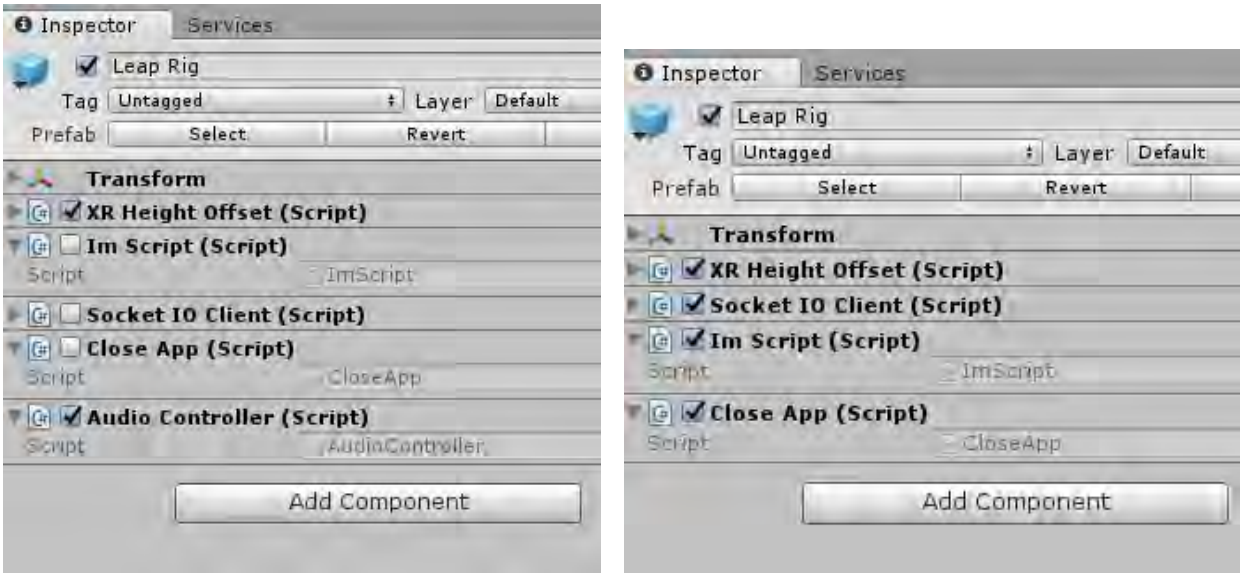

Figure 108: The inspector window for the Leap Rig game object in each environment.

While the previous figure indicates the differences between the two environments with regard to the Unity components that are attached to the main camera, the differences between the two scenes (environments) are more formally expressed in the following list:

- The playback control of the FMOD audio events was handled differently, as was the construction of the FMOD audio events.

- The headphone environment obviously did not need to encompass the network communication that the immersive audio capability requires. Consequently, the *SocketIO* game object and the *SocketIOClient* script (the modified ImmerGo client) were both disabled in the headphone environment (Figure 108 illustrates that the disabled *SocketIOClient* script).

- Finally, the playback control for each of the environments was enabled through two different scripts (see section 6.3.2). The headphone environment used the *AudioController* script and the speaker environment used the *ImScript.* Figure 108 once again illustrates this.

6.3.1 Contrasting Approaches to Assigning DSP Plugins to FMOD Audio Events

One of the fundamental differences between the two environments occurred when authoring FMOD audio events. Figure 103 shows the two sets of audio events that were created. The *Planet Sounds Oculus* audio events are applicable to the headphone-based implementation of the immersive VR application, while the *Planet Sounds* audio events are applicable to the speaker-based implementation. This grouping of events for different output systems is not unlike the groupings that would be implemented in a digital game or VR application in which the sound designer has provisioned different sets of events for different output configurations (stereo, 5.1, 7.1, etc.) or encodings of the audio tracks (monophonic, stereophonic, quadrophonic, etc.).

According to the requirement of the FMOD DSP implementation, i.e., that DSP plugins need to be applied to channel groups, the *ImDSP* plugins were applied to FMOD audio events (channel groups) through the *ImScript* script inside Unity. This was a conscious decision that enabled the developer to better debug issues that arose in the plugin's development. However, it should be recognized that the *ImDSP* plugins could just as easily have been applied to the FMOD audio events from within the FMOD Studio Application. In this scenario, the relevant parameter description callbacks are needed to be correctly implemented to enable the Studio Application to assign the unique identifiers required for each event on a particular event's bus, i.e., the DSP chain.

The assignment of an event identifier from within the Studio Application was undesirable, as it made it difficult to remain consistent with the audio event identification process from the *coordinate transmission* component of the immersive audio capability (assigning Game Objects to an array whose index determines their client identifier). Therefore, the *ImDSP* plugins were applied to FMOD audio events from within Unity. This process of assigning DSP plugins to audio events from outside the Studio Application became one of the major differences between the headphone- and speaker-based implementations, as the Oculus Spatializer was applied to FMOD audio events from within the FMOD Studio Application, and not from within Unity.

The FMOD Oculus Spatializer plugin forms part of the larger Oculus Unity Audio SDK. This SDK's documentation provides a brief overview of the necessity of audio in VR application development to better facilitate a user's sense of immersion. Additionally, the documentation indicates that the Oculus Spatializer is an HRTF-based spatialization system (Oculus, 2021). The actual FMOD C++ project for this plugin is not openly available to developers, but it can be assumed that it is consistent with the requirements of an FMOD Effect Module plugin. Indeed, if one considers how HRTF spatialization is generally achieved, the Oculus Spatializer will essentially apply a pair of HRTF functions (one for each ear) to a monophonic signal provided by the FMOD mixer in real time (this process is detailed in chapter three). The three-dimensional attributes of the audio event would then be provided to the plugin, so that it can correctly identify the HRTF pair that best aligns with the position of the audio event relative to the listener.

When using the Oculus Spatializer for the headphone-based implementation of Immersive Planets, the following process was adhered to. The spatializer plugin was copied to both the root of the FMOD Studio Application's plugin folder, as well as the plugin's folder for the Unity application (Figure 109 indicates both the FMOD Studio File Root at the top of the image, and the Oculus Spatializer plugin's location in the Unity application file system). This is the same process that was performed for the *ImDSP* plugin. After the *ImDSP* plugin had been compiled through the C++ project, the resultant .dll (plugin) file was copied to the Plugins folder within the Unity project.

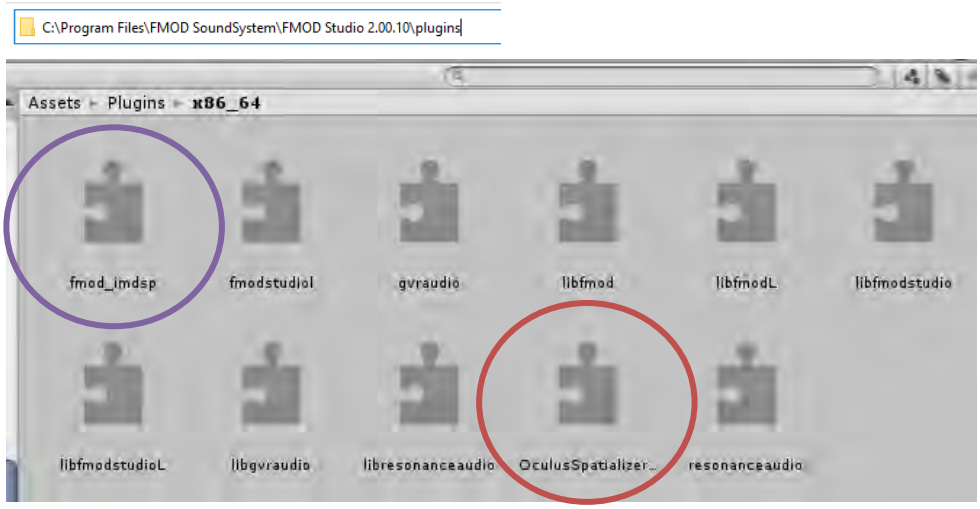

Figure 109: A screenshot indicating the root of the FMOD Studio Application, as well as the Plugin Folder for the Immersive Planets VR application. In the Unity Plugins folder, both the ImDSP (purple) and OculusSpatializer (red) plugins are indicated.

Once the Oculus Spatializer and *ImDSP* plugins had been imported into both the Unity engine as well as the FMOD Studio Application, both Unity application environments (*Speaker* and *Headphone*) could reference the plugins internally. The *ImDSP* plugin was referenced from within Unity, whilst the Oculus Spatializer plugin was referenced within the FMOD Studio Application.

In order to apply the Oculus Spatializer to an FMOD Audio Event, a develoepr is required to attach the spatializer onto the DSP chain of the particular audio event. Figure 110 indicates the appearance of the Oculus Spatializer on an FMOD audio event in the Studio Application when it is applied to an FMOD audio events bus:

Figure 110: The Oculus Spatializer as it appears on the DSP chain of an FMOD audio event.

As the figure indicates, the Oculus spatializer enables the developer to apply reflections as well as signal attenuation to the audio event from within the Studio Application.

The only requirement of the *ImDSP* plugin is to ensure that the relevant audio event(s) are three-dimensional (3D) events, so that the FMOD mixer recognizes that it is an audio event that needs to be spatialized. This therefore enables the event to contain *3D_Attributes*, which is a prerequisite of 3D FMOD audio events. In fact, another reason this is a requirement is to enable the native FMOD Spatializer to be applied to the event so that the event contains the attenuations/envelopment

parameters that dictate its gain values, depending on the distance from the audio source to the listener (see Figure 111).

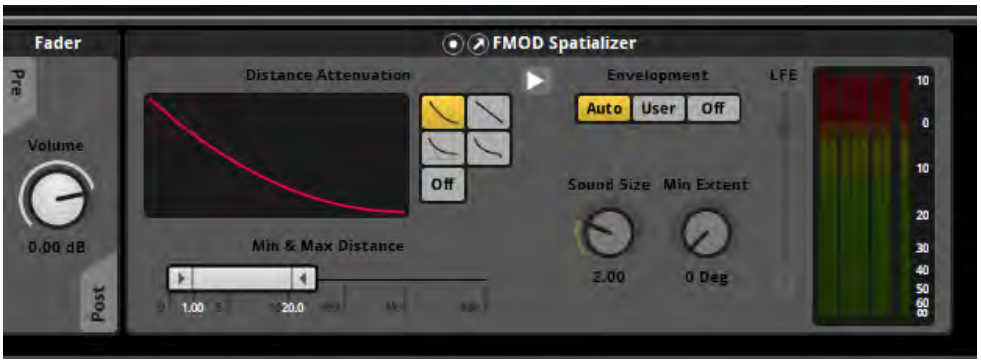

Figure 111: The native FMOD Spatializer that is attached to 3D audio events by default. This screenshot similarly shows the Spatializer attached to the DSP chain of an FMOD audio event.

Having applied either the Oculus Spatializer or the FMOD Spatializer to each of the FMOD audio events that need to be spatialized, the FMOD project should be built into the sound banks that the Unity project subsequently references. This in turn indicates the similarities and differences between both spatial audio implementations from the perspective of FMOD Audio Event authoring. A critical way in which the sound engineer can differentiate between events containing the same source audio track, but using different effects plugins, is to simply apply a different naming convention, depending on which plugin is to be used. In this project, all *ImDSP* events were provided a default name, and the Oculus Spatializer events had an "O" appended to the FMOD audio event's name (Figure 103 indicates the naming convention that was applied).

6.3.2 Contrasting Approaches to Triggering Audio Events

Having discussed how FMOD audio events were authored for both the ImDSP plugin as well as the Oculus Spatializer plugin, this section will briefly describe the differences and similarities by which FMOD audio events were triggered from within Unity. As has been reinforced throughout this chapter, the similarity between both environments needed to be maintained to enable an environment whereby the only independent variable of the experiments was the audio spatialization and output system being employed by each of the different Unity scenes (*HeadphoneEnv* and *SpeakerEnv*). Consequently, a fundamental consistency between both environments was the triggering of FMOD audio events, and the control thereof. Thus, the Headphone Environment, like the Speaker Environment, triggered FMOD audio events from script, and did not utilize the FMOD Studio Event Emitter script to trigger the relevant audio events (see Listing A2) .

Both the *ImScript* and *CloseApp* scripts functionality was detailed in the preceding section. However, due to the different set of audio events that were authored using the Oculus Spatializer, a separate mechanism by which FMOD audio events were initiated and controlled needed to be developed. Figure 108 indicates the attachment of the *AudioController* script to the Headphone Environment's *Leap Rig* object. This was the custom script that was created to control the audio for the headphone environment. As can be seen in this figure, the *ImScript, SocketIOClient,* and *CloseApp* scripts were all deactivated for the *Leap Rig* Game Object in the Headphone Environment.

The *AudioController* script performs and combines the same functionality exhibited by the playback control scripts for the Speaker Environment. Accordingly, the script initializes eight FMOD audio events (the ones to which Oculus Spatializer plugins are attached, i.e., the set of [planet]O audio events seen in Figure 103). However, as the Oculus Spatializer plugin has already been attached to these events from the Studio Application, the process of binding the plugin to each event was not necessary (unlike the *ImDSP* process of binding plugins to channel groups). Once the audio events have been initialized, each of the events is triggered when the application launches, once again adhering to the behavior of the *ImScript*. The playback control mechanisms that were implemented in the *CloseApp* script were replicated in the *AudioController* script. A more precise discussion on these playback controls is provided in chapter seven.

6.4 Capability Workflow

The previous sections have described the various processes involved in the creation of the "Immersive Planets" VR application. This section will provide an overview of the general workflow that was required to develop immersive VR applications that use these processes. The details of correctly configuring ImmerGo and the ASIO Playback server are omitted in this discussion (a review of this process is provided in the system manual which may be viewed in Appendix D). Aspects of this workflow that could be enhanced are discussed in the final chapter of this thesis. The steps within this workflow are indicated diagrammatically in Figure 112 below.

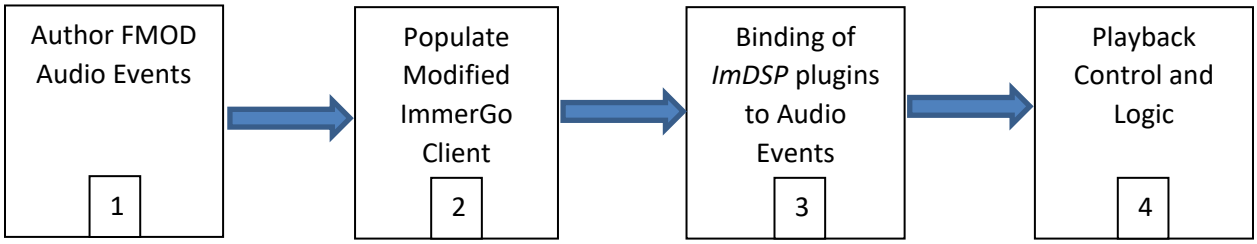

Figure 112: A schematic diagram indicating the steps involved in utilizing the Immersive Audio capability in a Unity project integrated with the FMOD audio middleware solution.

Figure 112 indicates that there are four primary steps that a developer needs to adhere to when utilizing the immersive audio capability. The first step requires the developer to author FMOD audio events in the respective FMOD project that accompanies their Unity application. In accordance with what has been mentioned in the previous sections, there are two requirements that the developer needs to fulfil here. Firstly, they need to ensure the audio tracks they are using are monophonic signals[73] and, secondly, that each audio event has the native FMOD Spatializer (Figure 111) attached on its respective DSP chain. Indeed, if they create a 3D audio event, then this spatializer will be attached to the event by default. An example of a monophonic and stereophonic version of a track is indicated in Figure 25. The difference between the two tracks is easily identifiable, as the first track has one waveform, whilst the second has two (one for each of the left and right channels). The figure also indicates the looping region

[73] If the reader reviews the process callback in the *ImDSP* implementation attached in the code-listing of this thesis, they will recognize that there is a solution to using mult-channel encoded audio signals should a developer forget to provide monophonic audio tracks in their project. In this instance, the algorithm will downmix the signal into a monophonic equivalent (summing each channel signal and dividing this sum by the channel number), thereby resulting in an inexact downmix of the multi-channel encodings.

of the FMOD audio track, i.e., is the region of a track that the developer wishes to loop. If this region is initialized, then the audio event to which this track pertains will loop continuously until the event is stopped by some or other action. In the case of Immersive Planets, this stop action is the closing of the application.

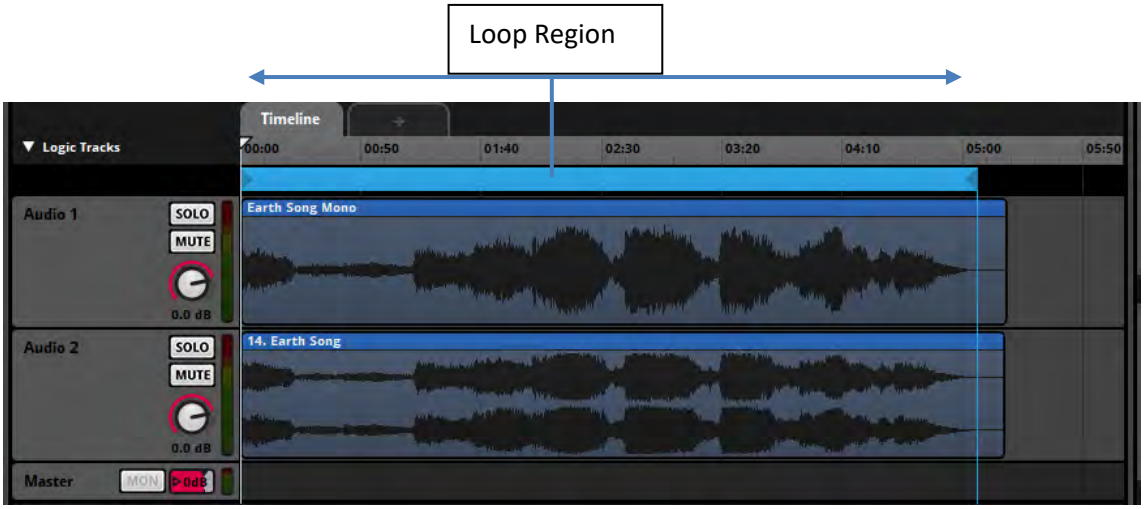

Figure 113: A screenshot indicating two versions of the same audio track. The first track is a monophonic version of the second track. As can be seen, the first track exhibits a single waveform, while the second track exhibits two waveforms: one for each channel embedded in the signal.

Having successfully authored compatible FMOD audio events, the second step in the workflow requires the population of the modified ImmerGo client Game Object array with the game objects to which FMOD audio events are attached (see Figure 105). The screenshot on the left of Figure 105 indicates this array being populated with the planet objects for the Immersive Planets VR application. Developers would populate this array with the objects in their scene that require spatial audio. These are the objects whose coordinates are transmitted to the ImmerGo server.

The third step in the capability workflow refers to the binding of *ImDSP* plugins to the FMOD audio events that are initialized. In the current iteration of this capability, this step requires the users to manually perform this function. In future, more robust iterations of the immersive audio capability, the *ImDSP* instance and FMOD event identifier will most likely be set from within the FMOD Studio Application. However, as this is not currently the case, the developer would need to bind *ImDSP* instances to FMOD audio events from within a custom script, which is still a simple process.

The fourth and final step in the workflow of this capability is the implementation of audio playback control, and developers would need to provide their own logic for the triggering and control of audio events in a Unity scene according to their requirements. As has been mentioned, the linearity of the Immersive Planets VR application required the audio events to begin playback as soon as the application launched. However, in a non-linear application implementation using the capability, the developer would need to bind *ImDSP* instances to spatial audio events before playback. A discussion of how this is achieved is provided in the closing chapter of this thesis.

6.5 Unity Assets and Resources Used

Of course, the real usefulness of the Unity game engine lies in the ability to employ and utilize a host of different Unity assets depending on the application that is being developed. In the creation of the Immersive Planets VR application, the primary assets that were used were the FMOD Integration, Oculus SDK, and LM Orion SDK provided by each of the associated companies. These assets facilitated the integration of the audio middleware solution into the Unity project, the utilization of Oculus-VR-specific constructs in the Unity scene, and the ability to employ hand-tracking and gesture recognition as the input modality for project interaction.

Other important assets used in the development of Immersive Planets were firstly, the MilkyWay skybox,[74] a planet texturing module[75], and a model of planet Earth. The latter model has subsequently been deprecated from the asset store. Finally, it should once again be mentioned that the Kyle Dixon and Michael Stein SPHERES soundtrack proved to provide optimal soundscapes for the planetary objects in the immersive VR scene.

6.5.1 Application Enhancements

Considering the complexity of integrating the immersive audio capability into the numerous frameworks that were utilized in the project, there were many developmental challenges, and therefore a few changes to the capability would lead it to being more user friendly. Many of these modifications are indicated in the concluding chapter of this thesis.

To reiterate, the primary goal of the project was to analyze and determine the differences between immersive spatial audio implemented with a speaker- and headphone-based approach in the context of VR applications. Thus, the *Speaker* and *Headphone* environments that have been detailed in the previous sections provide the foundation for determining and realizing this primary aim. Moreover, the Immersive Planets application that was constructed realizes a secondary goal of the project, i.e., to develop a system that utilizes the immersive audio capability in the development of an immersive VR application. Consequently, the immersive VR experience that hwas implemented can be deemed a successful use-case for the capability, whilst also providing the experimental framework by which the primary goal of the project was achieved.

The project video that has been alluded to, alongside the relevant usability and audiovisual fidelity results indicated in the following chapter, should provide the reader with a comprehensive idea as to how successful Immersive Planets was in providing audiovisual immersion to its users. The video was recorded from the *HeadphoneEnv* scene from the Immersive Planets application, and was captured using the NVIDIA GeForce Experience software.[76] The overlay software enables the visual capture of what is being output to a user's display, together with the auditory output of the audio being streamed to their audio driver. As a result, the Immersive Planets recording provides a visual and auditory rendition of a particular playthrough of the scene. The audio component of this recording represents the headphone and HRTF-based Oculus Spatializer implementation of the Immersive Planets application, and it is therefore recommended that the viewer watches the recording with headphones. In the recording, the user initially encounters all the planets beginning their audio playback. Subsequently, the

[74] https://assetstore.unity.com/packages/2d/textures-materials/milky-way-skybox-94001
[75] https://assetstore.unity.com/packages/tools/planet-texture-generator-51995
[76] https://www.nvidia.com/en-us/geforce/geforce-experience/download/

planets' orbiting function is enabled, and both the Earth and Mars planets audio is solo-ed. The recording also encompasses the interaction mechanism being used (i.e., "Selecting") to pan both Earth and Mars. The raw video file was recorded at a resolution of 1080p and at 60 frames per second, and was subsequently processed to indicate when the audio tracks for both the Earth and Mars planets were solo-ed.

7 COMPARATIVE TESTS TO DETERMINE SYSTEM EFFICACY

"Scientific Inquiry starts with observation. The more one can see, the more one can investigate."

— Martin Chalfie

The previous chapter analyzed the decisions that were taken in the design and implementation of the VR application that was used for comparative testing. This chapter will describe the experimentation that was performed to answer the research questions of this thesis. Essentially, the experimentation sought to compare a headphone and speaker-based audio approach to rendering immersive sound for an immersive VR experience. The results represent a multimodal analysis of the spatial accuracy of the two systems (whose implementation was detailed in the previous chapter). This analysis represents the quantitative findings of the research.

During the immersive VR experience that was used to provide the testing environment for the multimodal spatial analysis, qualitative data was gathered that helped determine the:

- The overall usability and visual fidelity of the system.
- Participant preference of the immersive audio systems.
- The perceived quality of the audio for each immersive audio environment.
- General qualitative data for both immersive audio systems (i.e., issues with either, and their reasons why).

The metrics and deductions gained from both sets of data (quantitative and qualitative) are used to contrast the relative immersive qualities of the different environments.

This chapter is comprised of four sections. The first section seeks to justify the experimental decisions that were taken according to research whose aims and goals are comparable. In addition, relevant audio research guidelines and recommendations are briefly analyzed. This constitutes a theoretical foundation for the experimentation. The second section provides an analysis of the testing framework, and therefore introduces the hardware and software components of the experiments. The third section indicates the methodology and the experimental conditions that were adhered to. The fourth and final sections of this chapter present the findings of the experiments and provide simple conclusions that these results point to. The more rigorous analysis of whether the experiments were successful in their aims is conducted in the concluding chapter of this thesis. This includes some of the participants' suggestions and subjective feelings about the system (including both immersive audio environments).

7.1 THEORETICAL FOUNDATION OF METHODOLOGY

The results and conclusions of this research stem from both quantitative and qualitative methodology. Before discussing these methodologies, it is worth indicating that the immersive audio capability that was developed operated in real time, which can be defined as the ability of the capability to guarantee a response within a specified time constraint. It is argued that latencies between 20 and 30ms, which can be considered high latencies in real time audio, are acceptable for the requirements of interactive multimedia applications (Lago & Kon, 2004). It is also suggested that most real time audio applications operating over networks generally operate at latencies higher than 10ms. Accordingly, the goal of the developed capability and overall system is to operate in these bounds (with a latency of under 30ms).

The following paragraphs and diagrams will aim to prove that the capability did, in part, operate within the bounds of what is considered real time in the field of audio technology. The real-time audio latency of the project exists in two parts: the playback latency and the spatialization latency. With regard to the playback time, the FMOD mixer was set to update every 10.67ms, in accordance with the following calculation which uses the sample rate of the project audio (48kHz), as well as the buffer size that was selected (512). [77]

$$\frac{512xpb}{48000}\ x\ 1000 = \mathbf{10.67}\boldsymbol{ms} \tag{13}$$

where xpb is the samples per block (buffer size) and ms is milliseconds. If one assumes the transmission delay for the sending and receiving of packets to be less than **1ms** between *ImDSP* clients and the playback server, then the latency from when an FMOD audio track is started to its samples arriving to the playback server would be around **~11.67ms**. The actual latency of socket IPC mechanisms is estimated to be around 400µs for packets of size 32KB, therefore the 1ms estimate indicated above more than accommodates this determined latency (Venkataraman & Jagadeesha, 2015). Of course, due to the ASIO driver requiring samples for playback before the server has received any, the padding of the circular buffers adds the largest delay before the initial playback time, which is approximately **21.3ms** when using 2 blocks of padding (see chapter five). As such, the initial playback delay for an FMOD event being launched in Unity and until its samples are provided to the ASIO buffer for streaming over the AVB network (which deterministically guarantees this transmission to occur in under 2ms) is approximately **~32.9ms**. This latency only pertains to the starting of an FMOD audio event from within Unity. After the padding component of the ASIO server is applied, no more padding/buffering requirements are needed throughout the FMOD audio track's duration. Therefore, it can be assumed that the playback server subsequently operates at the latency of the FMOD mixer and playback server and does not accrue any further latency. However, the **~32.9ms** initial latency will remain an overhead for the track's duration (see Figure 114 for an illustration of the latency components).

[77] https://documentation.help/FMOD-Studio-API/FMOD_System_SetDSPBufferSize.html

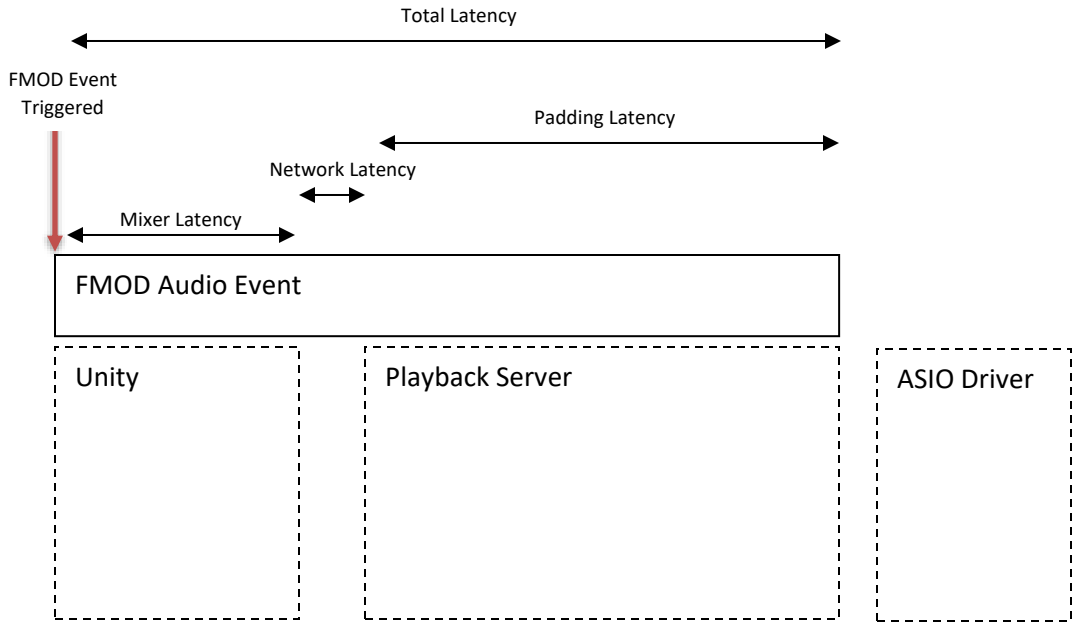

Figure 114: A diagram that indicates the components affecting the audio playback latency. The arrows indicating the latency are illustrative.

Since the ImmerGo system provides spatialization approximately every **8ms**, and the AVB network guarantees a **sub-2ms** latency, the spatialization latency can be assumed to be around **~10ms**.

Research suggests that the perceptual threshold of audio latency, which can be defined as the time taken before a user's performance or experience is impaired, exists between **20 – 70ms** depending on the physiological attributes of the user (Attig, et al., 2017). Therefore, both the playback latency and the spatialization latency represent a quantitative metric by which the developed system operated in real time.

7.1.1 Multimodal Importance in Determining Metrics

The majority of spatial audio and localization experimentation is based on audio listening tests, which are regarded as the most reliable means of determining audio quality (Zielinski, et al., 2008). While the term audio quality, in this context, often refers to bitrates and codec implementations, it is important to recognize that localization accuracy affects how someone perceives the quality of the audio. A distinction between the audio quality and localization metrics is provided consistently throughout the chapter. The primary attributes affecting the metric of audio quality in the experiments conducted for this project are indicated in section 7.1.2.1.

Traditional listening tests are generally mono modal, as they analyze the quality of the audio using only audio stimulus. Due to the questions and aims of this research, the quality of the audio was determined in the context of a VR experience, and therefore participants were experiencing multimodal stimulation, i.e., both visual and auditory stimulus. Research in this domain suggests that visual stimulus accompanying auditory stimulus influences the results of listening tests. In certain situations, visual stimulus has been shown to negatively influence the perceived quality of the audio when an experiment is conducted in which an audio-only format has been compared against an audio-visual format in which the audio remains consistent across both experiments (Suvorov, 2009).

The need to analyze audio quality and localization accuracy when using audio visual experimental frameworks is of great importance. We live in a visual culture, which has dramatically affected society in numerous ways, not least of all being the effect on how we socialize (Yılmaz, et al., 2019). The term visual culture refers to how technology has accelerated people's relations and interactions with generated visuals such as photographs, digital games, and now virtual reality. Accordingly, the audio component of the content that we consume is an important factor in influencing how a consumer reacts to the content. It follows that earlier research suggesting that visual stimulus negatively affects perceived audio quality is something that should be considered. However, there are suggestions that the intrinsic relationship between audio and visual stimuli denotes that audiovisual listening tests better represent real-life contexts (Wagner, 2010), which supports the argument that audio listening tests should also be performed in multimodal settings. Chapter four provided an important foundation as to why the audio component of a digital game (or VR experience) is of importance if the media wishes to positively influence the information, the entertainment, and the immersion of the user.

While both audio and visual quality and accuracy need to be investigated in monomodal experimentation, multimodal analysis of these metrics is useful in indicating the compatibility of integrated audiovisual systems. The audiovisual coherence (both spatial and temporal) is important if a digital environment is to be perceived as accurate. Certain research has sought to determine the perceptual thresholds of this coherence, i.e., what offset between both audio and visual stimulation is acceptable for spatial coherence to remain intact. The findings of this research suggest that the offsets of this alignment between stimuli are also influenced by features of the audio such as its harmonic or nonharmonic properties (Stenzel & Jackson, 2018). Indeed, in the context of VR, greater facilitation of immersion relies on multimodal stimulation, and therefore an analysis of different audio implementations in a multimodal simulation is useful as a framework for future research.

7.1.2 Qualitative Research and Listening Test Standards

The literature on listening tests that are used to assess audio quality and localization accuracy suggests that most research in this field implements subjective audio listening experiments. While there is no single standard or specification according to which subjective audio listening tests should be conducted, there are, however, 'recommendations', which can be defined as guidelines which experimentation should adhere to, in order to more accurately determine key audio quality metrics (Narbutt, et al., 2018). Due to the popularity and relatively fluid listening test methodology provided by MUSHRA (MUltiple Stimuli with Hidden Reference and Anchor), the methodology of the experiments conducted in this research is modelled on this experimental framework. The recommendations of this methodology are provided by the International Telecommunication Union (ITU). [78]

While the MUSHRA methodology was useful in addressing the primary and secondary goals of this research, the experimentation focused on a comparison between spatial audio systems, and their relative influence on immersion in a VR context. Accordingly, certain aspects of the methodology used in this research were unable to strictly conform to MUSHRA strategies, whose primary focus is to evaluate audio quality, rather than immersive qualities of audio systems. In other words, the recommendations of the testing methodology generally focus on monomodal subjective audio listening tests, and not on new multimodal technological frameworks and their requirements. However, there have been recent

[78] https://www.itu.int/en/Pages/default.aspx

discussions on how to better define objective metrics by which spatial audio quality can be evaluated (Narbutt, et al., 2018).

Another constraint of the methodology used in this research was a limitation regarding the selection of appropriate test candidates to perform the subjective listening tests. While this is discussed at greater lengths in the following section, it should be noted that the ITU ITU-R BS.1116-3 recommendation was used to aid the experimental design with regard to the provision of consistency in testing environments and with reducing external factors influencing the results of the experiment. In addition, aspects of the more contemporary ITU-T P.1310 and ITU-R BS.1284-2 recommendation for spatial audio meetings and general methods for subjective assessment were used as a reference. The latter two recommendations positively influenced what experimental conditions were decided on, as well as what attributes pertained to the audio quality of the two immersive audio systems being tested.

7.1.2.1 Brief Analysis of Core Aspects of These Recommendations

The ITU-R BS.1116-3 and ITU-T P.1310 recommendations provided a useful way of formulating statistically significant results for the subjective audio listening test. With regard to experimental design, this document suggests that subjective experiments are typically described by the control and manipulation of the experimental conditions, and, furthermore, by the accumulation of quantitative observational data from participants in the experiment (ITU, 2015). Of course, there is a suggestion that consistency in the design of a listening test requires that uncontrolled factors are mitigated to reduce "ambiguities". Apart from aiding the experimental design, this recommendation also indicated the relevant statistical tools and methods that should be applied to the type of data that was accumulated, in order to help determine statistical significance.

The ITU-R BS.1284-2 provided a formal understanding of some of the characteristics of multimodal audio listening tests, that is, listening tests that include both a visual and an auditory component. In essence, the recommendation points to how audio quality in a multimodal listening test can refer to numerous different attributes. However, in the context of this research, and the experiment that is detailed in this chapter, the **audio quality** can be said to have three critical characteristics.

I. The spatial coherence of both visual and auditory stimulus; "The correlation of source positions derived from the visual and audible cues".
II. How accurately the audio of an environment correlates to the distance of the visual objects from the observer/listener; "The correlation of spatial impressions between sound and picture".
III. The temporal coherence of both visual and auditory stimulus; "(The) time relationship between audio and video" (ITU, 2019).

7.2 TESTING FRAMEWORK

Before proceeding to a discussion of the experimental design and methodology, this section will reiterate important aspects of the framework that was used to create the environment by which the experiments could be conducted. The first part of the section briefly examines some of the core tenets of the VR application that has been developed. The second part then provides a brief overview of physical environment and the hardware that was used in the project.

7.2.1 The Virtual Reality Application

Chapter six introduced the design and implementation of the VR application that was used as the virtual environment in which the experiments were conducted and should be reviewed if the reader has any queries about its implementation. Of course, the sole purpose of this application was to enable an exact replication of a VR application for the two immersive audio systems that were being compared, thereby rendering a situation in which the only variable changing between the two immersive audio systems was the immersive audio system itself. How the consistency between the two environments was achieved is discussed in chapter six.

An important aspect of the VR application that was developed was its modelling of actual use-cases of VR digital game development, which provided the necessity for the game objects to move about and around the user—a common requirement of game objects in any virtual environment. This decision influenced how the subjective listening tests needed to be performed, as monomodal listening tests generally test the localization of audio sources that are statically placed around the listener.

The VR application was rendered using the Oculus Rift Development Kit 2 HMD and was compiled from within the Unity Editor for the Windows platform (the speaker environment could be launched from within the headphone environment, and vice versa). The computer that was used to host the application was a 64-bit Windows machine with requisite VR hardware to facilitate the processing requirements of the application. As such, a GTX 1060 Graphics card with six gigabytes(gb) of onboard VRAM was used. As mentioned in chapter six of the thesis, the Leap Motion controller was mounted atop the HMD to enable hand and gesture tracking for interaction with the game objects (planets) in the virtual scenes.

7.2.2 The Physical Environment

Participants were asked to inhabit a “user area”, which was positioned at the center of the hemispherical speaker array. Figure 115 indicates a diagram of the speaker array and where the user was positioned within this array (the blue sphere at the center). The array consisted of two layers, both an upper and lower layer or ring of speakers. The top layer comprised of seven speakers indicated by the purple speaker icons in the diagram, while the bottom layer comprised of eights speakers as shown by the blue speaker icons. Finally, a subwoofer was also part of the array and is represented by the large speaker icon in the diagram. Figure 116 shows a picture of the hemispherical array that was used for testing.

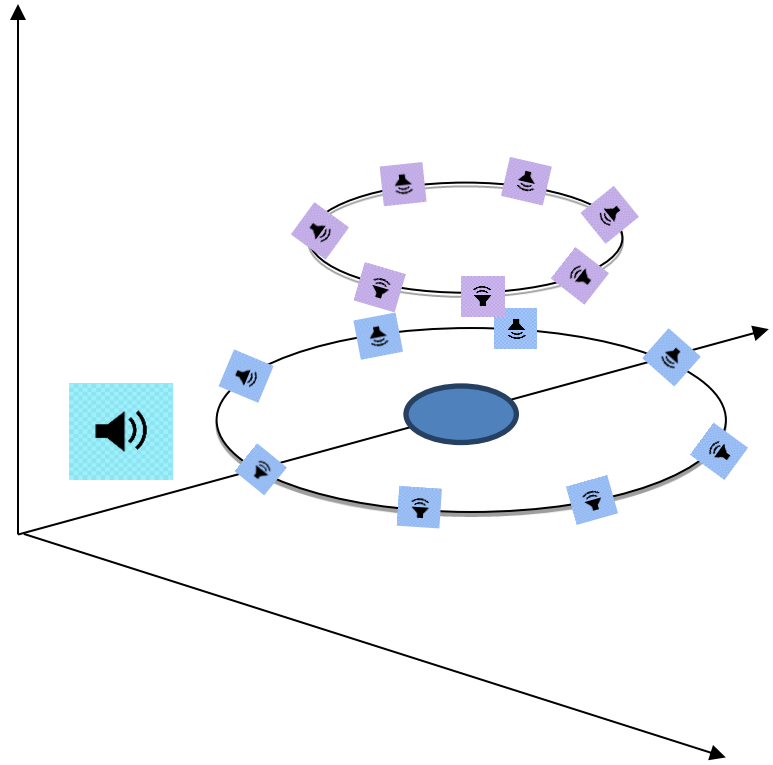

Figure 115: A diagram that indicates the speaker layout that was used by the ImmerGo spatialization system.

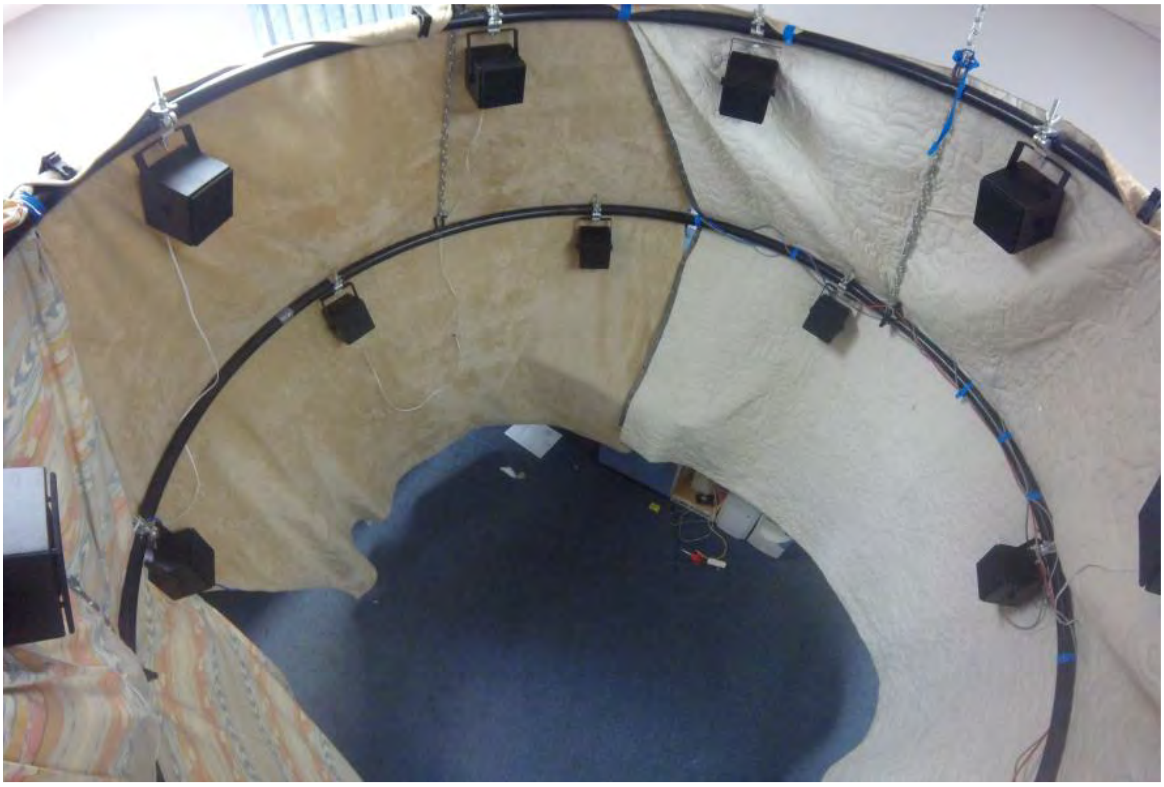

Figure 116: A picture of the speaker system, which indicates the two layers of speakers and the testing environment in which a user would be placed.

7.2.2.1 Speaker System Hardware

This section will briefly indicate the audio hardware that was utilized for the speaker environment component of this experimentation. Each of the hardware components is indicated in the following paragraphs, but the following schematic diagram shows how the hardware of the immersive speaker system interfaces with the software of the project.

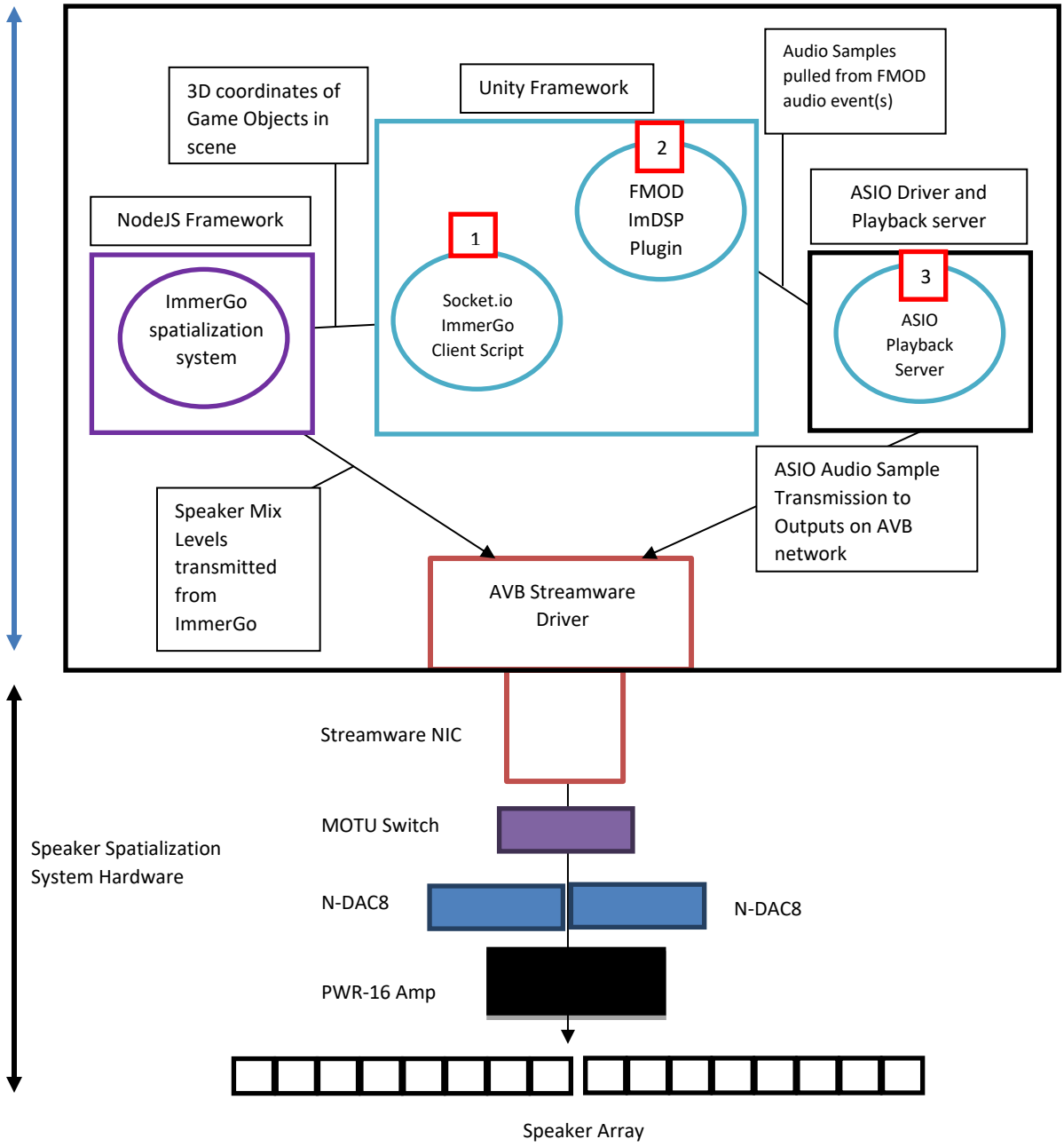

Figure 117: A schematic diagram that indicates the Software System and the Speaker System Hardware

The fifteen loudspeakers that were used in the array were SPK-4s[79] by MiniDSP. To enable ASIO and Ethernet AVB network communication, the Echo Streamware NIC-1[80] was installed within the computer hosting the VR application. The software driver and GUI to connect to the Ethernet AVB network for this Network Interface Card can be reviewed in the ImmerGo Unity Integration Capability Distribution Manual in Appendix D or by accessing the document through the link provided.[81]

Figure 118: A picture of the Echo Streamware NIC-1

Two MiniDSP NDAC-8s were used as the Ethernet AVB-capable Digital to Analogue Converters (DACs) for the system (see Figure 119). They in turn were connected to a MiniDSP PWR 16 amplifier and bridged across the AVB network using a MOTU AVB switch.[82] The switch was connected to the Streamware Card, which was then able to communicate the requisite speaker mix levels to the two NDAC-8s. Finally, a Yamaha NS-SW200 was used as the subwoofer to the system. [83]

[79] https://www.minidsp.com/products/usb-audio-interface/spk-4p-poe-avb-speaker-detail
[80] https://www.gearjunkies.com/2013/09/introducing-echo-audio-nic-1-pcie-network-adapter-and-streamware-sdk/#:~:text=The%20Streamware%20NIC%2D1%20is,networking%20applications%20for%20Microsoft%20Windows.
[81] https://drive.google.com/drive/folders/1_XlVvuaZ_gdzk5X9CsaiEy0lCgwXPQ_j?usp=sharing
[82] https://motu.com/en-us/products/avb/avb-switch/
[83] https://europe.yamaha.com/en/products/audio_visual/speaker_systems/ns-sw200/index.html

Figure 119: A picture of the NDAC-8 Digital to Analogue Converter

7.2.2.2 Headphone System Hardware

The headphone system was easily implemented in comparison to that of the speaker system. The output device for the headphone immersive audio implementation was the Sennheiser HD 450 headphones. Figure 120 provides a schematic overview of the headphone-based system. This diagram should be contrasted with Figure 117, which indicates a similar overview for the speaker-based system. As can be seen, the Oculus Spatializer is used to apply a set of HRTFs, which pertain to the position of the audio source in the virtual scene, to the source audio signal of the FMOD audio event. After which, the audio is provided to the audio driver (in this case WASAPI), and the driver feeds this audio into the output device (the headphones).

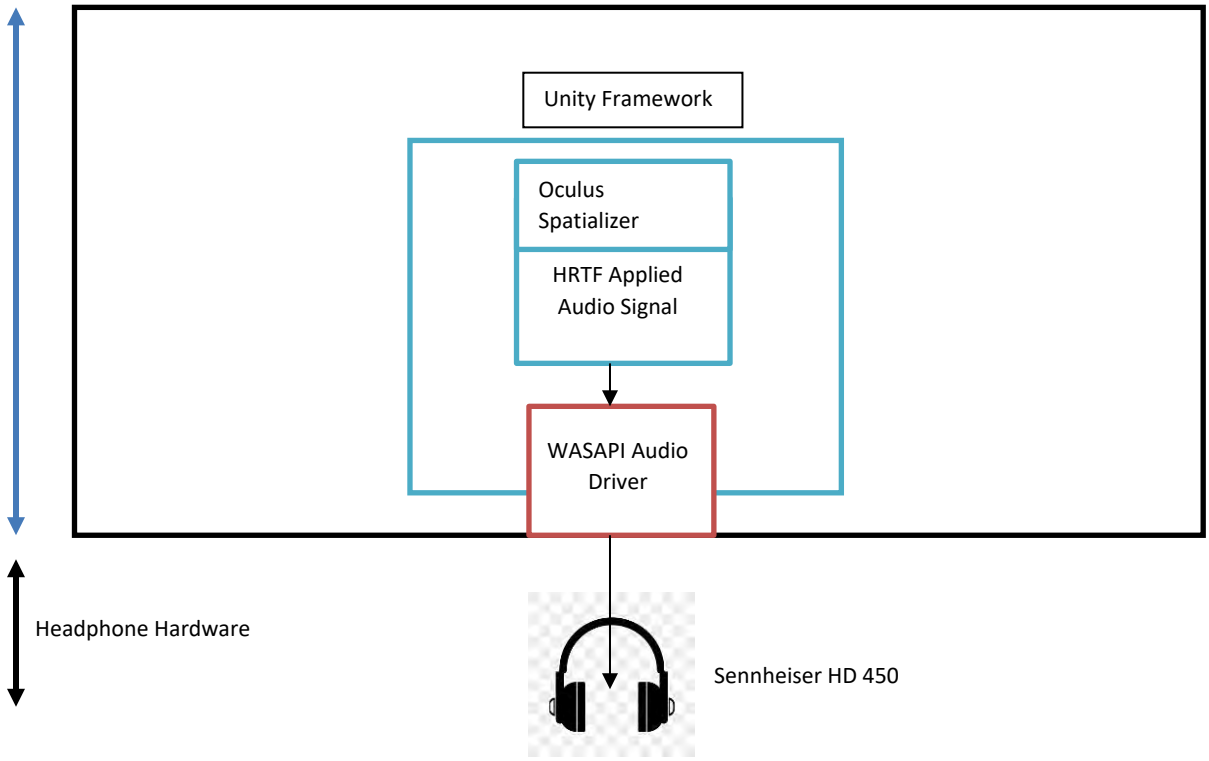

Figure 120: A schematic diagram of the headphone environment that indicates its software and hardware components.

7.3 METHODOLOGY AND EXPERIMENTAL CONDITIONS

This section will introduce the methodology that was used for the experiments that were conducted in this research. Before discussing the various aspects of the experiments, it is worthwhile to reiterate the primary goal of the project, that is, to analyze and determine the differences between immersive spatial audio implemented using speakers or headphones in the context of VR applications. A distinct approach to determining the immersive qualities of a VR system has still not been formalized, largely due to the variety of definitions that apply to what constitutes a VR system, and whether immersion can be reduced into a quantifiable variable of a system. However, the experimental methodology detailed in this section serves as a precursor to how the 'immersive' quality of a system might better be analyzed and determined in future research.

7.3.1 Experimental Design

The experiment comprised of two components which were duplicated for each of the immersive audio systems. The first component was a subjective audio listening test whose primary function was to determine the positional accuracy of the audio for each immersive audio system. This provided **quantitative** means to analyze each system. Once the listening test was completed, participants were told to freely pan audio around them in three dimensions, to provide an active experience whereby their interaction affected the environment around them.

As mentioned, this process was then repeated for either the speaker or headphone environment (depending on which environment the user had experienced first). After a participant had experienced both environments and immersive audio systems, and having performed the requisite tasks for both VR environments, they were provided with a questionnaire that they were required to complete. This served as a **qualitative** means to analyze the system, and the second component of the experiment. In subsequent experimental designs, the decision to analyze each system after a participant had completed the tasks, as opposed to completing both, would be looked into.

7.3.1.1 Metrics Used to Determine Differences Between Systems

The key metrics that were used to help determine the factors influencing the differences between immersive spatial audio for speakers and headphones whilst using the same virtual environment were the **positional accuracy** of each system and the **perceived audio quality** of each system (whose attributes and characteristics are detailed in section 7.1.2.1). The positional accuracy was a quantitative metric that was determined by asking users to correctly identify the audio sources in the virtual scene. The perceived audio quality of the each of the audio systems was a qualitative result that was provided by each of the participants upon completion of the experiment.

Additional metrics were used to validate the visual fidelity (quality) of the experience, and the usability of the VR application. Furthermore, subjective analysis of each immersive audio system was attained, so as to better recognize some of the core positives and negatives that participants experience for each system. The following diagram illustrates the metrics that were determined from the experiments, as well as their data classification.

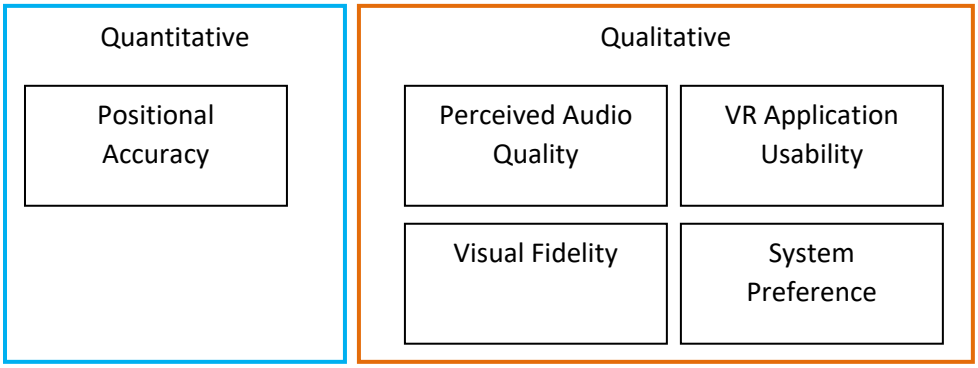

Figure 121: A diagram indicating the metrics that were determined from the experiment.

7.3.1.2 Design Consistency

To reliably determine the metrics detailed in the previous paragraph, thereby providing some insight into the differences between immersive spatial audio for VR when using headphones or speakers, the consistency across both virtual environments needed to be maintained. The only variable in the system that should change, across both virtual environments is the spatialization system. As was demonstrated and detailed in the previous chapter, the virtual environments for both the speaker and headphone-based immersive audio systems were identical, as the speaker environment was duplicated and then edited according to the requirements of the headphone spatialization system. As such, the size, orbital paths, and speeds of each of the game objects (planets) also remained consistent across environments.

The audio tracks that were used as FMOD assets for the FMOD audio events were consistent across both sets of FMOD audio events. The audio tracks were single channel (mono) .wav audio tracks. As the same audio codec was used for both sets of FMOD events, this therefore should have no influence on the audio quality metric. The only difference between the two sets of FMOD events was that the headphone events required the Oculus Spatializer to be attached to the DSP effect bus of the event, while the speaker events had the default FMOD spatializer attached. In addition, as the FMOD project was the same project for both environments, all the project's settings (such as the sample rate), were consistent.

7.3.2 The Experiment

As was alluded to earlier, the experiment consisted of two components. The first of these was a subjective listening test, which informed the quantitative component of the experiment, and served to primarily provide results on the positional accuracy of each of the immersive audio systems. The second component of the experiment was the questionnaire that was answered by participants once they had interacted with the virtual environments and completed both listening tests.

The following two sections provide background for the processes involved in collecting both the quantitative and qualitative data. However, before discussing these methodologies further, it is worthwhile to briefly indicate that all participants were in full knowledge of the aims of the research, the risks involved with experiencing VR (simulator sickness), and the tasks they had to complete within the virtual environments. These two components of the experiment were communicated to the experiment participants through a task sheet, with which they were provided remotely before testing. Information about the experiment and the research that was being conducted was also indicated to users before arrival using a system manual (see Appendix D). This manual also offers a brief overview of the requirements needed to successfully initialize the ImmerGo Spatialization system. Upon arrival, the

moderator of the tests reiterated all the information to ensure participants were fully aware of both the risks and the requirements of the tests, which constituted a briefing session. Once the tests, and questionnaire had been completed, the moderator then debriefed the participant by answering any questions they might have had.

In addition, as per the ethics committee regulations of Rhodes University,[84] all the participants signed an informed consent document, indicating they were in full knowledge of what they were required to do, as well the risks involved with VR. In addition, the moderator cleaned the headset between experiments to ensure that a hygienic environment was provided to all participants.

7.3.2.1 Listening Tests

The platform for the listening tests was relatively simple to implement. The *AudioController* and *CloseApp* script were modified to facilitate playback and application control via input from keystrokes provided by the tester/moderator of the listening tests while participants were immersed in VR (see Listing A3. The input from the moderator enabled the soloing of planets (virtual audio sources) in the scene, whilst also controlling the orbiting capability of the planets, and the launching of the ASIO playback server (for the speaker environment). Finally, either VR environment could also be launched by the moderator from within the other environment. The exact mechanisms of this functionality can be reviewed in the code-listing accompanying this thesis.

After a participant had been briefed about the system and the tasks they were needed to perform, they were led into the center of the hemispherical array, which represents the 'user area' or listening area in which the participants were encouraged to remain throughout the process (see Figure 115 and Figure 122 for a better demonstration of where the user area was located amidst the speaker array).

[84] https://www.ru.ac.za/researchgateway/ethics/

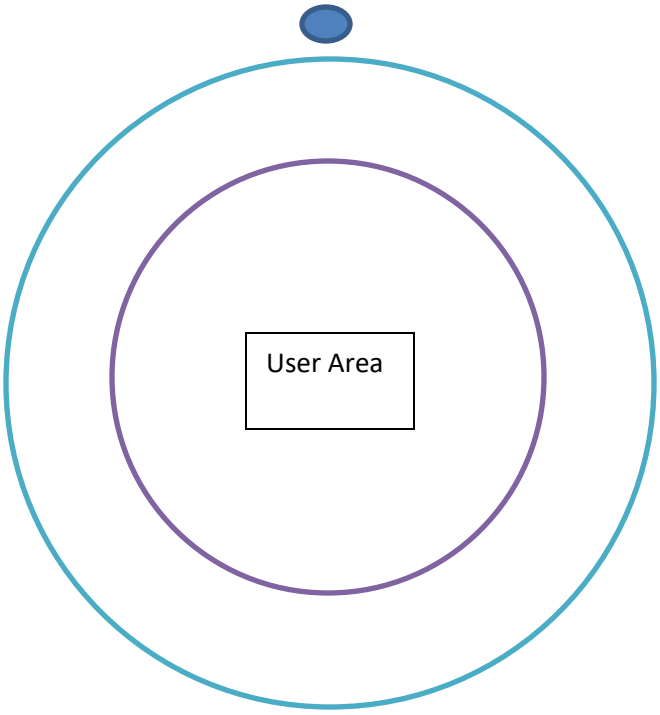

Figure 122: This figure provides a top-down representation of the two speaker layers in the speaker array and indicates where a user was located relative to these layers. In addition, the blue dot at the top of the diagram designates the position of the positional tracker for the HMD.

Once a participant was correctly positioned amidst the speaker array, they were provided the HMD, which was appropriately adjusted to their head until they verbally indicated they could clearly see the objects present within the Oculus Home application.

At this point, the moderator had completed all pre-experimental checks, and was ready to perform the listening tests and launch the compiled and distributable Immersive Planets application (.exe).

7.3.2.1.1 The Listening Test Process

Participant were given approximately thirty seconds to look around and acclimate to the virtual world of the Immersive Planets VR application. During this acclimation period, the audio sources from each of the planets would begin audio playback. The moderator could see exactly what the participants were looking at, as the display of the HMD was replicated on the 2D desktop monitor of the computer hosting the application.

Once the acclimation period was completed, the moderator would mute all audio sources and initialize the function of planet orbiting. The moderator would then verbally indicate to the participant that the audio listening test was about to begin, and the first audio source would be soloed. As per the ITU recommendations, participants were provided approximately twenty seconds to correctly identify which planet (audio source) was emitting audio. Failure to identify the planet in this time frame would constitute an incorrect identification of the planet and audio source.

To correctly identify planets and audio sources, the user was told to verbally indicate which audio source was active, whilst also pointing at the correct source using the three-dimensional cursor. Due to the color-coding of the planets, this proved to be a robust means by which a participant was able to indicate

which planet (audio source) they thought was active. Every attempt to identify a planet corresponded with the moderator determining whether this attempt was correct or incorrect, and inputting this data into an appropriate spreadsheet. After all audio sources and planets had been tested, the user was notified that the listening test had been completed, and that they were free to interact with the environment using the three-dimensional cursor.

Once they had finished interacting with the environment, they were notified that the next immersive audio system and listening test would begin after they had acclimated to the new environment, the next listening test would begin. Accordingly, the process was replicated for the second environment which, as indicated in section 7.3.2.4, was randomly determined for each experiment (i.e., whether the speaker or headphone environment was analyzed first).

7.3.2.1.2 Interacting with the Environment After Listening Test Completion

The moderator would ask the user which audio source they wished to pan, and would subsequently activate this source, so that the participant could pan the planet, and therefore audio source, in three-dimensions. This therefore provided the participant with an active means to interact with the environment, and the ability to pan and position the planets (audio sources) around them both horizontally and vertically (in three dimensions).

7.3.2.2 The Questionnaire

To attain the qualitative results of the experiment from the participants, a questionnaire was formulated that posed carefully constructed questions aiming to provide the qualitative metrics that were introduced earlier. The full version of this document, and all the questions that were posed to participants, is available in Appendix D of this thesis: for the sake of brevity, the questions themselves are excluded from this text. However, the core elements and design decisions of the questionnaire are indicated in the following paragraphs.

7.3.2.2.1 Questionnaire Aims

The format of the questions ensured users provided suitable answers that, upon collation, were able to determine whether the primary and secondary aims of the research were successful. The following list indicates which questions were used to determine the metrics introduced earlier in this chapter.

- Question **One** asked users to provide a numerical value that represented the usability of the system (by providing a value between 0 and 10). This pertained to how easily planets were panned around the virtual environment, and therefore provided an indication of how successful the intuitive form of interaction was.

- The **second** question helped categorize the listeners/participants into two groups: experienced and novel users.

- The **third** question aimed to address whether the VR project had visuals that were of a high enough fidelity to immerse participants in the planetary simulation.

- Question **four** sought to evaluate the pre-experiment conditions, that is, to determine whether users were adequately informed about the experiment and the tasks they needed to perform within the VR application.

- Question **five** was used to determine numerical values that represented the perceived audio quality of each of the immersive audio systems.

- Question **six** sought to determine which immersive audio system was preferred by the participant, whereas Question **seven** enquired about the advantages that system had over the other.

- Question **eight** asked users what they believed to be the biggest issues with each of the immersive audio systems and environments.

- The final question aimed to determine how useful and prevalent participants believe VR technology will become.

The format of the questionnaire required numerical responses with regards to perceived audio quality, as well as system usability (questions one and five). Questions two, three, four, and nine all required a (yes/no) values, although additional comments were welcomed. Question six required users to select which immersive audio system they preferred (speaker or headphone). Finally, questions seven, eight, and nine were all open-ended and therefore required participant comments.

7.3.2.3 Listener Panel Selection

The ITU-R 1116-3 recommendation suggests that a total of twenty subjects or participants in an experiment will enable sufficient conclusions to be drawn from the results of the tests (ITU, 2015). Accordingly, twenty-two participants performed the experiments detailed above, rendering a sample size which slightly exceeded that recommended.

The recommendation also indicates that, in ideal testing conditions, expert listeners should be given preference over novel or inexperienced listeners. Due to constraints imposed on the research, it was difficult to find more experienced listeners than novel ones. However, most of the participants who were selected were either computer science students with some experience in audio programming, musicians, sound technicians, or digital game enthusiasts, all of whom are accustomed to spatial audio being rendered to headphones.

Another issue with regards to participant selection, was that, at the time and in the location that the research and the experiments were conducted, exposure to Virtual Reality applications was consistently something that participants had not experienced. This problem was overcome by providing information to participants about VR technology before they performed the experiments. In addition, the acclimation period indicated in section 7.3.2.1.1 also served to introduce participants to VR before undertaking the experience. The results of question **two** indicate that nine of the twenty-two participants had some experience in VR or in the realm of audio listening. A discussion of these issues, and how reliable the question was at determining whether a participant was experienced or not is presented in the concluding chapter of this thesis.

7.3.2.4 Experimental Considerations

This brief section will indicate some of the measures that were taken to mitigate against biases that might have influenced the results. The most important consideration was the randomization of certain processes. As a result, the order in which a participant experienced an environment was random, and roughly fifty percent of the participants performed the tests in the speaker environment before the

headphone environment, and vice versa. In addition, the order in which planets were identified by the participant was also randomized by the moderator.

In terms of the application development, the velocity and orbital paths of each of the planets were also unique. However, the vertical tilt (or inclination) of the orbital path remained consistent across all eight planets. In addition, the size of the planets also differed. However, the relative spread of each of the planets was not considered in the implementation for both speaker and headphone environments.

Of course, as was mentioned in section 7.3.1.2, the most important element of the experiment was to denote the audio spatialization system as the independent variable of the experiment; the only variable that changed across both environments. As will be argued in the concluding chapter, this then suggests that the dependent variable (being the perceived immersion of a participant) was influenced by the audio spatialization system.

Another difference between traditional subjective audio listening tests, and the tests conducted in this experiment, is that most listening tests determine the localization accuracy of audio sources that are static around the listener. Due to the requirement of moving, and pannable audio sources in this research, to best model actual use-cases of spatial audio in digital game and VR development, the position of the audio sources was dynamic. This therefore made it difficult to determine how the speaker system performed at specific orientations about the speaker array (i.e., back/front/left/right). However, the distance of the source to the listener, as well as the height of the source, were variables that could be analyzed.

7.4 RESULTS

This section is comprised of two parts. The first part presents the analysis of the data that applies to the subjective listening tests, and therefore provides the quantitative findings of the experiments. Of course, graphical representations of this data and relevant statistical analyses seek to provide clarity on the results that were recorded for each immersive audio environment, and the **positional accuracy** that each achieved, as well as any relation between these metrics. The second part of the section analyzes and collates the findings of the questionnaire, and therefore provides answers as to how successful the experiment was at determining the qualitative metrics indicated in section 7.3.1.1. The relative success and shortcomings of the experiment are presented in the concluding chapter of this thesis, together with a discussion of how successfully the primary and secondary aims of the research were achieved.

7.4.1 Listening Test Results

The following two sections indicate the results of the subjective listening tests for both immersive audio systems. Both sections will follow a similar template, whereby the tabulated results are presented before graphical and statistical analyses of each are indicated. The third section seeks to provide various indications of how robust and statistically significant the results are. This therefore encompasses an analysis of each of the systems, as well as a combined analysis of both.

7.4.1.1 The Speaker-Based Results

The source data that informs the following tables and graphs is provided as a spreadsheet that is available from the Project Resources folder of this thesis. Therefore, the information presented below represents a curated form of the data that was accumulated in the listening tests conducted by the twenty-two participants.

Table 6 indicates how many planets (audio sources) were correctly determined by the participants. The table therefore represents a frequency distribution of the data that was accumulated. As can be seen, only one participant identified five of the eight planets, while most participants (14) were able to correctly identify seven of the eight planets. Three participants were able to identify all eight planets.

Correct Identification	0	1	2	3	4	5	6	7	8	Total:
Frequency	0	0	0	0	0	1	4	14	3	22

Table 6: Frequency table indicating the category of correct identifications (number of audio sources correctly identified), and the frequency of participants falling in this category.

If this data is then rendered into a frequency distribution graph, the resulting figure indicates that the data conforms to a pattern of normal distribution (see Figure 123).

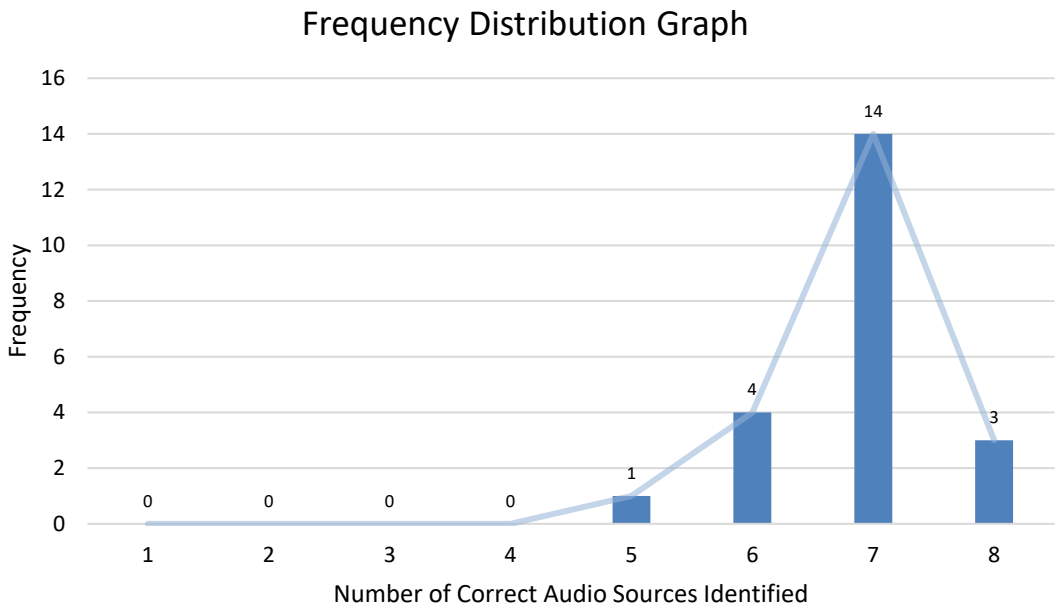

Figure 123: The frequency distribution graph when using the data from Table 1.

As there were twenty-two participants, and there were eight planets (audio sources) to be identified, there were a total of 176 audio source classifications. Of these, 151 were correctly identified, whilst 25 classifications were incorrect, meaning that the percentage average accuracy across all the listening tests for the speaker-based system was **85.80%.** This can also be expressed by saying that, on average, a participant was able to correctly identify 6.86 of the eight planets.

It is worthwhile to discuss the variance, and standard deviation of the data, to provide an idea of how dispersed the data is across the set of values. However, as is demonstrated in the following section, one of the participants scored the lowest value of correct predictions for both the speaker and headphone systems (see participant 13 in both positional accuracy source data tables). As a result, there are two sets of calculations for variance and standard deviation: one set in which these results are included, and

another where they are excluded on the basis of them being an outlier to both sets of data. Of course, this also influences the sample size (reducing it by one) for the excluded calculations.

The standard deviation of the data set is calculated according to the following formula:

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

(14)

where σ is equal to the standard deviation of the sample set, x_i is equal to each value of the sample set, μ is equal to the average of the sample set, and N is the size of the population. Table 7 indicates the standard deviation and variance for both the inclusive and exclusive data sets.

Data Sets	Standard Deviation (σ)	Variance
Inclusive	0.71	0.50
Exclusive	0.59	0.35

Table 7: Standard Deviation and Variance for both the Inclusive and Exclusive data sets for the speaker environment.

In both instances (exclusive and inclusive), the standard deviation is relatively low. This then indicates that most of the data falls close to the average, which, if one reviews Figure 123, is apparent, as most participants correctly identified seven audio sources (planets). Indeed, a low standard deviation indicates that the distribution of data in the sample set is close to the average of the sample set, which is the case in this instance. A coefficient of variation is calculated according to the following formula:

$$CV = \frac{\sigma}{\mu}$$

(15)

where CV is the coefficient of variation, σ is the standard deviation, and μ is the mean (average) of the sample set. The CV is **0.10** (inclusive) and **0.08** (exclusive). These low values indicate that the degree of variation in the data is minor.

With regards to which planets were more accurately identified, it was found that certain planets scored higher than others:

Planets	Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus	Neptune
Accuracy (%)	63.6%	68.2%	72.7%	95.5%	95.5%	100.0%	90.9%	100.0%

Table 8 and Figure 124 indicate the identification accuracy of each of the planets (audio sources). If one were to group the first three planets into a close planets group, and the remaining five planets into a far planets group, the results then suggest that localization accuracy was more accurate for planets that were further away from the user. A discussion on the factors that might influence this hypothesis is provided in the final chapter of this thesis.

Planets	Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus	Neptune

Accuracy (%)	63.6%	68.2%	72.7%	95.5%	95.5%	100.0%	90.9%	100.0%

Table 8: A comparison of the accuracy at which the planets (audio sources) were identified. In this instance, the order of the planets from left to right corresponds to how close they are to the participant (Mercury being the closest, and Neptune being the furthest).

In the following figure, the two groups that were mentioned are indicated by a difference in color. The close planet group is indicated in orange, while the far planet group is shown in blue.

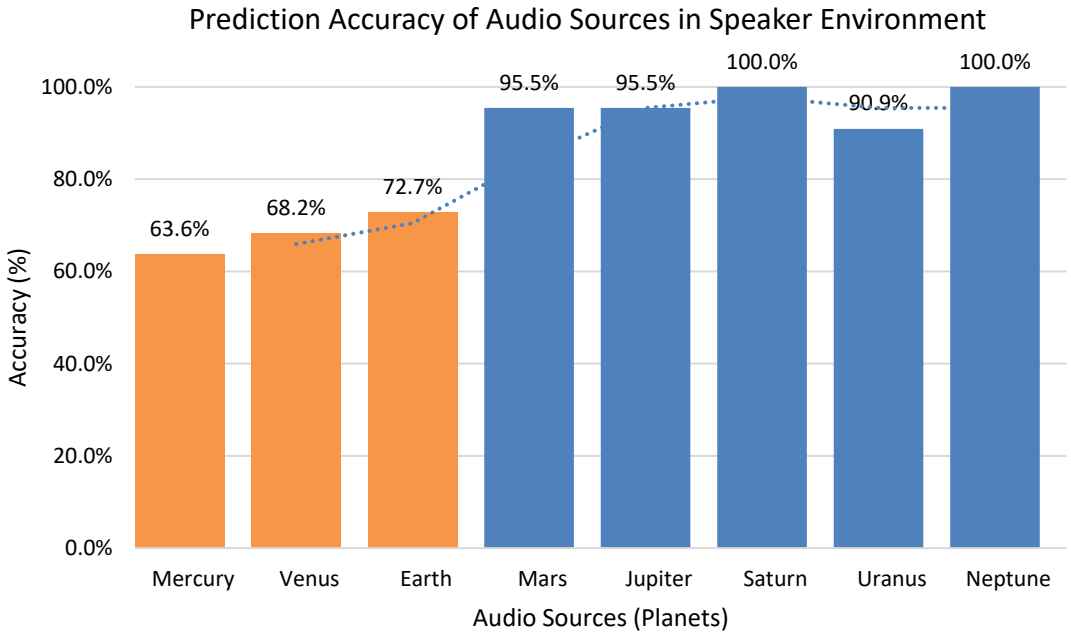

Figure 124: Prediction accuracy of each of the planets in the speaker environment.

7.4.1.2 The Headphone-Based Results

As with the Speaker-Based results, the source data for the following tables and graphs are provided in the Project Resources folder of this thesis. The following information remains consistent with the structure that the previous section provided, to better facilitate a comparison between the two sets of information.

Table 9 indicates the frequency distribution of the headphone-based immersive audio environment. A single participant could only correctly identify three of the eight audio sources (planets). Similarly, only one participant correctly identified six of the eight audio sources. Nine participants were able to correctly identify seven of the eight sources, and eleven participants were able to correctly identify all eight planets.

Correct Identification	0	1	2	3	4	5	6	7	8	Total:
Frequency	0	0	0	1	0	0	1	9	11	22

Table 9: Frequency table indicating the category of correct identifications (number of audio sources correctly identified), and the frequency of participants falling in this category.

As with the previous section, Figure 125 provides a graphical representation of the data presented in Table 9:

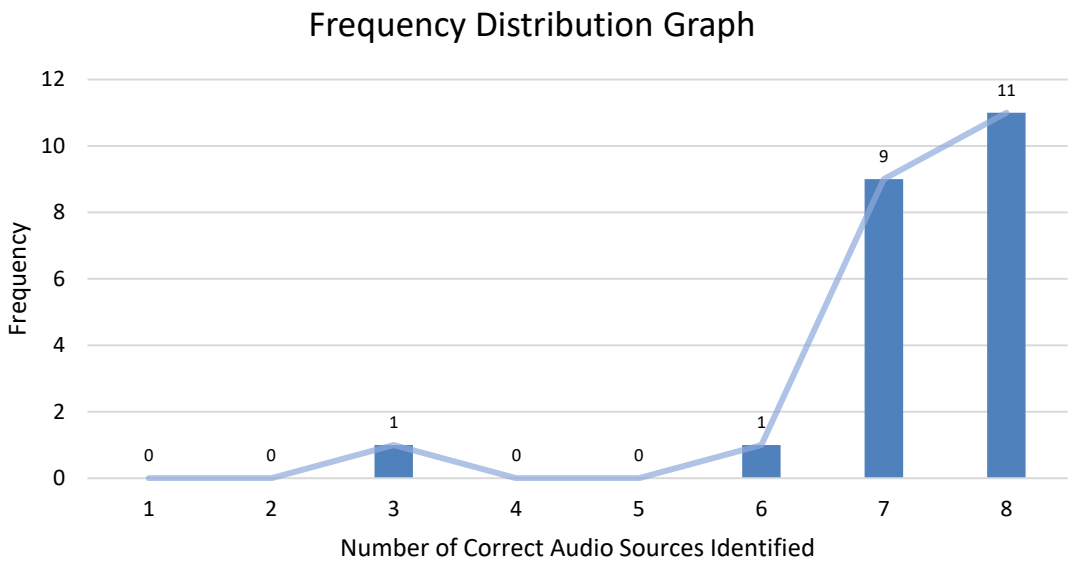

Figure 125: A frequency distribution graph for the headphone environment when using the data presented in table 4.

Of the 176 audio classification tests that were conducted for the headphone environment, 160 of these tests were correct identifications of the audio sources (planets). Sixteen (16) were incorrect classifications, which denotes that the percentage average accuracy achieved by the headphone system was **90.9%** or 7.27 correct audio source identifications per participant.

The standard deviation and variance of the results attained in the headphone environment are indicated in Table 10. As was the case with the speaker-based results, there is a variance and deviation value for a data set containing the outlying participant's results, and one that does not.

Data Sets	Standard Deviation (σ)	Variance
Inclusive	1.12	1.23
Exclusive	0.60	0.36

Table 10: Standard Deviation and Variance for both the Inclusive and Exclusive data sets for the headphone environment

Unlike the speaker environment, the variance and standard deviations of the headphone environment differ considerably when including and excluding the outlying data points. If one regards the inclusive data set, the standard deviation indicates that, although the average correct identification of audio sources in the headphone environment was 7.27, and therefore higher than that of the speaker environment, the distribution of data was less uniform. However, if one reviews the exclusive data set, the standard deviation is equal to that of the standard deviation recorded in the speaker environment, which suggests that the distribution was very uniform. The CV for the headphone data sets is calculated to be **0.15** (inclusive) and **0.08** (exclusive), which once again indicates a reasonably low degree of variation in the data. However, as has been indicated, the outlying data point does render a scenario whereby the degree of variation is significantly more pronounced than when excluding this data point. This once again makes sense when one considers Figure 125, which indicates that twenty participants

were able to identify either seven or eight audio sources (planets), only one participant could only identify six of the audio sources, and the outlier could only identify three audio sources.

Unlike the speaker environment, it should be noted that the relative prediction accuracy of the audio sources (planets) in the headphone environment did not exhibit the same trends of closer planets being less easily identifiable. However, an interesting observation of the data shows that three of the eight audio sources were identified the same number of times and, in fact, the only unique identification accuracy was the audio source from the planet Mercury, which was correctly identified by every participant.

Planets	Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus	Neptune
Accuracy (%)	100.0%	81.8%	81.8%	90.9%	95.5%	90.9%	95.5%	90.9%

Table 11: A comparison of the accuracy at which planets (audio sources) were identified. In this instance, the order of the planets from left to right corresponds to how close they are to the participant (Mercury being the closest, and Neptune being the furthest).

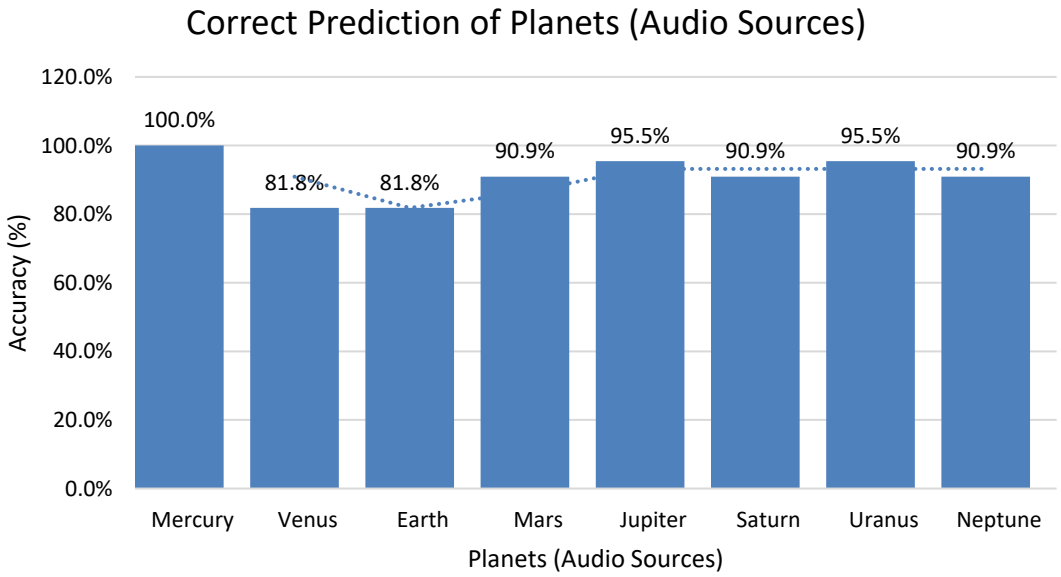

Figure 126: Prediction accuracy of each of the planets in the headphone environment.

7.4.1.3 Statistical Analysis of Both Systems

Before providing a rigorous statistical analysis of the two systems, the following figures assimilate the information that has been provided in the two previous section:

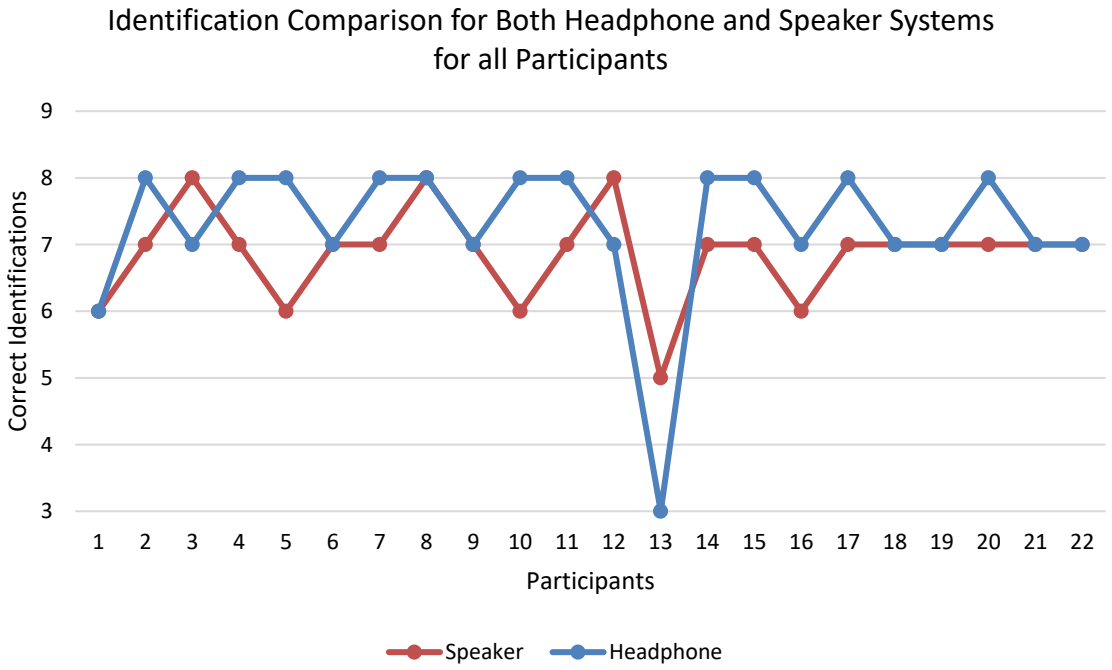

Figure 127: A comparison of identification metrics for each participant for both sets of listening tests.

Figure 127 indicates a direct comparison between all participants for both sets of listening tests. In the figure, the reader should note that, when using the speaker system, three participants were able to correctly identify more audio sources that when using the headphone system; six participants correctly identified the same number of audio sources when using both systems; and eleven participants correctly identified more audio sources when using the headphone environment. Another interesting aspect of this figure is that it indicates that the maximum difference between correct identifications across the two systems for a single participant was two audio sources.

Figure 128 provides a direct comparison of how accurately the specific audio sources (planets) were identified. The trendline for the speaker environment (dotted red line) indicates the apparent trend that was discussed earlier: that the close planets were incorrectly identified more often than the far planets.

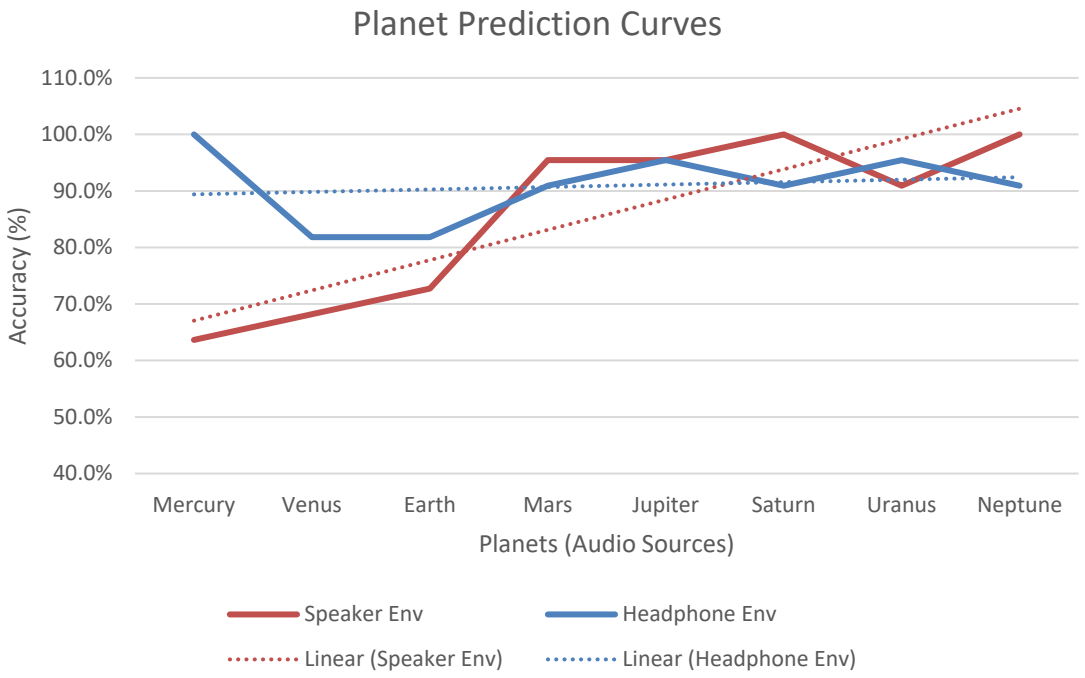

Figure 128: A comparison of the planet prediction curves of the two immersive audio systems. The trendlines for both data sets are also indicated in this graph.

Table 12 presents the differences between the average accuracies across both immersive audio systems. While it indicates a relatively small difference in accuracy across the two systems, it is important to provide a means to determine whether this is a statistically significant difference, or whether it is negligible. The following paragraphs discuss a statistical test that was used to achieve this answer.

Average (Mean) Accuracy of Headphone System	**Average (Mean) Accuracy of Speaker System**	**Difference Between Accuracies**
90.9%	85.8%	5.1%
7.27 correct identifications per participant	6.86 correct identifications per participant	0.41

Table 12: Average comparison across both audio environments (systems).

As the data indicated in the frequency distribution graphs of both the speaker and headphone environments is assumed to conform to a pattern of normal distribution, it is worthwhile to perform a *t*-test to determine whether there is a significant difference in the mean of the positional accuracy across both the speaker- and headphone-based immersive audio systems. The ITU recommendations suggest using an Analysis of Variance (ANOVA) test in determining the statistical significance of two groups (ITU, 2015). However, an ANOVA test is only useful when analyzing the difference in means across more than two groups. For this reason, the parametric statistical *t*-test was selected to determine the difference between means across the two data sets that were recorded.

More specifically, the *t*-test was used to indicate whether the variance in the positional accuracy of immersive audio systems differs according to their implementations (speaker-based or headphone-based). The null hypothesis (H_0) of this scenario is that the difference in variance between the two

systems is negligible (less than a threshold value of 0.05). The alternate hypothesis (H_a) is that the true difference between the systems is not negligible and is therefore greater than threshold value (0.05).

Due to the type of data and their distribution, a correlated or paired *t*-test was performed. This version of the test was selected as the same participants were used to provide input for both systems, and the same metric (positional accuracy) was being determined for each system. The following formula was used for this paired *t*-test calculation:

$$T = \frac{\mu_1 - \mu_2}{\frac{\sigma(diff)}{\sqrt{(n)}}} \quad (16)$$

where T is the *t*-test value, μ_1 and μ_2 are equal to the averages of the sample sets, $\sigma(diff)$ is the standard deviation of the differences of the paired data values, and n is the number of samples in the sets. Table 13 indicates the *t*-test results that were determined for the inclusive and exclusive data sets:

Data Sets	Determined *t*-test values
Inclusive	0.029
Exclusive	0.004

Table 13: paired T-values that were calculated for the inclusive and exclusive data sets of both the speaker and headphone listening tests.

If one reviews the threshold value that was provided (0.05) in H_0 and H_a, then it can be concluded that both data sets exhibit a *t*-value that is below this threshold, and so the H_0 stands, which is that the difference in variance of positional accuracy between the two systems is indeed negligible.

7.4.2 Qualitative Analysis of the Questionnaire

Having presented the quantitative results for the listening tests that were performed, the remainder of this chapter will focus on the other results that emerged from the questionnaire, and therefore constitute the qualitative results component of the experiment. This section will also therefore indicate how some of the crucial qualitative metrics that were introduced earlier were determined. Initially the **perceived audio quality** of the systems will be analysed alongside the metric of system preference, and subsequently the results used to determine system usability and visual fidelity will be detailed.

7.4.2.1 Perceived Audio Quality and System Preference

The metric of audio **perceived audio quality** was determined by asking participants to provide a numerical rating of what they believed the quality of audio was for either system. As was mentioned in section 7.1.2.1, the quality of audio pertained to the temporal and spatial coherence of the audiovisual stimuli, as well as the spatial impression of the audio relative to the visual stimulus (i.e., the correlation of the depth of the audio relative to the position of the planet in the virtual scene). Both Table 14 and Figure 129 present the averages of the metric for both audio environments.

Average Audio Quality Rating	Rating (0-10)	Percentage (%)
Speaker Environment	7.5	75%

Headphone Environment	8.9	89%

Table 14: The average perceived audio quality of both immersive audio environments

As can be seen, the headphone environment outperformed the speaker environment with regards to perceived audio by a margin of 14%. A few possible reasons for this difference in perceived audio quality are provided in the concluding chapter, but as it is evident that closer planets were less easily identified in the speaker environment, this would no doubt influence the spatial impression of the audio of these closer planets, and therefore the perceived audio quality of the system.

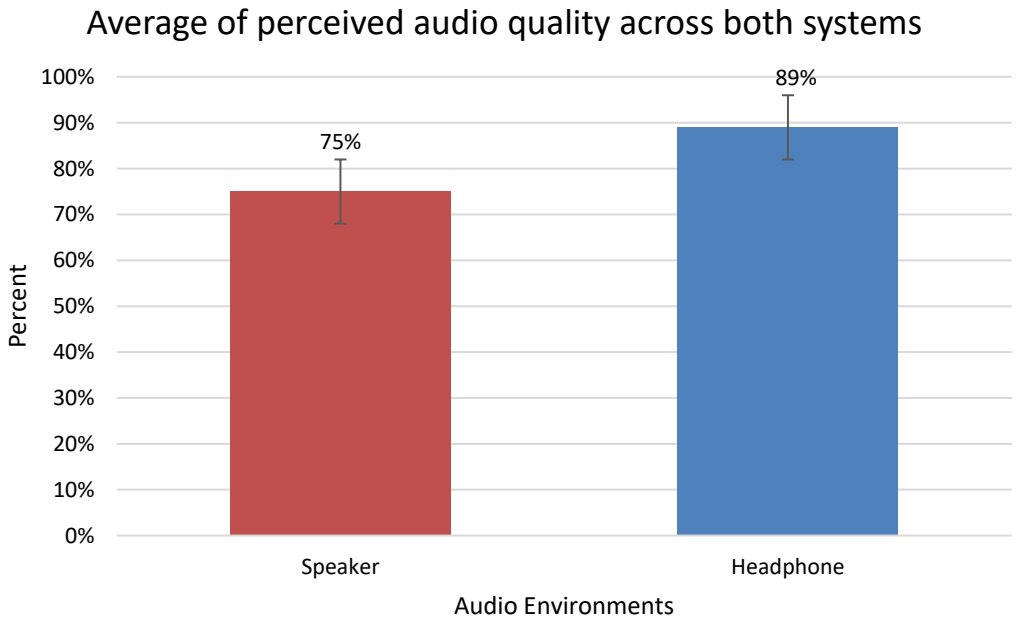

Figure 129: A graphical interpretation of the averages of perceived audio quality for both environments.

With regards to system preference, it was found that thirteen (13) participants preferred the speaker environment, even though the perceived audio quality was determined to be higher for the headphone-based immersive audio system. Some of the qualitative answers that users provided are presented in the final chapter, and these explain why participants indicated this preference even though the headphone environment seemingly performed better.

System Preference	Speaker	Headphone	Total
Frequency	13	9	22

Table 15: A frequency distribution table that indicates which system was preferred by participants.

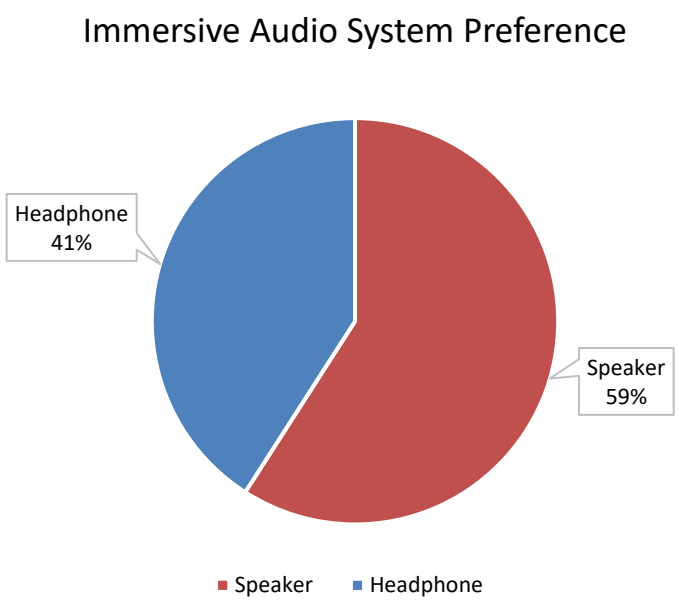

Figure 130: A Pie chart that indicates the difference in system preference for both audio systems.

7.4.3 System Usability and Visual Fidelity

The system was determined to be user-friendly, as it scored an average value of **8.3** from all the participants. The following table indicates the distribution of the scores:

Usability Score	0	1	2	3	4	5	6	7	8	9	10	Total:
Frequency	0	0	0	0	0	0	1	2	12	3	4	22

Table 16: Frequency table indicating usability scores.

Most users provided a comment suggesting that they would have found the system more usable had there been a demo explaining how to properly perform the required interaction of selecting and placing an audio source around them. There was also a suggestion that initially they struggled to perform the interaction as it was novel and not something they had tried before. Self-evidently, by the time they were asked to perform the panning and placement of planets in the second environment (which depended on whether they performed the tests for the speaker or headphone environment first), they had a much better idea of how to perform the requisite interactions.

Of the twenty-two participants, only two found that the project was not of a high enough fidelity. **91%** of participants found the visuals appealing (see Figure 131). The two participants who determined the system's visuals to not provide high enough fidelity also suggested that the visuals appeared blurred, which points to a problem with the VR HMD not having been securely fastened to their heads, and was therefore not necessarily due to the graphical implementation of the Immersive Planets application.

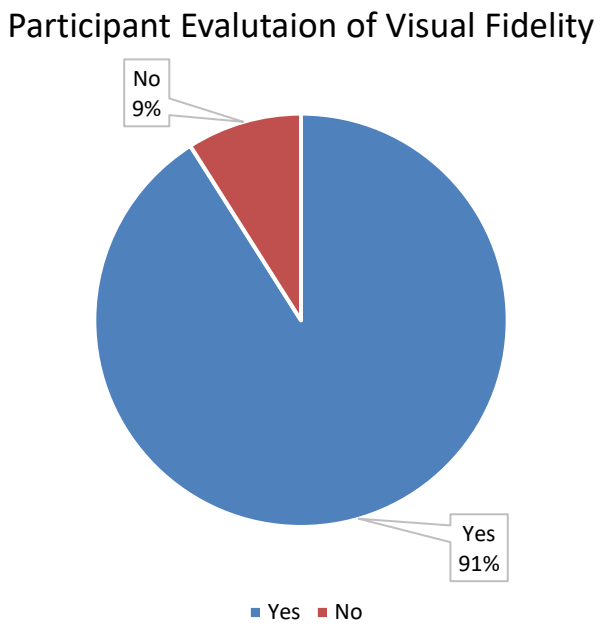

Figure 131: Pie chart indicating that 91% of users found the project's visual fidelity of a high enough standard, while 9% did not.

8 CONCLUSION

"Both science and art have to do with ordered complexity."

— Lancelot Law Whyte

The first section of this conclusion discusses the results that were presented in the previous chapter, and how they relate to the hypotheses of the research. The second section considers the factors that influenced the results and therefore conclusions that have been drawn. In addition, the second section addresses the limitations of both the testing methodology and the implementation. The third section discusses future work for this project and projects within its domain, and, as such, suggests possible modifications of both the methodology and implementation that could alleviate the limitations that are indicated in section two. The fourth and final section provides closing comments from the researcher.

8.1 DISCUSSION OF RESULTS

As previously indicated, the primary hypothesis of the research is that **speaker-based spatialization systems facilitate more natural and intuitive experiences when utilized in immersive VR applications**. It is proposed that despite accuracy and precision differences, speaker-based systems are better candidates for narrative driven immersive worlds and VR art exhibitions and film. The degree to which this hypothesis is correct is discussed in the following section.

Furthermore, the research sought to ascertain how multimodal VR applications affect perceived immersion, and whether the developed system can be categorized as a "highly" immersive one (application). Accordingly, a secondary hypothesis regarding the developed system is that **the immersive system in question is a "highly" immersive system**, and, more generally, that **multimodal VR applications, i.e., applications that enable 3D video, spatial audio, and interactivity, enhance immersion**. This hypothesis is evaluated in the section following the primary hypothesis evaluation.

8.1.1 Primary Aim and Hypothesis

To test the primary hypothesis that was stated in the previous paragraphs, the project, and therefore research, encompassed two important components.

- Firstly, to develop a bespoke testing framework to analyze the speaker-based spatialization system.

- Secondly, to develop a testing methodology that enabled a fair evaluation of both the headphone and speaker-based systems.

The second component depended on the success of the first. The immersive audio capability, which formed the crucial area of the project implementation, was developed to facilitate immersive audio for

the Unity game engine. The capability provided a way to interface the Unity game engine and FMOD Audio Middleware software frameworks with the immersive audio spatialization system, ImmerGo. As indicated, the testing methodology was modelled on MUSHRA recommendations, and a suitably immersive VR application was developed to provide a virtual testing environment by which both the speaker and headphone systems could be evaluated.

The implementation steps of this immersive audio capability are detailed in chapter five. The developed capability did indeed enable the integration of immersive audio into the Unity and audio middleware frameworks. Both the quantitative and qualitative results verify the success of the immersive audio capability's implementation. The listening tests, and subsequent statistical analysis of the accumulated quantitative data, including the distribution of the two data sets, and subsequent variance comparisons, indicate that, although the speaker-based system achieved a lower average positional accuracy than that of the headphone-based environment (**85.8% vs 90.9%**), the variance in positional accuracy of both systems was not statistically significant (and a summary of the key metrics is viewable in Table 17).

These conclusions suggest that, despite a lower average positional accuracy for the speaker environment, this difference can also be attributed to factors other than the implementation itself. Some of these factors are outlined in section 0.

Immersive Audio Environment:	**Average Positional Accuracy:**	**Average Perceived Audio Quality:**	**Environment Preference:**
Speaker-Based	85.8%	75%	59%
Headphone-Based	90.9%	89%	41%

Table 17: A table summarizing the key metrics determined from the experimentation.

The qualitative component of the results indicates that the **perceived audio quality** of the speaker-based system was also lower than that of the headphone-based system **(75% vs 89%)**. Although this represents a substantial difference (of 14%), it was also determined that **59%** of participants preferred the speaker environment, despite this difference in perceived audio quality. When asked to explain why participants preferred the speaker environment, certain key terms such as **more natural**, **less claustrophobic**, and **more immersive** were regularly indicated.

The **positional accuracy** metric would have a distinct influence on the spatial coherence of the audio in either environment and would thus affect the quality of the audio that was reproduced in each system. It is therefore important to recognize that the **perceived audio quality** metric that was obtained via qualitative analysis is a metric that would have intrinsically been affected by the **positional accuracy** of either system. However, the exact relationship between positional accuracy and audio quality remains undetermined. It would be presumptuous to assume that the positional accuracy was the only factor affecting the perceived audio quality. Other factors that might have influenced the audio quality are indicated in section 8.2.1. The information presented in Table 17 shows that this is indeed the case, as the positional accuracy of the headphone environment was higher than that of the speaker environment. Consequently, the expected **perceived audio quality** metric for the headphone environment would be higher than that of the speaker environment. As the data indicates, this is once again true, since the headphone environment was determined to exhibit higher **perceived audio quality** than the speaker environment.

From the qualitative test results, the primary hypothesis appears to be proven as thirteen of the twenty-two participants preferred the speaker environment (which represents approximately 2/3 of the sample group). Further evidence in support of this hypothesis is the fact that, despite a recorded lower positional accuracy and a lower perceived audio quality metric, the speaker environment was preferred to that of the headphone environment. This indicates that the system preference could have been even more skewed towards the immersive audio speaker system had the accuracy and perceived audio quality been higher for this system. Some of factors that would have affected the perceived audio quality of both systems are detailed in the following section.

8.1.2 Secondary Aims and Hypothesis

The secondary aims of the thesis, which provide the foundation for its primary aims, were to:

- Develop an immersive audio capability that integrated the Unity game engine and the Object Based ImmerGo spatialization system frameworks.

- To utilize the implemented capability in the construction of an immersive VR application with high fidelity audiovisual components.

- To facilitate intuitive interaction for this immersive VR application.

The classification of the system as a "highly" immersive one is determined by reviewing the criteria on how an immersive VR system might be graded (which can be reviewed in chapter two). Table 18, which reiterates these criteria, may be used as a reference point in the following paragraphs.

Aspect of Immerstion					
Level Of Immersion	***Inclusive***	***Extensive***	***Surrounding***	***Vivid***	***Matching***
High	Limited signals indicating the presence of device(s) in the physical world (e.g., the weight of an HMD or an eye-tracking device)	Accommodates >2 sensory modalities (e.g., auditory, visual, motor/ proprioceptive); stimuli are spatially oriented	Head-mounted device or surround projection	High fidelity and visual/color resolution; display closely replicates multiple features of the simulated environment in great detail (e.g., correctly placed, dynamic shadows)	Full-body motion capture; visual experience altered to closely match proprioceptive feedback based on whole body movement

Table 18: Excerpt taken from immersive system grading table in chapter two (Miller & Bugnariu, 2016)

If one reviews the table, one should be able to see that the system that has been developed strictly adheres to the *Extensive* and *Surrounding* criteria. The system also met the requirements of the *Inclusive* criterion, although it could be argued that the "claustrophobia" participants experienced when using the headphone environment suggests that the headphone component of the system might more accurately be classified as a *moderately* immersive system. The *Vivid* criteria for *high* immersion was also achieved, since the qualitative results show that **91%** of participants found the VR application's visuals to have been at a high fidelity.

The final criterion, namely *Matching*, indicates that the immersive VR application that was implemented did not strictly meet the requirements of Full-body motion capture. However, the application did consider a user's movements in the virtual space. That is, by using the positional tracker, a user's real world (body) movements were accurately rendered into the virtual environment. In addition, the intuitive interaction modality achieved using the LMC was a useful way of accurately translating users' arm and hand movements into virtual movements. Indeed, the usability metric was determined to be **8.3** (out of 10), which suggests that the interaction modality was successfully implemented, although the precision of this interaction was not high. It is therefore argued that the *Matching* aspect of the project sits between a *moderate* and *high* level of immersion. Nevertheless, this aspect does indicate that the project succeeded in its aim of facilitating intuitive interaction.

Table 19 indicates the grading results of the immersive VR application according to the criteria that have been detailed.

	Aspect of Immersion				
	Inclusive	***Extensive***	***Surrounding***	***Vivid***	***Matching***
Headphone	M	H	H	H	MH
Speaker	H	H	H	H	MH
Overall	MH	H	H	H	MH

Table 19: A table that indicates the immersive grading of the VR application that was developed. The 'M' indicates a moderate level of immersion, and the 'H' represents a high level of immersion.

The previous table shows that the overall system had three out of five aspects of immersion that were *highly* immersive. The other two aspects were graded as *moderately* to *highly* immersive.

This therefore indicates that the secondary hypothesis of the project was not proven completely. However, if one considers that the speaker environment had four out of five aspects that could be classified as *highly* immersive, it follows that the system would fall under a *highly* immersive categorization had full body tracking been implemented into the solution. This further **proves the primary hypothesis**, since the headphone environment can be said to have **increased** the number of "signals indicating the presence of device(s) in the physical world", which has a negative effect on the level of immersion of a system (Miller & Bugnariu, 2016).

8.2 Discussion of Influencing Factors

This section will discuss some of the issues that both the testing methodology and implementation (testing framework) encountered. It also addresses the effect that certain aspects of both components had on the conclusions and results that are presented in the previous section. In presenting these issues, the reasons behind design choices and the implementation constraints that allowed for these issues are also provided. Possible solutions for some of these problems and factors will be presented in section 0.

8.2.1 Methodology Factors

There were a few aspects of the methodology that could have influenced the results. The most important of these is the attributes that were determined to have affected **audio quality.** Those attributes that were considered were selected according to the ITU-R BS.1284-2 recommendation and included both the temporal and spatial coherence of the audio across both environments, as well as the "coherence of spatial impressions between sound and picture" (ITU, 2019). In other words, the three

attributes that were chosen to best represent **audio quality** in this research. While the recommendation indicates that any number of attributes could influence the quality of the audio of a listening test, the project's experimental design and implementation attempted to reduce as many of these factors as possible by remaining consistent in numerous areas across both audio environments and implementations. However certain attributes were not considered, the most significant of which being the testing conditions of the speaker environment, i.e., the room response frequency for the speaker environment.

8.2.1.1 Multimodal Listening Tests

While a discussion of multimodal requirements and how they affect listening tests was provided in the previous chapter, it is nevertheless worthwhile to review an important source of research that sought to determine the influence of visual information in sound source localization from the perspective of VR. In the research conducted by Ahrens and associates, eight tests were conducted that comprised of different auditory-visual scenarios (Ahrens, et al., 2019). The visual information comprised of three categories that consisted of a listener being present within a real acoustic reproduction chamber, another being situated in a virtual acoustic reproduction chamber in VR (modelled on the real chamber), and a third being blindfolded. It was found that sound localization accuracy in VR was only slightly negatively affected when compared to localization accuracy in the real visual information category (Ahrens, et al., 2019). In addition, the researchers' findings suggest that sound localization accuracy was higher for both visual information categories than was the case with the blindfolded tests (Ahrens, et al., 2019).

A similar study that aimed to assess the impact of visual stimuli on perceptual attributes of immersive audio found that experimentation that aimed to establish a sense of realism and presence should consider the impact of both visual and auditory modalities (Woodcock, et al., 2019). This research suggests that visual stimulus influenced "realism, sense of space, and spatial clarity" (Woodcock, et al., 2019). Both the reviewed studies thus indicate that there is an intrinsic link between audio and visual information and reinforce the notion that multimodal listening tests are important in the context of categorizing audio in the new age media and VR technologies.

8.2.1.2 Experimental Metrics

With reference to the metrics that were determined and presented in chapter seven, there are reasonable grounds to suggest that an alternative and more representative set of metrics may have been used in the assessment of the spatial audio systems. Shoeffler et al classify certain metrics that assess timbral and spatial audio quality as "basic audio quality" (BAQ) metrics which result from traditional listening tests (Schoeffler, et al., 2016). They argue that BAQ "abstracts" from other factors such as the audio tracks that have been selected for use in the testing, and this facilitates listener bias in the test results as some listeners would like or dislike the selected track(s). The corollary, here, is that BAQ metrics, which are regarded as clear ways of determining conclusions from results, in fact facilitate biases (Schoeffler, et al., 2016).

To mitigate against the bias of BAQ tests, the research suggests that "Quality of Experience" (QoE) metrics, which are used to determine user satisfaction, be employed when modeling real world situations. When expressly assessing audio signals, Shoeffler et al propose that the "Overall Listening Experience" (OLE) is a more relevant way of assessing audio when using subjective listening tests as the mode to determine audio quality (Schoeffler, et al., 2016). There is an argument that OLE metrics

provide a better indication of the wholistic experience a listener might have, and that the metric would facilitate a better reflection as to whether an experience has been positively perceived (Schoeffler, et al., 2016).

8.2.1.3 Listening Test Panel

The ITU-R 1116-3 recommendation of using twenty or more test subjects to validate the data from the listening tests was adhered to. However, as was indicated in the previous chapter, it was difficult to select 'experienced' listeners for testing due to the location in which the research was conducted. Although nine of the twenty-two tests subjects were regarded as experienced listeners, the recommendation indicates that at least half of the sample group should be "experienced." Importantly, in this regard, the relative novelty of VR technologies at the time and location in which the research was completed were also aspects that affected whether a participant was considered "experienced" or not.

8.2.1.4 Time Taken to Identify Audio Sources

Another issue with the methodology was that the researcher did not record how long it took for participants to correctly identify audio sources. This metric would have provided useful insight into whether there were distinct patterns for audio sources in either environment. Similarly, it would also have facilitated a different way in which the two immersive audio systems could have been contrasted with each other: that is, by enabling the researcher to determine whether one system allowed participants to identify audio sources more rapidly than the other. However, while the actual value for the time taken for each listening test was not recorded, the experimental conditions were such that each participant was provisioned 20 seconds for each listening test. As mentioned in the previous chapter, if participants were unable to identify the source within this time frame, the identification was considered a failure, and the next test would begin.

8.2.2 Implementation Factors

While the previous section indicated some of the experimental and methodological factors and elements that could have been applied differently, this section, and the ones that follow, will analyze some of the factors and design decisions that were taken with regard to the implementation of the immersive audio capability and the testing framework (the 'immersive planets' VR application).

8.2.2.1 The IPC mechanism

The Windows Sockets API (Winsock) proved to be an effective means of providing Interprocess communication (IPC). It was also a way to remain consistent with the networking characteristics of other components in the overall system, for example, the traditional ImmerGo client and server use Winsock communication.

In the evaluation of different IPC mechanisms with regard to transfer rate and latency, it has been suggested that alternate IPC mechanisms enable higher throughput and lower latency than socket IPC implementations (Venkataraman & Jagadeesha, 2015). The implication is that both shared memory and Pipes (Named and Unnamed), which exhibit high transfer rates and low latencies, offer greater efficiency than Sockets. However, the research does also suggest that the bidirectional design of sockets and the ability to perform IPC across different machines provide an incentive to use them in IPC due the scalability that these two features enable (Venkataraman & Jagadeesha, 2015).

Accordingly, despite the efficiency advantages of alternate IPC mechanisms, the Socketing implementation remained a desirable means to achieve communication between the components of the immersive audio capability. Section 8.3.2.1 discusses possible ways in which alternate IPC mechanisms could have been used.

8.2.2.2 Buffer Padding the ASIO Playback Server

One of the critical issues that requires addressing was the need of the ASIO playback server to receive audio samples (defined by a variable indicating how many blocks of the buffer size were needed) before audio was streamed out over the AVB network using the ASIO Streamware NIC. Although this issue was mitigated using a small padding size, i.e., setting the number of audio sample blocks to two sets of 256 samples (indicating one full set of samples as the buffer size is 512), it was something that would affect the temporal coherence of certain types of audio sources within a digital game or VR experience. Even a small padding size of 1024 samples would induce an additional latency of 21.3ms. Time sensitive audio, such as the dialogue of a character, would not tolerate additional latency before a user would become aware of a temporal mismatch. This can be attributed to the perceptual threshold of audio latency, which can be defined as the time taken before a user's performance or experience is impaired – a threshold which is suggested to be between **20 – 70ms**, depending on the physiological attributes of the user (Attig, et al., 2017). In this instance, the cumulative temporal latency at the start of an audio source has been estimated to be around **~32.9ms** when using a padding of two blocks. This playback latency is therefore an acceptable latency for this system, but a reduction of this latency to beneath **20ms** would be ideal. Moreover, as was suggested in the previous chapter it can be assumed that this latency would remain static until audio playback has finished. However, no additional padding requirements were needed past the triggering phase of the audio event.

8.2.2.3 Speaker-Based Rendering Algorithm

A further aspect of the implementation that might have affected the results that were determined, was the selection of the rendering algorithm for the speaker-based immersive audio system. In chapter three, it was indicated that the DBAP algorithm was desirable for a number of reasons, the most important of which was that it does not require listeners to inhabit a "sweet-spot". In other words, the algorithm performs well when listeners inhabit areas offset from the central spot (Simon, et al., 2017). Due to the nature of the experience, it was almost impossible to keep a participant from moving away from the "sweet-spot". Of course, the listener could have been seated as opposed to standing, but due to the application seeking to realistically model how VR experiences are being produced (enabling users to move about a physical space and these movements being mapped virtually), it was decided that they should be standing for the experimentation.

8.2.2.4 Rendering Algorithms for Both Environments

Another important issue when considering factors that might influence the experiments is that the rendering algorithm for each environment was different. The loudspeaker rendering algorithm for the speaker environment was DBAP (see previous section). For the headphone environment, the Oculus implementation of spatial sound makes use of a generalized HRTF implementation, and the ITDs/ILDs as well as spectral cues will not match each individual listener. The choice of rendering algorithm for either system and environment can be seen as a hard limitation of each output system (speakers or headphones). HRTFs cannot be applied to loudspeakers with satisfactory results, as the HRTFs are calculated for on or in-ear speakers, i.e., headphones. The same can be said for DBAP when applied to

headphones, as DBAP is an algorithm that was devised for an array of loudspeakers, and headphone audio reproduction was not considered in its design (Tarzan, et al., 2019). There are suggestions that DBAP can be applied to headphones, however. This would involve a translation of the determined output values for virtual loudspeakers using DBAP, into HRTFs which would be calculated with the aid of a head-tracking device (Tarzan, et al., 2019). However, in this scenario, the same issue would persist, as the comparison would remain a comparison between DBAP for loudspeakers and HRTFs for headphones (albeit determined from DBAP calculations). For the reasons mentioned above, it was unfeasible to utilize the same rendering algorithm for the two different output systems.

8.2.2.5 Audio Source Position

The results pertaining to which audio sources were more accurately identified than others on average indicated an interesting trend with regards to the speaker environment, since they suggest that audio sources (planets) closer to the listener were less accurately identified. This trend also implies that both the spatial coherence and the "coherence of spatial impression" attributes of how **audio quality** was analyzed would have been negatively influenced for the speaker environment.

A possible answer as to why this was the case could be that the DBAP algorithm, although allowing a listener to move away from the "sweet spot", would still exhibit localization inaccuracies when a listener (participant) moved too far from the "sweet-spot". This indicates that there was a maximal offset by which a user could stray from the center of the speaker array. Another plausible answer as to why closer audio sources were less accurately identified than further sources could be that the rendering algorithm provides higher localization accuracy for objects that are located along the surface of the recreated sound field. This then denotes less accuracy of audio sources that are located within, or closer to, the centroid of the hemispherical speaker array.

8.2.2.6 Output Device Selection

There is reason to believe that the speakers may have been preferred over the headphones due to the specific output devices that were employed in the study. However, the following section provides an argument that this would only have had a minor influence. Moreover, despite the headphone environment enabling more accurate spatialization, as well as a higher average **perceived audio quality**, the participants still preferred the speaker system.

8.2.2.6.1 Audio Codec, Volume and Equalization Consistency

The same software audio codec was used for both sets of FMOD audio events, as the same source audio track was used for each of the eight audio sources. This therefore rendered a situation in which the software audio codec should not have had an express impact on the quality metric that was attained. However, there is a suggestion that, due to the proximity of headphones to a participant's ears, the latter might better detect slight distortions in the audio tracks (Kallinen & Ravaja, 2007).

While the software audio codecs for both audio environments were consistent, it is important to recognize that the hardware that performed the Digital to Analogue conversions for both environments was different (see chapter seven). The DAC for the speaker environment was performed by the N-DAC8s, while the same process for the headphone environment was performed by the WASAPI audio driver and native DAC of the computer's motherboard. This difference would have resulted in a difference of audio quality between both systems, but it is assumed that this difference was insignificant.

The volume and equalization settings of the two sets of FMOD audio events for both the speaker and headphone environments were kept consistent when the events were authored within the FMOD Studio Application, as well as when they were triggered from the Unity game engine.

8.3 FUTURE WORK

"We can't stop here, this is bat country!"

— Hunter S. Thompson

This section should be viewed in conjunction with the previous section, which indicates some of the factors that might have influenced the experimentation and project implementation. As a result, the section provides possible solutions to the issues and factors that have been raised, and suggests modifications that could be made to the methodology and implementation in future iterations of the project. Similarly, it indicates aspects of the project that are relevant to related fields of research.

8.3.1 Possible Modifications to Methodology

The following paragraphs indicate possible enhancements to the methodology that was adopted for this project and indicate ways in which the conclusions drawn at the start of the chapter might be more robustly determined.

8.3.1.1 Experimental Metrics, Identification Time, and Listening Test Panel Selection

As was indicated in section 8.2.1.2, future research would include the addition of QoE and OLE metrics, to better determine user experience of the system, and to remove some of the intrinsic bias of BAQ metrics. Section 8.2.1.4 discussed how the time taken to identify audio sources was not a metric that was recorded. In future iterations of this experiment, the time taken to identify an audio source should be recorded and statistically interpreted.

In addition, it would be beneficial for the experimenter to draw conclusions from a sample group that is more experienced with listening test procedures, and therefore what to focus on when analyzing audio quality. Of course, the experiments would also benefit from having more participants who are accustomed to VR experiences. With the rise of the Oculus Quest 2, and subsequent endorsement of VR in the public sphere,[85] it is far more likely that in future experimental scenarios participants will have had more exposure to both spatial audio and VR. However, the reader should recognize that accruing such a sample group with this expertise was just not possible in the context in which the research was performed.

8.3.2 Possible Modifications to Implementation

The following paragraphs provide suggestions about possible modifications to the implementation of both the immersive audio capability and the testing framework (the VR application that was developed to contrast the headphone- and speaker-based immersive audio systems).

[85] https://qz.com/2107700/metas-oculus-was-the-top-app-in-the-us-on-christmas/

8.3.2.1 Winsock and other IPC mechanisms

In future iterations of this project, it would prove beneficial to compare differing IPC mechanisms to determine which one provides the lowest latency. One of the reasons Sockets (Winsock) provides lower efficiency than either Shared Memory or Pipes, is that Sockets will often have to break large messages into separate packets to split the payload (Venkataraman & Jagadeesha, 2015). In the implementation of the ImDSP plugin and the ASIO Playback server, it was found that the payload of a single Winsock UDP packet could not contain a full set of 512 32-bit floating point audio samples. As a result, two UDP packets were transmitted by the ImDSP plugins to the ASIO Playback server, which each contained 256 audio samples.

Although the motivation behind using a Socket approach to providing IPC in the immersive audio capability's implementation is valid, alternate implementations would be useful to compare efficiencies. In these modified versions of the immersive audio capability, presumably the strategy would be for the playback server to allocate a portion of Shared Memory through which the ImDSP clients would be able to access and offload their audio samples. Of course, various locking mechanisms such as semaphores or mutex objects would need to be implemented to ensure that the playback server would have access to the memory as required. Additionally, these locking mechanisms would need to ensure that each client was provided a piece of Shared Memory that only they and the playback server could access exclusively. Figure 132 indicates diagrammatically how this solution might function.

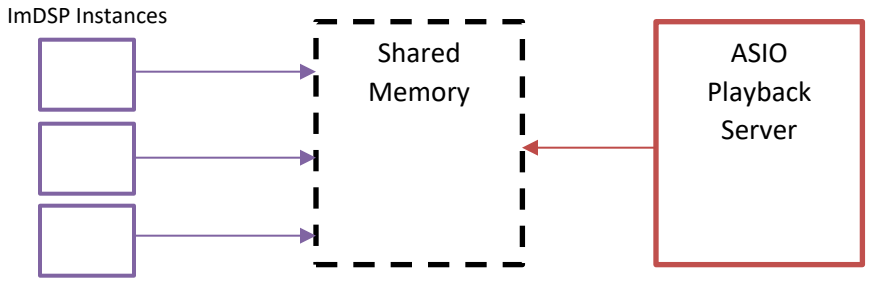

Figure 132: A diagram indicating how a Shared Memory IPC solution might function in a modified implementation.

8.3.2.2 More than Monophonic Audio Tracks

One of the restrictions imposed on the sound designer using the immersive audio capability is that the source audio tracks need to be downmixed. Hypothetically this downmixing could be achieved by the ImDSP plugin itself when interpreting audio sample data, as the *process* and *read* callbacks indicate the channel format of the audio data being processed.

If a non-monophonic audio track was being processed (i.e., a stereophonic or five-channel track), a suitable downmixing process could be selected to process each sample, and then populate the audio sample packets being transmitted to the playback server. For example, in the case of a stereophonic audio track, the average of every two samples (a rudimentary downmixing function) could be transmitted to the server as a single sample. This single sample would represent the average of the two channels encoded in the audio data. Similarly, if a five-channel audio track was being processed, every five samples could be averaged, and this resultant sample (representing the monophonic interpretation of the five-channel audio) could be used to populate the audio packet being transmitted to the server.

If the immersive audio capability were to be used in commercial developmental processes, then this type of functionality would serve as a failsafe in case Unity and FMOD developers were to use audio

tracks that were not monophonic. Of course, the process of determining an average would add to the processing requirement of each ImDSP plugin, and this may thus not be a desirable function if there is a large array of ImDSP plugin instances, and therefore audio events being triggered at a single time.

8.3.2.3 Support for Additional Rendering Algorithms

While the DBAP rendering algorithm proved to be suitable for this project's requirements, it would also be beneficial to determine which audio rendering algorithms might perform better in different circumstances. In other words, a desirable aspect of the final implementation would be enabling developers to decide which rendering algorithm they would like to use in their specific project. This would also provide a flexibility to the immersive audio capability that is currently not present, i.e., the developer of the project would decide which rendering algorithm best suits their playback environment.

Additionally, seeing as Ambisonics has become a highly endorsed mechanism by which immersive audio is rendered in VR, modifications to this project would encompass determining how ambisonic recordings of a virtual sound field might differ when rendered using a speaker-based array or headphones. However, the rendering algorithm of the speaker-based spatialization system would remain the same. The major implementation difference would revolve around how the encoding of audio sources would be consistent across both speaker and headphone environments, and how the positional metadata would be interpreted by the ImmerGo spatialization system.

Another Audio Middleware, WWise, provides an ambisonic audio solution,[86] and therefore a potential investigation stemming from this project would be to contrast how the implemented immersive audio capability using FMOD rivals that of the Wwise immersive audio ambisonics encoder and decoder.

Furthermore, the utilization of generalized HRTFs as opposed to individualized HRTFs, would have had an impact on the test results, and the experimental conditions. Therefore, in future iterations of this research, it would be desirable to perform individualized HRTF recordings for each participant of the experiments. After which, a custom virtual loudspeaker rendering solution could be implemented to integrate the individualized sets into the audio reproduction system. Due to resource and time constraints, this addition to the experimental design was not feasible, however.

8.3.2.4 Native Playback Server

An optimization of the final implementation could also be to find a way to embed the ASIO playback server inside the Unity game engine. At present, the server exists as a discrete console application, but in future iterations of the implementation, it would be advantageous to find a way to embed the server within the Unity application, so that a developer could edit settings from the Unity Editor and thereby enable a less segmented solution, i.e., a solution consisting of fewer discrete parts.

8.3.2.5 Enhancing the ImDSP plugin

As with the previous section, an enhancement of the ImDSP plugin would also be a desirable change to the project's current implementation. Presently, the GUI controls for the plugin, when imported into the FMOD Studio Application, only indicate the name of the plugin as well as the client number (which can be set to 32 different clients). This is indicated in Figure 133. The figure also provides an example of how

[86] https://www.audiokinetic.com/products/ambisonics-in-wwise/

the native Oculus Spatializer plugin's GUI looks within the FMOD Studio Application, and what variables this plugin exposes to the sound designer.

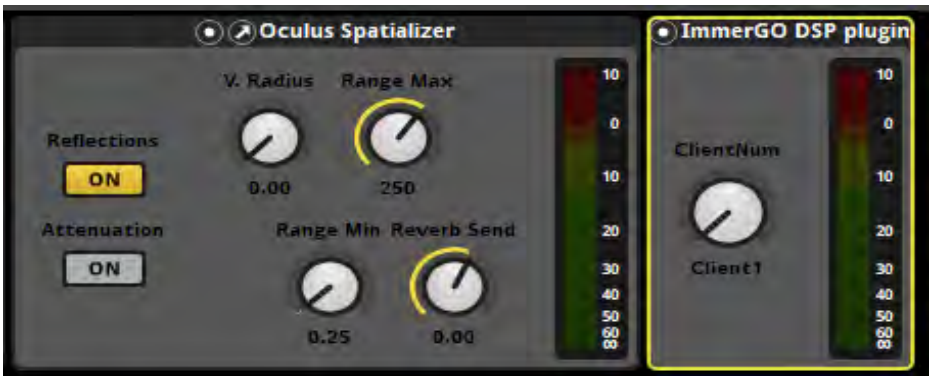

Figure 133: A comparison of the GUI for the Oculus Spatializer and the ImDSP or ImmerGo DSP plugin on the effect deck of an FMOD audio event in the studio Application.

In future implementations of the ImDSP plugin, it would be useful to facilitate the addition of parameters that a sound designer might set. For example, the spread of the FMOD audio source, as well as attenuation parameters could be provided, and the latter would ideally be exposed to the sound designer from within the FMOD Studio Application. Also, despite significant overheads, the test could be run with individualized HRTF measurements taken from the actual physical loudspeaker rig.

8.4 Closing Comments

It is important to recognize that, although the primary hypothesis was proven to be correct (that is, that **speaker-based spatialization systems facilitate more natural and intuitive experiences when utilized in immersive VR applications**), this proof will not apply to all instances of VR applications. Indeed, VR applications requiring more precise and accurate spatial audio should consider using a headphone rather than speaker implementation of immersive audio, since the overheads of a user requiring all the requisite hardware and software for an immersive speaker system are much higher than those of headphones (Garner, 2017). These overheads include, but are not limited to:

- The space requirements, i.e., enough space in a room or theatre to facilitate a large speaker array.

- The acoustic qualities of the reproduction space also need to be considered, to provide the best conditions by which audio can be reproduced by speakers.

- The financial cost of a large speaker spatialization system, which is very high when considering the equivalent cost of high-quality headphones.

As was indicated in section 8.1, the researcher believes that a speaker-based immersive audio system would better suit VR art exhibitions and narrative-driven experiences (such as VR film), in which the overheads of a speaker-based immersive system are not a prohibitive factor when deciding on how best to present the digital media. With regards to the immersive qualities that either headphone- or speaker-based reproduction devices offer, it was found that, when used for the constructed immersive VR experience, speakers provided higher immersive qualities.

Additionally, it has been suggested that the psychological conditions of listeners have a direct influence on their preference for the mode in which audio is output. Research suggests that listeners with high sociability and activity personalities are more likely to favor audio output through speakers (Kallinen & Ravaja, 2007). Similarly, the degree to which a user is immersed in a VR application will also depend on their response to the visual and auditory content which is presented to them. For example, certain virtual scenarios might deeply influence one user, whilst the same scenario might elicit a minor response in another (Bollmer, 2017). As a result, the researcher acknowledges that immersion in VR is a concept that is affected by the content that is presented in the experience.

Nevertheless, this research sought to distinguish how headphone- or speaker-based audio might influence immersive VR applications and did produce clear results which favored the latter's immersive impact. Moreover, if one considers a more technical definition of immersion, and the layered model by which immersion can be categorized (see chapter two), then this research aimed to formally analyze the auditory layer of immersion and, more specifically, how headphone- and speaker-based spatial audio systems affect this layer. Here, too, the multi-speaker array produced a more layered and immersive experience. The research has therefore laid the groundwork for further investigations into how immersion is affected by the modality of the audio output system.

A further conclusion that can be drawn from the results of the experiments, is that there is a consensus that VR technology will **undoubtedly disseminate throughout society**. All twenty-two participants indicated that they believed VR would become more prevalent in our society (see question 9 of the questionnaire). Accordingly, this research contributes to the analysis of the current state of audio in VR and enables a useful means of abstracting the concept of immersion into various layers which indicate how the immersive quality of VR systems might better be determined in future research.

REFERENCES

Ahrens, A., Lund, K. D., Marschall, M. & Dau, T., 2019. Sound source localization with varying amounts of visual information in virtual reality. *PloS one,* 14(3).

Ahrens, J., Rabenstein, R. & Spors, S., 2008. The theory of wave field synthesis revisited. In: *Audio Engineering Society Convention 124.* s.l.:Audio Engineering Society.

Alberti, P. W., 2001. The anatomy and physiology of the ear and hearing. *Occupational exposure to noiseL Evaluatiion, prevention, and control,* pp. 53-62.

Alderisi, G., Iannizzotto, G. & Bello, L. L., 2012. Towards IEEE 802.1 Ethernet AVB for advanced driver assistance systems: A preliminary assessment. In: *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation.* s.l.:IEEE, pp. 1-4.

Al-Sheikh, B., Matin, M. A. & Tollin, D. J., 2019. Head related transfer function interpolation based on finite impulse response models. In: *2019 Seventh International Conference on Digital Information Processing and Communications.* s.l.:IEEE, pp. 8-11.

Anthes, C., García-Hernández, R. J., Wiedemann, M. & Kranzlmüller, D., 2016. *State of the art virtual reality technology.* s.l., 2016 IEEE Aerospace Conference.

Astley, M., Sturman, D. C. & Agha, G. A., 2001. Middleware. *Communications of the ACM,* 44(5), p. 99.

Atiyah, F. & Izzah, L., 2019. A comparative study on the effectiveness of using direct and audiovisual methods for enhancing students listening comprehension. *English Language in Focus (ELIF),* 2(1), pp. 9-16.

Attig, C., Rauh, N., Franke, T. & Krems, J. F., 2017. System latency guidelines then and now-- is zero latency really considered necessary?. In: *International Conference on Engineering Psychology and Cognitive Ergonomics.* s.l.:Springer, pp. 3-14.

Bachmann, D., Weichert, F. & Rinkenauer, G., 2018. Review of three-dimensional human-computer interaction with focus on the leap motion controller. *Sensors,* 18(7), p. 2194.

Basso, A., 2017. Advantages, Critics and Paradoxes of Virtual Reality Applied to Digital Systems of Architectural Prefiguration, the Phenomenon of Virtual Migration. In: *Multidisciplinary Digital Publishing Institute Proceedings.* s.l.:s.n., p. 915.

Bates, J., 1992. Virtual Reality, art, and entertainment. *Teleoperators & Virtual Environments,* 1(1), pp. 133-138.

Berger, C. C. et al., 2018. Generic HRTFs may be good enough in virtual reality. Improving source localization through cross-modal plasticity. *Frontiers in neuroscience,* Volume 12, p. 21.

Berkhout, A. J., de Vries, D. & Vogel, P., 1993. Acoustic control by wave field synthesis. *Acoustical Society of America,* 93(5), pp. 2764-2778.

Biocca, F. & Delaney, B., 1995. *Immersive Virtual Reality Technology.* New Jersey: Lawrence Erlbaum.

Boas, Y. A. G. V., 2013. *Overview of Virtual Reality Technologies.* Southhampton, Interactive Multimedia Conference.

Bollmer, G., 2017. Empathy Machines. *Media International Australia,* 165(1), pp. 63-76.

Bos, D., Miller, S. & Bull, E., 2021. Using virtual reality (VR) for teaching and learning in geography: fieldwork, analytical skills, and employability. *Journal of Geography in Higher Education,* pp. 1-10.

Botha, M., 2017. *The Parsing and Construction of a SMPTE-Compliant Bitstream.* Grahamstown: Rhodes University.

Breebaart, J. et al., 2008. Spatial audio object coding (SAOC) - The upcoming MPEG standard on parametric object based audio coding. In: *Audio Engineering Society Convetion 124.* s.l.:Audio Engineering Society.

Brooks, F. P., 1999. What's real about virtual reality?. *IEEE Computer graphics and applications,* 19(6), pp. 16-27.

Bufacchi, V., 2021. *Facebook: Virtual or Virtueless Reality?.* s.l.:Irish Examiner.

Bun, P., Grajewski, D. & Gorski, F., 2021. Assessment of mixed-reality devices for production engineering. In: *International Conference Innovation in Engineering.* s.l.:Springer, pp. 472-483.

Buyuksalih, I. et al., 2017. 3D Modelling and visualization based on the Unity game engine- advantages and challenges. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences,* Volume 4.

Çamcı, A. & Hamilton, R., 2020. Audio-First VR: New perspectives on musical experience in virtual environments. *Journal of New Music Research,* 49(1), pp. 1-7.

Cecchi, S., Carini, A. & Spors, S., 2018. Room response equalization - A review. *Applied Sciences,* 8(1), p. 16.

Chattha, U. A. et al., 2020. Motion sickness in virtual reality: an empirical evaluation. *IEEE Access,* Volume 8.

Cheng, C. I. & Wakefield, G. H., 1999. Introduction to head-related transfer functions (HRTFs): Representations of HRTFs in time, frequency, and space. In: *Audio Engineering Society Convention 107.* s.l.:Audio Engineering Society.

Cipresso, P., Giglioli, I. A. C., Raya, M. A. & Riva, G., 2018. The past, present, and future of virtual and augmented reality research: a network and cluster analysis of the literature. *Frontiers in psychology,* Volume 9, p. 2086.

Cogburn, J. & Silcox, M., 2014. Against brain-in-a-vatism: on the value of virtual reality. *Philosophy & Technology,* 27(4), pp. 561-579.

Coleman, P. et al., 2018. An audio-visual system for objet-based audio: from recording to listening. *IEEE Transactions on Multimedia,* 20(8), pp. 1919-1931.

Collins, K., 2008. *Game sound: an introduction to the history, theory, and practice of video game music and sound design.* s.l.:MIT Press.

Daniel, J., Moreau, S. & Nicol, R., 2003. Further investigations of high-order ambisonics and wavefield synthesis for holophonic sound imaging. In: *Audio Engineering Society Convention 114.* s.l.:Audio Engineering Society.

Davis, L. S. et al., 2005. High order spatial audio capture and its binaural head-tracked playback over headphones with HRTF cues. In: *Audio Engineering Society Convention 119.* s.l.:Audio Engineering Society.

de França, A. C. P. & Soares, M. M., 2017. Review of virtual reality technology: an ergonomic approach and current challenges. In: *International Conference on Applied Human Factors and Ergonomics.* s.l.:Springer, pp. 52-61.

De Vries, D. & Boone, M. M., 1999. Wave Field synthesis and analysis using array technology. In: *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.* s.l.:IEEE, pp. 15-18.

Denisova, A. & Cairns, P., 2015. *First person vs. third person perspective in digital games: do player preferences affect immersion?.* s.l., Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.

Devonport, S. & Foss, R., 2018. A Flexible Approach for the Application of Immersive Audio to an Installation Performance. *International Syumposium on Electronic Art,* pp. 12-21.

Devonport, S. & Foss, R., 2018. A Flexible Approach for the Application of Immersive Audio to an Installation Performance. *International Syumposium on Electronic Art,* pp. 12-21.

Devonport, S. & Foss, R., 2019. The Distribution of Ambisonic and Point Source Rendering to Ethernet AVB Speakers. *ICASA.*

di Lanzo, J. A. et al., 2020. A review of the uses of virtual reality in engineering education. *Computer Applications in Engineering Education,* 28(3), pp. 748-763.

Dichgans, J. & Brandt, T., 1978. Visual-vestibular interaction: Effects on self-motion perception and postural control. In: *Perception.* s.l.:Springer, pp. 755-804.

Didyk, P. et al., 2010. *Perceptually-motivated real-time temporal upsampling of 3D content for high-refresh-rate displays.* s.l., Computer Graphics Forum, pp. 713-722.

Dolby, 2018. *Dolby Atmos® Home Theater Installation Guidelines,* s.l.: Dolby Laboratories, Inc..

Dolby, 2021. *Module 6.4 - Unlocking a Dolby Atmos Master File for Punch-ins and Metadata Updates,* s.l.: Dolby Laboratories, Inc.

Dolby, 2021. *Module 7.3 - Re-renders,* s.l.: Dolby Laboratories, Inc.

Dolby, 2021. *professional.dolby.com.* [Online]
Available at: https://professional.dolby.com/content-creation/Dolby-Atmos-for-content-creators/3
[Accessed 10 10 2021].

Eschen, H. et al., 2018. Augmented and virtual reality for inspection and maintenance processess in the aviation industry. *Procedia manufacturing,* Volume 19, pp. 156-163.

European Broadcasting Union, 2021. *EBU ADM Guidelines,* s.l.: s.n.

Feng, J., Kim, J., Luu, W. & Palmisano, S., 2019. Method for estimating display lag in the Oculus Rift S and CV1. In: *SIGGRAPH Asia 2019 Posters .* s.l.:s.n., pp. 1-2.

Firelight Technologies Pty, Ltd, 2021, 2021. s.l.: s.n.

Firelight Technologies Pty, Ltd, 2021, 2021. *Studio API Guide,* s.l.: s.n.

Firelight Technologies Pty, Ltd, 2021. *FMOD for Unity,* s.l.: FMOD.

Firelight Technologies Pty, Ltd, 2021. *FMOD Low Level API Documentation,* s.l.: Firelight Technologies Pty, Ltd.

Fleming, J. T., Noyce, A. L. & Shinn-Cunningham, B. G., 2020. Audio-visual spatial alignment improves integration in the presence of a competing audio-visual stimulus. *Neuropsychologia,* Volume 146.

Flores-Arredondo, J. H. & Assad-Kottner, C., 2015. Virtual Reality: a look into the past to fuel the future. *The Bulletin of the Royal College of Surgeons of England,* 97(10), pp. 424-426.

FMOD, 2021. *White Papers | DSP Plugin API.* [Online]
Available at: https://fmod.com/resources/documentation-api?version=2.01&page=white-papers-dsp-plugin-api.html
[Accessed 10 4 2019].

Foss, R. & Rouget, A., 2015. Immersive audio content creation using mobile devices and ethernet avb. *Audio Engineering Society,* Volume Convention 139.

Foss, R. & Rouget, A., 2016. *Approaches to immersive audio content creation.* s.l., Tonmeistertagung-VDT International Convention.

Frank, M., Zotter, F. & Sontacchi, A., 2015. Producing 3D audio in ambisonics. In: *Audio Engineering Society Conference: 57th Internation Conference: The Future of Audio Entertainment Technology.* s.l.:Audio Engineering Society.

Freedman, E., 2018. Engineering Queerness in the Game Development Pipeline. *Game Studies,* 18(3).

Freeman, J., Avons, S. E., Davidoff, J. & Pearson, D. E., 1997. Effects of stereo and motion manipulations on measured presence in stereoscopic displays. *Perception,* 26(1), p. 144.

Freina, L. & Ott, M., 2015. *A literature review on immersive virtual reality in education: state of the art and perspectives.* s.l., s.n.

Freina, L. & Ott, M., 2015. *A Literature Review on Immersive Virtual Reality in Education: State Of The Art and Perspectives.* Bucharest, s.n.

Furness, R. K., 1990. *Ambisonics- an overview.* s.l., Audio Engineering Society.

gameprogrammingpatterns, 2021. *Component.* [Online]
Available at: https://gameprogrammingpatterns.com/component.html
[Accessed 20 11 2021].

Gardner, W. G. & Martin, K. D., 1995. HRTF measurements of a KEMAR. *The Journal of the Acoustical Society of America,* 97(6), pp. 3907-3908.

Garner, T. A., 2017. *Echoes of Other Worlds: Sound in Virtual Reality: Past, Present and Future.* s.l.:Springer.

Garrett, B. et al., 2018. Virtual reality clinical research: promises and challenges. *JMIR serious games,* 6(4).

Geier, M., Ahrens, J. & Spors, S., 2010. Object-based audio reproduction and the audio scene description format. *Organised Sound,* 15(3), pp. 219-227.

Genzel, D. et al., 2018. Psychophysical evidence for auditory motion parallax. *Proceedings of the National Academy of Science,* 115(16), pp. 4264-4269.

Gerschütz, B., Fechter, M., Schleich, B. & Wartzack, S., 2019. *A Review of Requirements and Approaches for Realistic Visual Perception in Virtual Reality.* s.l., Cambridge University Press.

Gerzon, M. A., 1973. Periphony: With-height sound reproduction. *Journal of the audio engineering society,* 21(1), pp. 2-10.

Goradia, I., Doshi, J. & Kurup, L., 2014. A review paper on Oculus Rift & Project Morpheus. *International Journal of Current Engineering and Technology,* 4(5), pp. 3196-3200.

GrandViewResearch, 2021. *Grand View Research.* [Online]
Available at: https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market
[Accessed 24 10 2021].

Grau, O., 2003. *Virtual Art: from illusion to immersion.* s.l.:MIT Press.

Gualeni, S., 2016. The experience machine: Existential reflections on virtual worlds. *Journal of Virtual Worlds Research,* 9(3).

Guna, J. et al., 2014. An analysis of the precision and reliability of the leap motion sensor and its suitability for static and dynamic tracking. *Sensors,* 14(2), pp. 3702-3720.

Guo, F. et al., 2020. An adaptive wireless virtual reality framework in future wireless networks: A distributed learning approach. *IEEE Transactions on Vehicular Technology,* 69(8), pp. 8514-85288.

Haas, J., 2014. A history of the Unity game engine. *Diss. WORCESTER POLYTECHNIC INSTITUTE.*

Habgood, J. M., Wilson, D., Moore, D. & Alapont, S., 2017. *HCI lessons from PlayStation VR.* s.l., Extended abstracts publication of the annual symposium on computer-human interaction in play, pp. 125-135.

Harley, D., 2020. Palmer Luckey and the rise of contemporary virtual reality. *Convergence,* 26(5-6), pp. 1144-1158.

Heim, M. R., 2017. Virtual reality wave 3. In: *Boundaries of self and reality online.* s.l.:Elsevier, pp. 261-271.

Heller, A. J. & Benjamin, E. M., 2014. The ambisonic decoder toolbox: Extensions for partial-coverage loudspeaker arrays. In: *Linux Audio Conference.* s.l.:s.n.

Heller, A. J. & Benjamin, E. M., 2014. *The Ambisonics Decorder Toolbox: Extensions for Partial-Coverage Loudspeaker Arrays.* s.l., Linux Audio Conference.

Herre, J., Hilpert, J., Kuntz, A. & Plogsties, J., 2015. MPEG-H 3D audio-The new standard for coding of immersive spatial audio. *IEEE Journal of selected topics in signal processing,* 9(5), pp. 770-7799.

Hettinger, L. J. et al., 1990. Vection and Simulator Sickness. *Military Psychology,* 2(3), pp. 171-181.

Hoene, C., Patino Mejia, I. C. & Cacerovschi, A., 2017. MySofa- Design Your Personal HRTF. In: *Audio Engineering Convention 142.* s.l.:Audio Engineering Society.

Hofman, P. M., Van Riswick, J. G. & Van Opstal, J. A., 1998. Relearning sound localization with new ears. *Nature neuroscience,* 1(5), pp. 417-421.

Horowitz, S. & Looney, S., 2014. *The essential guide to game audio: the theory and practice of sound for games.* s.l.:Routledge.

Huang, J., Lucash, M. S., Scheller, R. M. & Klippel, A., 2021. Walking through the forests of the future: using data-driven virtual reality to visualize forests under climate change. *International Journal of Geographical Information Science,* 35(6), pp. 1155-1178.

ITU, 2015. *ITU-T Rec. BS.1116-3: Methods for the subjective assessment of small impairments in audio systems.* Geneva: Int. Telecomm. Union.

ITU, 2019. *ITU-R BS.1284-2: General methods for the subjective assessment of sound quality.* Geneva: Int. Telecomm. Union.

Jones, T., Moore, T. & Choo, J., 2016. The impact of virtual reality on chronic pain. *PloS one,* 11(12).

Kallinen, K. & Ravaja, N., 2007. Comparing speakers vs headphones in listening to news from a computer- individual differences and psychophysiological responses. *Computers in Human Behavior,* 23(1), pp. 303-317.

Kapralos, B. et al., 2015. Sound localization on tabletop computers: A comparison of two amplitude panning methods. *Computers in Entertainment (CIE),* 12(2), pp. 1-19.

kardong-Edgren, S. S., Farra, S. L., Alinier, G. & Young , M. H., 2019. A call to unify definitions of virtual reality. *Clinical Simulation in Nursing,* Volume 31, pp. 28-34.

Kardong-Edgren, S. S., Farra, S. L., Alinier, G. & Young, M. H., 2019. A Call to Unify Definitions of Virtual Reality. *Clinical Simulation in Nursing,* Volume 31, pp. 28-34.

Kavanagh, S., Luxton-Reilly, A., Wuensche, B. & Plimmer, B., 2017. A systematic review of Virtual Reality in Education. *Themes in Science and Technology Education,* 10(2), pp. 85-119.

Keller, A. et al., 2019. *Are we done with ray tracing?.* s.l., s.n.

Kenwright, B., 2018. Virtual Reality: Ethical challenges and dangers [opinion]. *IEEE Technology and Society Magazine,* 37(4), pp. 20-25.

Kim, J., Luu, W. & Palmisano, S., 2020. Multisensory integration and the experience of scene instability, presence and cybersickness in virtual environments. *Computers in Human Behavior,* Volume 113.

Kosa, M. & Uysal, A., 2020. Four pillars of health escapism in games: Emotion regulation, mood management, coping, and recovery. In: *Game user experience and player-centered design.* s.l.:Springer, Cham, pp. 63-76.

Kostadinov, D., Reiss, J. D. & Mladenov, V. M., 2010. *Evaluation of distance based amplitude panning for spatial audio.* s.l., ICASSP.

Kraemer, A., 2001. Two speakers are better than 5.1 [surround sound]. *IEEE Spectrum.*

Krueger, M. W., 1991. Artificial Reality II. *CUMINCAD.*

Kugler, L., 2021. The state of virtual reality hardware. *Communications of the ACM,* 62(2), pp. 15-16.

Lago, N. P. & Kon, F., 2004. The quest for low latency. In: *ICMC.* s.l.:s.n.

Lanier, J., 2017. *Dawn of the new everything: a journey through virtual reality.* s.l.:Random House.

Lanier, J., 2021. *The Metaverse: Expectations vs. Reality* [Interview] (11 11 2021).

LaValle, S. M., Yershova, A., Katsev, M. & Antonov, M., 2014. *Head tracking for the Oculus Rift.* s.l., 2014 IEEE International Conference on Robotics and Automation (ICRA).

Lee, L.-H.et al., 2021. All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda. *arXiv preprint.*

Leider, C., 2004. *Digital Audio Workstation.* s.l.:McGraw-Hill.

Lertzman, R., 2015. *Environmental melancholia: Psychoanalytic dimensions of engagement.* s.l.:Routledge.

Letowski, T. R. & Letowski, S. T., 2012. *Auditory spatial perception: Auditory localization,* s.l.: Army Research Lab Aberdeen Proving Ground MD Human Research and Engineering.

Li , S. & Peissig, J., 2020. Measurement of head-related transfer functions: A review. *Applied Sciences,* 10(14), p. 5014.

Liagkou, V., Salmas, D. & Stylios, C., 2019. Realizing virtual reality learning environment for industry 4.0. *Procedia CIRP,* Volume 79, pp. 712-717.

Liang, S., Zhong, X. & Yost, W., 2015. Dynamic binaural sound source localization with interaural time difference cues: Artificial listeners. *The Journal of the Acoustical Society of America,* 137(4).

Lieberman, A. J., Amir, O. & Schroeder, J., 2016. "Coming Alive" Through Headphones: Listening to Messages Via Headphones Vs. Speakers Increases Immersion, Presence, and Liking. *ACR North American Advances.*

Li, L. et al., 2017. Application of virtual reality technology in clinical medicine. *American journal of translational research,* 9(9), p. 3867.

Lim, H.-T., Herrscher, D., Waltl, M. J. & Chaari, F., 2012. Performance analysis of the IEEE 802.1 ethernet audio/video bridging standard. *SimuTools,* Volume 3, pp. 27-36.

Lopreiato, J. O. et al., 2016. *Healthcare Simulation Dictionary,* s.l.: http://www.ssih.org/dictionary.

Lossius, T., Baltazar, P. & de la Hogue, T., 2009. *DBAP - Distance-based amplitude panning.* s.l., ICMC.

Mabrook, R. & Singer, J. B., 2019. Virtual reality, 360 video, and journalism studies: conceptual approaches to immersive technologies. *Journalism studies ,* 20(14), pp. 2096-2112.

Majdak, P. et al., 2013. Spatially oriented format for acoustics: a data exchange format representing head-related transfer functions. In: *Audio Engineering Society Convention 134.* s.l.:Audio Engineering Society.

Maloney, D., Freeman, G. & Wohn, D. Y., 2020. "Talking without a voice" Understanding Non-verbal Communication in Social Virtual Reality. *Proceedings of the ACM on Human-Computer Interaction,* Volume 4, pp. 1-25.

Markowitz, D. M. & Bailenson, J. N., 2021. Virtual reality and the pyschology of climate change. *Current Opinion in Psychology.*

Martirosov, S. & Kopecek, P., 2017. Virtual reality and its influence on training and education-literature review. *Annals of DAAM & Proceedings,* Volume 28.

Masurovsky, A. et al., 2020. Controller-free hand tracking for grab-and-place tasks in immersive virtual reality: Design elements and their empirical study. *Multimodal Technologies and Interaction,* 4(4), p. 91.

Mazurek, J. et al., 2019. Virtual reality in medicine: a brief overview and future research directions. *Human Movement,* 20(3), pp. 16-22.

Microsoft, 2021. *Windows Sockets 2.* [Online]
Available at: https://docs.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2
[Accessed 01 07 2019].

Milgram, P., Takemura, H., Utsumi, A. & Kishino, F., 1995. Augmented reality: A class of displays on the reality-virtuality continuum. In: *Telemanipulator and telepresence technologies.* s.l.:International Society for Optics and Photonics, pp. 282-292.

Miller, H. L. & Bugnariu, N. L., 2016. Level of immersion in virtual environments impacts the ability to assess and teach social skills in autism spectrum disorder. *Cyberpsychology, Behaviour, and Social Networking,* 19(4), pp. 246-256.

Mishkind, M. C., Norr, A. M., Katz, A. C. & Reger, G. M., 2017. Review of virtual reality treatment in pyschiatry: evidence versus current diffusion and use. *Current psychiatry reports,* 19(11), pp. 1-8.

Moor, J. H., 2005. Why we need better ethics for emerging technologies. *Ethics and information technology,* 7(3), pp. 111-119.

Murray, J. H., 1999. *Hamlet on the holodeck: The future of narrative in cyberspace.* s.l.:MIT press.

Mütterlein, J., 2018. *The Three Pillars of Virtual Reality? Investigating the Roles of Immersion, Presence, and Interactivity.* s.l., Hawaii Internation Conference of System Sciences.

Mystakidis, S., 2022. Metaverse. *Encyclopedia,* 2(1), pp. 486-497.

Nacke, L. E. & Grimshaw, M., 2011. Player-Game Interaction Through Affective Sound. In: *Games Computing and Creative Technologies: Book Chapters.* Bolton: University of Bolton Institutional Repository, pp. 264-285.

Nalbant, G. & Bostan, B., 2006. *Interaction in Virtual Reality.* s.l., 4th International Symposium of Interactive Medial Design (ISIMD).

Narbutt, M. et al., 2018. AMBIQUAL- a full reference objective quality metric for ambisonic spatial audio. In: *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX).* s.l.:IEEE, pp. 1-6.

Nebeling, M. & Speicher, M., 2018. The trouble with augmented/virtual reality authoring tools. In: *2018 IEEE international symposium on mixed and augmented reality adjunct (ISMAR-Adjunct).* s.l.:IEEE, pp. 333-337.

Neumann, D. L. et al., 2018. A systematic review of the application of interactive virtual reality to sport. *Virtual Reality,* 22(3), pp. 183-198.

Newton, C., 2021. *Verge.* [Online]
Available at: https://www.theverge.com/22588022/mark-zuckerberg-facebook-ceo-metaverse-interview
[Accessed 16 8 2021].

Nicoll, B. & Keogh, B., 2019. The Unity game engine and the circuits of cultural software. In: *The Unity game engine and the circuits of cultural software.* s.l.:Springer, pp. 1-21.

Nieborg, D. B., 2005. In: *Creative Gamers Seminar- Exploring Participatory Culture in Gaming.* Tampere, Finland: University of Finland.

Oculus, 2016. *Oculus rift pre-orders now open, first shipments March 28.* [Online]
Available at: https://www.oculus.com/blog/oculus-rift-pre-orders-now-open-first-shipments-march-28/
[Accessed 25 May 2018].

Oculus, 2021. *Audio SDK.* [Online]
Available at: https://developer.oculus.com/documentation/unity/unity-audio/
[Accessed 14 08 2021].

Oh, Y. & Yang, S., 2010. Defining exergames & exergaming. *Proceedings of meaningful play,* Volume 2010, pp. 21-23.

Park, B. J. et al., 2020. Augmented and mixed reality: technologies for enhancing the future of IR. *Journal of Vascular and Interventional Radiology,* 31(7), pp. 1074-1082.

Pasnau, R., 1999. What is sound?. *The Philosophical Quarterly,* 49(196), pp. 309-324.

Paul, C., 2003. *Digital Art.* s.l.:London: Thames and Hudson.

Peckham, M., . The Inside Story of Oculus Rift and How Virtual Reality Became Reality. *Wired,* , (), p. .

Peek, E., Wünsche, B. & Lutteroth, C., 2013. Virtual reality capabilities of graphics engines.

Peterson, n.d. Virtual Reality, Augmented Reality, and Mixed Reality Definitions. *EMA.*

Piechowicz, J., 2011. Sound Wave Diffraction at the Edge of a Sound Barrier. *Acta Physica Polonica, A,* Volume 119.

Polančec, D. & Mekterović, I., 2017. *Developing MOBA games using the Unity game engine.* s.l., IEEE, pp. 1510-1515.

Pottle, J., 2019. Virtual reality and the transformation of medical education. *Future healthcare journal,* 6(3), p. 181.

Pueo, B., Rico, J. V. & Lopes, J. J., 2011. Vibration Analysis of Edge and Middle Exiters in Multiactuator Panels. In: *Audio Engineering Convention 131.* s.l.:Audio Engineering Society.

Pulkki, V., 1997. Virtual sound source positioning using vector based amplitude panning. *journal of the audio engineering society,* 45(6), pp. 456-466.

Pulkki, V., 2000. *Generic panning tools for MAX/MSP.* s.l., ICMC.

Pulkki, V. & Karjalainen, M., 2008. Multichannel audio rendering using amplitude panning [dsp applications]. *IEEE Signal Processing Magazine,* 25(3), pp. 118-122.

Radianti, J., Majchrzak, T. A., Fromm, J. & Wohlgenannt, I., 2020. A systematic review of immersive virtual reality applications for higher education, design elements, lessons learned, and research agenda. *Computers & Education,* Volume 147, p. 103778.

Rajguru, C., Obrist, M. & Memoli, G., 2020. Spatial soundscapes and virtual worlds: Challenges and oppurtunities. *Frontiers in Psychology,* Volume 11, p. 2714.

Roberts, B., 2021. *whathifi.com.* [Online]
Available at: https://www.whathifi.com/advice/dolby-atmos-what-it-how-can-you-get-it
[Accessed 9 10 2021].

Robinson, C., 2019. *Game Audio with FMOD and Unity.* s.l.:Routledge.

Rolland, J. P. & Hong, H., 2005. Head-Mounted Display Systems. In: *Encyclopedia of Optical Engineering 2.* s.l.:s.n.

Rumsey, F., 2012. *Spatial Audio.* s.l.:Routledge.

Saffo, D., Yildirim, C., Di Bartolomeo, S. & Dunne, C., 2020. *Crowdsourcing virtual reality experiments using vrchat.* s.l., s.n., pp. 1-8.

Schacter, D., Gilbert, D., Wegner, D. & Hood, B., 2011. *Pyschology.* European Edition ed. s.l.:Worth Publishers.

Schiavoni, F. L. & Gonçalves, L. L., 2017. *From virtual reality to digital arts with mosaicode.* s.l., IEEE, pp. 200-206.

Schoeffler, M., Silzle, A. & Herre, J., 2016. Evaluation of spatial/3D audio: Basic audio quality versus quality of experience. *IEEE Journal of Selected Topics in Signal Processing,* 11(1), pp. 75-88.

Schroeder, R., 1993. Virtual Reality in the Real World. *Futures,* 25(9), pp. 963-973.

Schutte, N. S. & Stilinović, E. J., 2017. Facilitating empathy through virtual reality. *Motivation and emotion,* 41(6), pp. 708-712.

Schütze, S. & Irwin-Schütze, A., 2018. *New Realities in Audio: A Practical Guide for VR, AR, MR and 360 Video.* Boca Raton: CRC Press.

Sevinc, V. & Berkman, M. I., 2020. Psychometric evaluation of Simulator Sickness Questionnaire and its variants as a measure of cybersickness in consumer virtual environments. *Applied Ergonomics,* Volume 82.

Shalf, J., 2020. The future of computing beyond Moore's law. *Philosophical Transactions of the Royal Society A,* Volume 378.

Sherman, W. R. & Craig, A. B., 2002. Understanding Virtual Reality: Interface, application, and design. *Elsevier.*

Siani, A. & Marley, S. A., 2021. Impact of the recreational use of virtual reality on physical and mental wellbeing during the Covid-199 lockdown. *Health and Technology,* 11(2), pp. 425-435.

Simon, L. S., Wuethrich, H. & Dillier, N., 2017. Comparison of Higher-Order Ambisonics, Vector-and Distance-Based Amplitude Panning Using a hearing device beamformer.

Sinclair, J.-L., 2020. *Principles of Game Audio and Sound Design: Sound Design and Audio Implementation for Interactive and Immersive Media.* s.l.:CRC Press.

Slater, M., 2018. Immersion and the illusion of presence in virtual reality. *British journal of pyschology,* 109(3), pp. 431-433.

Somberg, G., 2016. *Game audio programming: principles and practices.* s.l.:Taylor & Francis, CRC Press.

Statistica, 2018. *Forecast augmented (ar) and virtual reality (vr) market size worldwide from 2016 to 2021 (in billion u.s. dollars).* [Online]
Available at: https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/ [Accessed 15 May 2018].

Stauffert, J.-P., Niebling, F. & Latoschik, M. E., 2018. *Effects of Latency Jitter on Simulator Sickness in a Search Task.* s.l., IEEE, pp. 121-127.

Steinberg, 2019. *ASIO 2.3 Streaming Input Output Specification.* s.l.:Steinberg Media Technologies GmbH.

Stenzel, H. & Jackson, P. J., 2018. Perceptual thresholds of audio-visual spatial coherence for a variety of audio-visual objects. In: *Audio Engineering Society Conference: 2018 AES internation conference for virual and augmented reality.* s.l.:Audio Engineering Society.

Steuer, J., 1992. Defining Virtual Reality: Dimensions determining telepresence. *Journal of communication,* pp. 73-93.

Sun, X., 2019. Immersive audio, capture, transport, and rendering: a review. *APSIPA Transactions on Signal and Information Processing,* Volume 10.

Sutherland, I., 1965. The Ultimate Display. *International Federation of Information Processing Congress ,* 62(2), pp. 506-508.

Sutherland, I., 1968. *A Head-Mounted Three Dimensional Display.* s.l., Fall Joint Computer Conference.

Suvorov, R., 2009. Context visuals in L2 listening tests: The effects of photographs and video vs. audio-only format. *Developing and evaluating language learning materials,* pp. 53-68.

Szpak, A., Michalski, S. C. & Loetscher, T., 2020. Exergaming with beat saber: an investigation of virtual reality aftereffects. *Journal of Medical Internet Research,* 22(10).

Tarzan, A., Alunno, M. & Bientinesi, P., 2019. Assessment of sound spatialization algorithms for sonic rendering with headphones. *Journal of New Music Research,* 48(2), pp. 107-124.

Therrien, C., 2015. Inspecting video game historiography through critical lens: Etymology of the first-person shooter genre. *Game Studies,* 15(2).

Thorn, A., 2013. *Unity 4 fundamentals.* s.l.:CRC Press.

Ting, T. M. et al., 2021. Binaural Modelling and Spatial Auditory Cue Analysis of 3d-Printed Ears. *Sensors,* Volume 21, p. 227.

Toftedahl, M. & Engström, H., 2019. *A taxonomy of game engines and the tools that drive the industry.* s.l., Digital Games Research Association (DiGRA).

Tuan, Y.-F., 2000. *Escapism.* s.l.:JHU Press.

Tuena, C. et al., 2020. Usability issues of clinical and research applications of virtual reality in older people: a systematic review. *Frontiers in human neuroscience,* Volume 14, p. 93.

Turner, C. J., Hutabarat, W., Oyekan, J. & Tiwari, A., 2016. Discrete event simulation and virtual reality use in industry: new opportunities and future trends. *IEEE Transactions on Human-Machine Systems,* 46(6), pp. 882-894.

Tussyadiah, I. P., Wang, D., Jung, T. H. & tom Dieck, C. M., 2018. Virtual reality, presence, and attitude change: Empirical evidence from tourism. *Tourism Management,* Volume 66, pp. 140-154.

Unity Technologies, 2021. *https://docs.unity3d.com/ScriptReference/index.html.* [Online]
Available at: https://docs.unity3d.com/ScriptReference/index.html
[Accessed 20 11 2021].

Unity, 2021. *Made with Unity.* [Online]
Available at: https://unity.com/madewith
[Accessed 10 January 2017].

Valve, 2020. *Half-Life Alyx.* [Online]
Available at: https://www.half-life.com/en/alyx/
[Accessed 1st June 2021].

Van Dam, A., 1997. Post-WIMP user interfaces. *Communications of the ACM,* 40(2), pp. 63-67.

Västfjäll, D., 2003. The subjective sense of presence, emotion recognition, and experienced emotions in auditory virtual environments. *CyberPsychology & Behavior,* 6(2), pp. 181-188.

Venkataraman, A. & Jagadeesha, K. K., 2015. Evaluation of inter-process communication mechanisms. *Architecture,* Volume 86, p. 64.

Viola, P. & Jones, M., 2001. Robust real-time object detection. *International journal of computer vision,* 4(4), pp. 34-47.

Voigt-Antons, J.-N., Kojic, T., Ali, D. & Möller, S., 2020. Influence of hand tracking as a way of interaction in virtual reality on user experience. *Twelfth International Conference on Quality of Multimedia Experience (QoMEX),* pp. 1-4.

Wachowski, L. & Wachowski, L., 1999. *The Matrix.* California, USA, Warner Bros.

Wagner, E., 2010. The effect of the use of video texts on ESL listening test-taker performance. *Language testing,* 27(4), pp. 493-513.

Wang, M., 2020. *Social VR: A New Form of Social Communication in the Future or a Beautiful Illusion?.* s.l., Journal of Physics: Conference Series.

Watson, B. et al., 2021. *Esports and High Performance HCI.* s.l., Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-5.

Weichert, F., Bachmann, D., Rudak, B. & Fisseler, D., 2013. Analysis of the accuracy and robustness of the leap motion controller. *Sensors,* 13(2), pp. 6380-6393.

Wells, J. R. & Winkler, C. A., 2019. *Facebook fake news in the post-truth world.* s.l.:Harvard Business Publishing Education, September 24.

Woodcock, J., Davies, W. J., Cox, T. J. & others, 2019. Influence of visual stimuli on perceptual attributes of spatial audio. *Journal of the Audio Engineering Society,* 67(7), pp. 557-567.

World Health Organization, 2006. *Primary ear and hearing care training resource.* s.l.:World Health Organization.

Yang, L. I. et al., 2019. Gesture interaction in virtual reality. *`Virtual Reality & Intelligent Hardware,* 1(1), pp. 84-112.

Yao, S.-N., Collins, T. & Jančovič, P., 2015. Timbral and spatial fidelity improvement in ambisonics. *Applied Acoustics,* Volume 93, pp. 1-8.

Yeung, A. W. K. et al., 2021. Virtual and augmented reality applications in medicine: analysis of scientific literature. *Journal of Medical Internet Research,* 23(2).

Yılmaz, M., Yılmaz, U. & Demir Yılmaz, E. N., 2019. The relation between social learning and visual culture. *International Electronic Journal of Elementary Education,* 11(4), pp. 421-427.

Yost, W. A., 2018. Auditory motion parallax. *Proceedings of the National Academy of Sciences,* 115(16), pp. 3998-4000.

Zackariasson, P. & Wilson, T., 2012. *The video game industry: Formation, present state, and future.* New York: Routledge.

Zhang, H., 2017. Head-mounted display-based intuitive virtual reality training system for the mining industry. *International Journal of Mining Science and Technology,* 27(4), pp. 717-722.

Zhang, M. et al., 2009. HRTF measurement on KEMAR manikin. In: *Proceedings of Acoustics.* s.l.:s.n.

Zielinski, S., Rumsey, F. & Bech, S., 2008. On some biases encountered in modern audio quality listening tests- a review. *Journal of the Audio Engineering Society,* 56(6), pp. 427-451.

Zotter, F., Pomberger, H. & Noisternig, M., 2010. *Ambisonics decoding with and without mode-matching: A case study using the hemisphere.* s.l., Proceedings of the 2nd International Symposium on Ambisonics and Spherical Acoustics.

APPENDIX A

The sections in this appendix provide listings of the functions that were used in the provision of 3D coordinates of game objects (audio source) to the modified ImmerGo server. As such, they inform the coordinate transmission component of the immersive audio capability that has been developed, and also provide the context to how the FMOD Unity integration package was leveraged in custom scripts to initialize FMOD audio events from code and bind ImDSP instances to these events.

The codebase of this project is available for further inspection should the following listings not provide enough context.

A1 TRANSMITTING COORDINATES

To send Game Object, and therefore Audio Source Coordinates, the *SendCoords* coroutine was employed. This coroutine is executed from the *Start* function in the *SocketIOClient* script that was attached to the player controller object in the scene. Listing A1 also indicates the data curation function call (*PercentileConversion*), and the population of a JSONObject with coordinate data, which is transmitted to the ImmerGo server.

```
private IEnumerator SendCoords()//A Co-Routine that transmits coordinates of audio sources to server
every ~10msec
    {
        //Swapped 'z' and 'y' coords just before sending, as 'z' represents up and down in ImmerGo
server
        Vector3 posa1 = new Vector3(0, 0, 0);

        Boolean condition = true;
        while (condition)
        {
            yield return new WaitForSeconds(0.10F); //Equivalent to ticks on mobile client; coords
                                                    transmitted every 10msecs.


            //Audio Source 1
            Vector3 temp1 = posa1;
            posa1 = source1.transform.localPosition;

            if (posa1 != temp1)
            {

                if (recording) { trackedCoordsA1.Add(posa1); tracks[0].recordState = true;
                } else
                {
                     tracks[0].recordState = true;
                }

                Vector3 percentPos = PercentileConversion(posa1);

                Dictionary<string, string> data1 = new Dictionary<string, string>();
                data1["x"] = System.Convert.ToString(percentPos.x);//ax
                data1["y"] = System.Convert.ToString(percentPos.y);//ay
                data1["z"] = System.Convert.ToString(percentPos.z);
                data1["spread"] = System.Convert.ToString(0);
                data1["trackNo"] = System.Convert.ToString(0);

                JSONObject bob = new JSONObject(data1);
                socket.Emit("coordsvr", new JSONObject(data1));
            }
```

```
            yield return new WaitForSeconds(0.10f);
```

Listing A1: The SendCoords cooroutine that detects whether the GameObject (Audio Source) has changed position. The logic is provided for Audio Source 1 (Planet Earth) although the same sequence would apply to all eight sources in the scene, but has been omitted for brevity.

A2 Audio Event Playback

The following listing represents a coroutine that is called in scrips for both speaker and headphone environments. The coroutine was used to launch all eight audio sources at five second intervals when the application was launched.

```
    IEnumerator LaunchEvents(FMOD.Studio.EventInstance[] instances)
    {
        for (int i = 0; i < 8; i++)
        {
            instances[i].start();
            yield return new WaitForSeconds(5);
        }
    }
```

Listing A2: Launching each FMOD Audio Event.

A3 Soloing Audio Sources

The experimenter was able to solo audio sources for the subjective audio listening tests by using a control script for either environment (*CloseApp* and *AudioController* scripts). As can be seen in Listing A3, the FMOD audio event instance has its volume raised, while the remaining events are muted. This process was replicated for all audio events and bound to different key presses. In this instance, the *EarthInst* function is shown, and is triggered by the ‘e’ key.

```
        //Earth
        if (Input.GetKey("e"))
        {
            //solo volume up
            EarthInst.setVolume(0.5f);
            EarthInst.setMute(false);
            JupInst.setMute(true);
            MercInst.setMute(true);
            NeptInst.setMute(true);
            SatInst.setMute(true);
            UranInst.setMute(true);
            VenusInst.setMute(true);
            MarsInst.setMute(true);
        }
```

Listing A3: Solo control of audio sources for testing in the control scripts.

APPENDIX B

The code-listings that follow all pertain to the custom FMOD DSP plugin that was implemented and attached to all audio sources within the Unity-authored speaker environment scene of the immersive VR application. This DSP plugin (ImDSP) forms another critical component of the immersive audio capability that was developed.

The codebase of this project is available for further inspection should the following listings not provide enough context.

B1 IMDSP DESCRIPTION

Listing B1A indicates the ImDSP description object which conforms to the FMOD_DSP_DESCRIPTION struct. The relevant callbacks of the ImDSP plugin are indicated, as well as a single integer parameter representing the trackID/eventID. This description enables the FMOD System to recognize which callbacks the plugin provides, and the number of parameters it employs.

```
FMOD_DSP_DESCRIPTION FMOD_ImDSP_Desc =
{
        FMOD_PLUGIN_SDK_VERSION,
        "FMOD ImmerGO DSP plugin",                  // name
        0x00010000,                                 // plugin version
        1,                                          // number of input buffers to process
        1,                                          // number of output buffers to process
        FMOD_ImDSP_dspcreate,
        FMOD_ImDSP_dsprelease,
        FMOD_ImDSP_dspreset,
#ifndef FMOD_IMDSP_USEPROCESSCALLBACK
        FMOD_ImDSP_dspread,
#else
        0,
#endif
#ifdef FMOD_IMDSP_USEPROCESSCALLBACK
        FMOD_ImDSP_dspprocess,  // *** declare this callback instead of FMOD_ImDSP_dspread if the
plugin sets the output channel count ***
#else
        0,
#endif
        0, //setposition
        FMOD_ImDSP_NUM_PARAMETERS, //FMOD_ImDSP_NUM_PARAMETERS,
        FMOD_ImDSP_dspparam, //FMOD_ImDSP_PARAMETER_DESC
        0, //FMOD_ImDSP_dspsetparamfloat,
        FMOD_ImDSP_dspsetparamint, // FMOD_ImDSP_dspsetparamint,
    0, // FMOD_ImDSP_dspsetparambool,
        0, // FMOD_ImDSP_dspsetparamdata,
        0, //FMOD_ImDSP_dspgetparamfloat,
        FMOD_ImDSP_dspgetparamint, // FMOD_ImDSP_dspgetparamint,
        0, // FMOD_ImDSP_dspgetparambool,
        0, // FMOD_ImDSP_dspgetparamdata,
        FMOD_ImDSP_shouldiprocess,
        0, // userdata
        0, // sys_register
        0, // sys_deregister
    0  // sys_mix
};
```

Listing B1: The FMOD_ImDSP_Desc object that adheres to the FMOD_DSP_DESCRIPTION struct requirements.

B2 Creating and Releasing an ImDSP instance

The create callback is called by the FMOD System when an event instance with the DSP plugin is first created from within Unity. In this instance, the callback performs three tasks. The first is to provide a pointer to a TrackID variable (in C++) that can be assigned and referenced from within Unity in C# (see section 6.2.3.2.2). The second task is to call the *winsockInitialize* function which initializes both receive and send sockets and assigns the ImDSP instance as a client within a Multicast group. The final task of the create callback is to allocate memory for the *FMODImDSPState* object which is detailed in the previous section.

The release callback mirrors the functionality of the create callback except that it is concerned with the deallocation of the socket variables and *FMODImDSPState* object. It is called when the FMOD System releases the plugin once an audio event has finished playback.

```
//CREATE CALLBACK
FMOD_RESULT F_CALLBACK FMOD_ImDSP_dspcreate(FMOD_DSP_STATE *dsp_state)
{
        TrackIDPtrPtr = (void**)CoTaskMemAlloc(sizeof(int));

        //Socket Initalization function which creates the Receive and Send sockets.
        winsockInitialize();

        //Allocation of FMODImDSPState object as per the FMOD_DSP_DESCRIPTION Struct.
        dsp_state->plugindata = (FMODImDSPState *)FMOD_DSP_ALLOC(dsp_state, sizeof(FMODImDSPState));
        if (!dsp_state->plugindata)
        {
                return FMOD_ERR_MEMORY;
        }
        return FMOD_OK;
}

//RELEASE CALLBACK
FMOD_RESULT F_CALLBACK FMOD_ImDSP_dsprelease(FMOD_DSP_STATE *dsp_state)
{
        //Socket Deallocation function which releases Receive and Send sockets.
        winsockFree();

        //FMODImDSPState Deallocation function.
        FMODImDSPState *state = (FMODImDSPState *)dsp_state->plugindata;
        FMOD_DSP_FREE(dsp_state, state);
        return FMOD_OK;
}
```

Listing B2: Both the Create and Release callbacks for the ImDSP plugin

B3 ImDSP Process Function

The process function is indicated in Listing B3 and is used to populate a *SampleBuffer* array. This array is set to the buffer length that has been decided by developers—in this instance, 512, using a macro at the top of the file. The function determines how many samples to expect, which in this case will always be 512 due to the audio track of the FMOD audio event being a monophonic signal. If the signal was a multiple channel signal, then the total samples for the audio track would be the number of channels multiplied by the length of the buffer (512). In this listing, the relevant downmixing of multi-channel audio is excluded.

Once the total samples variable has been determined, a bit mask is applied to each of them, and the resultant sample is placed within the *SampleBuffer* array. This bit mask is used to ensure that the

resultant audio sample is compatible with the type of audio sample required by the ASIO playback server.

```
//"inbuffer" = pointer to in buffer, "outbuffer" = pointer to out buffer, "length" = length of incoming
and outgoing buffer in samples, "channels" = number of in buffer channels (encoded in .wav file)
FMOD_RESULT FMODImDSPState::process(float *inbuffer, float *outbuffer, unsigned int length, int
channels)
{
        FMOD_SOUND_FORMAT format = FMOD_SOUND_FORMAT_PCM32;

        //NEED TO ACCOMODATE FOR SOUNDS WITH MULTIPLE CHANNELS (stereo, quad, etc).
        memFlag = true;
        unsigned int totsamples = length * channels;
        int value = 0;
        //0x7fffffff a bit mask: the mask has all bits of a 32-bit integer set, except the signed bit.
        double sc = 0x7fffffffL;
        for (int i = 0; i < totsamples; i++)
        {
                if (i % channels == 0)
                {
                        SampleBuffer[value] = inbuffer[i+1] * (0x7fffffff);
                        value++;
                }
                outbuffer[i] = inbuffer[i];
        }

        return FMOD_OK;
}
```

Listing B3: Process function which is called from the Process Callback. The function populates the SampleBuffer array which is used in the transmission of the audio packets to the playback server.

B4 IMDSP PROCESS CALLBACK

The process callback is called for every FMOD Mixer update. After extracting the plugin data, and determining whether the in and out buffers exist, the function calls the *process* function (see Listing B3). Once the *process* function has populated the *SampleBuffer* array with audio samples obtained from the **inbuffer*, the callback then proceeds to extract the trackID from the *getuserdata* function, which entails dereferencing pointers. Once the trackID has been determined to be the valid track, the *SendSamples* function is then called (see Listing B5), and the callback returns information to the Low Level System with an FMOD_OK result.

```
FMOD_RESULT F_CALLBACK FMOD_ImDSP_dspprocess(FMOD_DSP_STATE *dsp_state, unsigned int length, const
FMOD_DSP_BUFFER_ARRAY *inbufferarray, FMOD_DSP_BUFFER_ARRAY *outbufferarray, FMOD_BOOL inputsidle,
FMOD_DSP_PROCESS_OPERATION op)
{
        FMODImDSPState *state = (FMODImDSPState *)dsp_state->plugindata;
        if (op == FMOD_DSP_PROCESS_QUERY)
        {
                if (outbufferarray && inbufferarray)
                {
                        outbufferarray[0].bufferchannelmask[0] = inbufferarray[0].bufferchannelmask[0];
                        outbufferarray[0].buffernumchannels[0] = inbufferarray[0].buffernumchannels[0];
                        outbufferarray[0].speakermode = inbufferarray[0].speakermode;
                }

                if (inputsidle)
                {
                        return FMOD_ERR_DSP_DONTPROCESS;
                }
```

```
        }
        else
        {
                state->process(inbufferarray[0].buffers[0], outbufferarray[0].buffers[0], length,
inbufferarray[0].buffernumchannels[0]); // input and output channels count match for this effect


//======== USING getuserdata TO GET CLIENT/EVENT/TRACK ID==========
                dsp_state->functions->getuserdata(dsp_state, TrackIDPtrPtr);
                trackIDPtr = (int*)*TrackIDPtrPtr;
                trackID = (int)*trackIDPtr;
                if (trackID >= 0 && trackID < 33) //Make Sure a valid trackID is selected
                {
                        sendSamples(SampleBuffer, trackID);
                }
        }

        return FMOD_OK;
}
```

Listing B4: The process callback used by the ImDSP plugin for processing Audio Samples to which the ImDSP instance is attached.

B5 SendSamples function and Packet Population

The packet object indicated in Listing B5 below is defined with three members: a c*lientID*, *packNum*, and the audio samples of type int32_t. This function is called from the process callback and is used to populate two packet objects with audio samples. The function waits for the "SendSamples" message to be transmitted from the multicast server on the *recvfrom* socket. After receiving this message, it populates the packets with audio samples that are provided in a parameter to the function.

```
//Struct Implementation Schematic
struct packet
{
        int clientID;
        int packNum;
        int32_t Samples[DEFAULT_BUFLEN/2];
};

//Send Samples Function
void sendSamples(int32_t* samples, int identifier)
{
        iResult = 100;
        // Receive until the peer closes the connection
        // Use this loop to continuously send sound samples to be played out on server side, until a
stop request is sent.
        int RecvLen = sizeof(recvSock);

        packet One;
        packet Two;
        if (iResult > 0) {
                //RECIEVE MESSAGE FROM SERVER REQUESTING FOR SAMPLES
                iResult = recvfrom(MultiSocket, requestBuf, requestBufLen, 0, (struct sockaddr
                FAR*)&recvSock, &RecvLen);
                One.clientID = identifier;
                Two.clientID = identifier;
                One.packNum = 1;
                Two.packNum = 2;

                int holder = 0;

                for (int i = 0; i < DEFAULT_BUFLEN; i++)
                {
                        //DEFAULTBUFLEN/2 SOLUTION (2 packets being sent)
```

```
                        if (i < (DEFAULT_BUFLEN / 2))
                        {
                                One.Samples[i] = samples[i];
                        }
                        else if (i >= (DEFAULT_BUFLEN / 2) && i < (DEFAULT_BUFLEN))
                        {
                                Two.Samples[holder] = samples[i];
                                holder2++;
                        }
                }

                //MAX SAMPLE SEND
                iResult = sendto(ListenSocket, (char*)&One, sizeof(packet), 0, (SOCKADDR *)&serverAddr,
                sizeof(serverAddr));
                iResult = sendto(ListenSocket, (char*)&Two, sizeof(packet), 0, (SOCKADDR *)&serverAddr,
                sizeof(serverAddr));
        }
}
```

Listing B5: The Packet Struct and SendSamples function.

APPENDIX C

This appendix will provide the relevant code listings pertaining to the playback server, which forms the third and final component of the immersive audio capability. The ASIO thread implementation, which is detailed in section 5.3.3.2, is excluded from these listings.

The codebase of this project is available for further inspection should the following listings not provide enough context.

C1 ASIO SET UP

The receiver thread for the playback server is detailed in section 5.3.3.1, and the following listing indicates the various ASIO requirements before transmission and reception of messages is possible. The *if* statement in the listing indicates all the states that the ASIO driver needs to adhere to before being finally initialized successfully with the *ASIOStart* call. The states of the driver can be seen in Figure 74.

Once the driver has been successfully started and is in the *Running* state, a loop is launched that will execute as long as the driver has not been stopped, and therefore as long as it remains in the *Running* state. The contents of this loop are indicated in Listing C2. This stop condition can only be met if the ASIO Thread (see section 5.3.3.2) stops the driver.

```
// load the driver, this will setup all the necessary internal data structures
        if (loadAsioDriver(ASIO_DRIVER_NAME))
        {
                // initialize the driver
                if (ASIOInit(&asioDriverInfo.driverInfo) == ASE_OK)
                {
                        printf("\nasioVersion:   %d\n"
                                "driverVersion: %d\n"
                                "Name:          %s\n"
                                "ErrorMessage:  %s\n",
                                asioDriverInfo.driverInfo.asioVersion,
                                asioDriverInfo.driverInfo.driverVersion,
                                asioDriverInfo.driverInfo.name,
                                asioDriverInfo.driverInfo.errorMessage);

                        //After this method has been called, 'holder' is a pointer to an array of
                          integers holding the samples for the .wav file

                        if (init_asio_static_data(&asioDriverInfo) == 0)
                        {

                                // set up the asioCallback structure and create the ASIO data buffer
                                asioCallbacks.bufferSwitch = &bufferSwitch;
                                asioCallbacks.sampleRateDidChange = &sampleRateChanged;
                                asioCallbacks.asioMessage = &asioMessages;
                                asioCallbacks.bufferSwitchTimeInfo = &bufferSwitchTimeInfo;
                                int counter = 0;
                                if (create_asio_buffers(&asioDriverInfo) == ASE_OK)
                                {
                                        if (ASIOStart() == ASE_OK)
                                        {
                                                // Now all is up and running
                                                fprintf(stdout, "\nASIO Driver started
                                                successfully.\n\n");

                                                //Temporary Int array
                                                int tempor[DEFAULT_BUFLEN] = { 0 };
```

```
                                        //Useful for keeping track of threads, etc.
                                        while (!asioDriverInfo.stopped)
```

Listing C1: The listing indicates the ASIO prerequisites that the playback server needs to adhere to before entering the loop waiting to receive samples.

C2 Providing ASIO Driver with Audio Samples

The following listing should be seen as an extension of the previous listing, as the conditional loop with which it begins is the one that is indicated at the end of Listing C1.

The receiver thread functions by sending a request for samples to all multicast clients, and then by receiving audio samples from each client indicating that the request has been successful.

Once the audio samples have been received, an inspection of whether they have originated from the same client (through the identifier that is transmitted with the audio samples) is provided. If the arriving packets are from the same client, an appropriate switch statement determines which client the samples are from and populates the appropriate circular buffer. This is accomplished by the *PopulateOnceCircular* function, which is detailed in the following listing.

For the sake of brevity, the switch statement which is viewable applies to only one client/track/eventID. The other cases have been excluded.

```
while (!asioDriverInfo.stopped)                 //Carry on sending samples until ASIO device is closed
(drivers are stopped)
{
        if (indicator) {
                if (enumerator == true)
                                {

                                iResult = sendto(MultiSocket, requestBuf, requestbufflen, 0,
                                (SOCKADDR*)&recvSock, sizeof(recvSock));
        //Moved Sendto() into loop (as each client will require a seperate sendto() call for each set
        of samples
                                packet recvPacketOne;
                                packet recvPacketTwo;
                                iResult = recvfrom(ListenSocket, (char*)&recvPacketOne,
                                sizeof(recvPacketOne), 0, (SOCKADDR*)&client, &clilen);
                                iResult = recvfrom(ListenSocket, (char*)&recvPacketTwo,
                                sizeof(recvPacketTwo), 0, (SOCKADDR*)&client, &clilen);

                                if (recvPacketOne.clientID == recvPacketTwo.clientID
                                        {
                                              clientNum = recvPacketOne.clientID;
                                              switch (clientNum)
                                              {
                                              case 1:
        //Populate the requisite circular buffer according to the client number and with the two sets
        of audio samples
        PopulateOnceCircular(clientNum, recvPacketOne.Samples, recvPacketTwo.Samples, tempor);
        RecCli1.put(tempor, clientNum);

                                            //ALL OTHER CASES HERE (FOR EACH ACTIVE OUTPUT: 8 OUTPUTS
                                              IN THIS CASE

                                              }
                                         }
                                }
                        }//while
```

Listing C2: An overview of the receiver thread.

C3 Setting Sample Rate

The sample rate of the ASIO Streamware driver was set to 48000 by obtaining the *asioDriverInfo* object and setting its referenced *sampleRate* parameter. Listing C3 indicates how this was achieved and is an excerpt from the *init_asio_static_data* function.

```
// get the usable buffer sizes
                if (ASIOGetBufferSize(&asioDriverInfo->minSize, &asioDriverInfo->maxSize,
&asioDriverInfo->preferredSize, &asioDriverInfo->granularity) == ASE_OK)
                {
                        printf("ASIOGetBufferSize (min: %d, max: %d, preferred: %d, granularity:
%d);\n",
                                asioDriverInfo->minSize, asioDriverInfo->maxSize,
                                asioDriverInfo->preferredSize, asioDriverInfo->granularity);

                        // get the currently selected sample rate
                        if (ASIOGetSampleRate(&asioDriverInfo->sampleRate) == ASE_OK)
                        {

                                if (ASIO_DRIVER_NAME == "ASIO4ALL v2")
                                {
                                        asioDriverInfo->sampleRate = 44100.0;           //ASIO4ALL
                                }
                                else
                                {
                                        asioDriverInfo->sampleRate = 48000.0;           //Streamware
                                }
//Continued…
```

Listing C3: Setting the ASIO driver sample rate.

The following three sections indicate the functions that were developed to facilitate the implementation of the circular buffer data structure for storing audio samples.

C4 Circular Buffer Population

The function for populating the circular buffers is indicated in the following listing. As shown, the function takes in four parameters: the first of these is the identifier for the audio event, the second and third are pointers to the first sample in the audio sample arrays that arrived in the first and second packet (respectively) transmitted by the ImDSP instance (multicast client), and the final parameter is a temporary audio sample array with the size of the buffer length (512).

The function simply populates the temporary array with the audio samples that are supplied by both packets. This is performed in the loop indicated in the listing.

```
//CIRCULAR BUFFER POPULATE FUNCTION
int* PopulateOnceCircular(int clientID, int* One, int * Two, int * temp)
{
        int holder = 0;

        for (int i = 0; i < DEFAULT_BUFLEN; i++)
        {
                //TWO PACKETS
                if (i < DEFAULT_BUFLEN/2)
                {
```

```
                        temp[i] = One[i];
                }
                else if (i >= (DEFAULT_BUFLEN / 2) && i < DEFAULT_BUFLEN)
                {
                        temp[i] = Two[holder];
                        holder++;
                }
        }
        return temp;
}
```

Listing C4: The CircularBufferPopulation function.

C5 Circular Buffer Class

As has been mentioned, a circular buffer implementation was provided to store audio samples transmitted from ImDSP instances (multicast clients), and to provide these samples to the correct audio output channel of the ASIO Streamware driver. Section 5.3.3 indicates an overview of these processes.

The following figure presents a schematic diagram of how circular buffer implementations "look" and function.

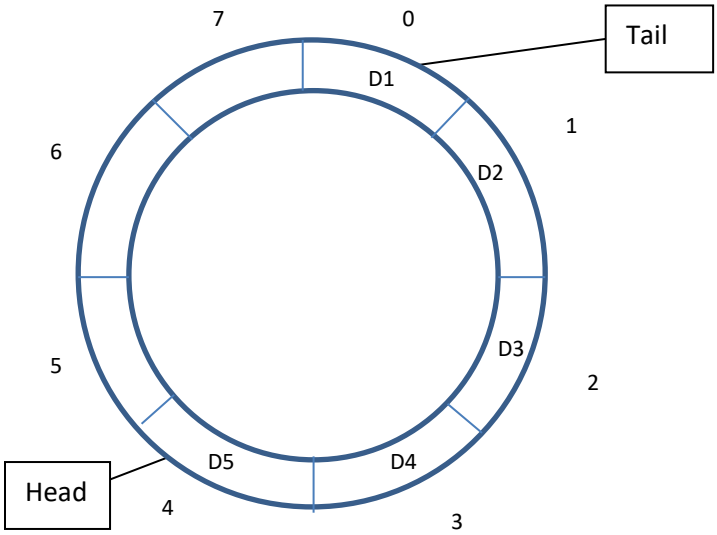

Figure 134: A diagram representing the elements of a circular buffer, and how this type of data structure functions. The outer numbers reference the positions (indexes) of the array, whilst the Dx variables represent the data, of size 512 samples in this implementation, stored at these positions.

The public variables are provided first, and the *head* and *tail* integers are used to keep track of where the head and tail point to in the array. Of course, these variables also enable the developer to determine whether the buffer is empty or full, and where sets (frames) of audio samples should be placed or taken from.

The *put* function is used to place audio samples into the buffer at the appropriate location. The *get* function is used to provide audio samples to the ASIO driver, and will only do so if the buffer is not empty.

```
//Circular Buffer Implementation
//Circular Buffer Class
```

```
class circular_Buffer {
public:
        //public variables
        int head_ = 0;
        int tail_ = 0;
        int sizeBytes_;
        int size_;
        bool startRead = false;
        circular_Buffer(int size)
        {
                sizeBytes_ = size;
                size_ = size/ DEFAULT_BUFLEN; //How many blocks per circular buffer
        }

        //Put Function
        void put(int *sampleArray, int OutputNum)
        {
                std::lock_guard<std::mutex> lock(mutex_);
                memcpy(&buf_[head_*DEFAULT_BUFLEN], sampleArray, DEFAULT_BUFLEN * sizeof(int));
                head_ = (head_ + 1) % size_;

                if (head_ == tail_)
                {
                        tail_ = (tail_ + 1) % size_;
                }
        }


        //Get Function
        void get(int* sampleArr, int OutputNum)
        {
                std::lock_guard<std::mutex> lock(mutex_);

                if (empty())
                {
                        if (OutputNum <= 8) {

                        }
                        for (int i = 0; i < DEFAULT_BUFLEN; i++)
                        {
                                sampleArr[i] = 0;
                        }

                        return;
                }else
                {
                        memcpy(sampleArr, &buf_[tail_*DEFAULT_BUFLEN], DEFAULT_BUFLEN * sizeof(int));
                        tail_ = (tail_ + 1) % size_;
                        return;
                }
        }


        void reset()
        {
                std::lock_guard<std::mutex> lock(mutex_);
                head_ = tail_;
        }

        bool empty()
        {
                //if head and tail are equal, we are empty
                return head_ == tail_;
        }

        bool full()
        {
                //If tail is ahead the head by 1, we are full
```

```
                //Kyle: What about if head_ = size?
                return ((head_ + 1) % size_) == tail_;
        }

        size_t size(void)
        {
                return size_ - 1;
        }

private:
        //private variables
        std::mutex mutex_;
        //used to dissallow simultaneous "reads and writes"/ "gets and puts"
        int buf_[(DEFAULT_BUFLEN * NUM_BLOCKS_PER_CIRCULAR)+1];
                //size enough for 4096 samples
};
//END OF CIRCULAR BUFFER CLASS
```

Listing C5: Circular Buffer class

APPENDIX D

This appendix provides all the material associated with the experimentation detailed in chapter six of this thesis. As such, The Questionnaire that was provided to participants, along with the Immersive Planets Manual are included. In addition, the Developer's Manual and the ImmerGo Unity Integration Capability Distribution Manual are also provided.

THE QUESTIONNAIRE

Answer the following questions in no more than two sentences. If the question asks for a rating, that is all you need to supply. If you have any queries regarding the questions, don't hesitate to ask the investigator.

i. Did you find the system usable; were you able to easily identify individual planets, and easily pan the planets around the scene (provide a rating; 0 being unusable, 10 being easy to use)?

ii. Have you experienced anything similar (demoing other immersive sound systems/ VR projects)?

iii. Were the visuals of the project at a high enough fidelity?

iv. How well did the manual/information sheet inform you of what tasks you were required to perform?

v. What was your perceived quality of the audio in both environments (provide a rating; 0 being poor quality, 10 being excellent quality)?

a) Speaker Environment:

b) Headphone Environment:

vi. In your opinion, do you think headphone or speaker-based immersive sound best complements an immersive experience?

vii. In your opinion, were there any obvious advantages that each of the environments had over the other, i.e., The speaker environment "felt" more realistic. Or vice versa.

viii. After having experienced both environments, what were the biggest problems for each (if any)? E.g Latency, precision, accuracy.

a) For the headphone environment;

b) For the speaker environment;

ix. Do you believe that Virtual Reality is something that will become more and more prevalent in our society, or do you believe the technology will remain niche?

IMMERSIVE PLANETS MANUAL

This document is divided into three sections, and a final subsidiary section that outlines the risks of participating in the experiment. The first section provides background information pertaining to the implemented system. The second section indicates the physical environment in which a user will find themselves and an explanation of how to navigate said environment. Finally, the third section indicates how users can interact with the environment, and how to navigate through the two scenes in the application.

Background Information

This experiment forms part of on-going research whose primary aim is to enable an immersive audio middleware capability within the Unity development framework. As such, the investigator has developed this capability, and is looking for input to analyse the current implementation.

As a participant, you will be asked to evaluate the capability via the use of a Virtual Reality application that has been developed using said capability.

The VR application was rendered using the Unity game engine. The Oculus Rift Development Kit 2 has been selected for the Head-Mounted Display (VR headset). Information pertaining to this iteration of the Oculus Rift is available online.[87] This HMD was released in 2014, and is the direct predecessor to the final product (which was released in 2016).

The motion-capture controller being used in the demo is the Leap Motion Controller.[88] The infrared sensors, and accompanying API enabled the rendering of a user/developers' hands in the game/virtual world. The Controller is also able to detect gestures, but currently, this implementation of the system only supports the "pinch" gesture detailed in the second section. In future implementations of the system, more gestures will be included in the project.

To spatialize the audio using a speaker array, the ImmerGo system was employed.[89] The speaker array consists of fourteen to sixteen MiniDSP speakers that were connected to a PWR 16 Amplifier. The amplifier was connected to two NDAC-8 Digital Signal Processors. These were bridged using a MoTu AVB switch. The Echo Streamware card was utilized to enable AVB audio on the host machine.

Physical Environment

A demarcated area ("user area") is located in the center of the array of speakers. This area is where the user should position his or herself. The "user area" is indicated in the diagram below (see Figure 1). It is important to recognize that in this instance there will be fourteen speakers mounted around the user, and for this reason, users should not move after they have entered the environment, as the test area is a confined space. Additional speakers may be added depending on the investigator's decision. After they have seated themselves on the chair within the area, the users will be given the HMD, and it will be adjusted on their heads. There should be nothing displayed at this point, and upon vocal agreement, the

[87] https://en.wikipedia.org/wiki/Oculus_Rift
[88] https://www.leapmotion.com/
[89] https://www.immersivedsp.com/products/immergo-16

application will be launched, and the user should perform the tasks detailed in the task sheet document accompanying this manual. After having closed the application, the user will be directed to a table where they can fill out the required questionnaire.

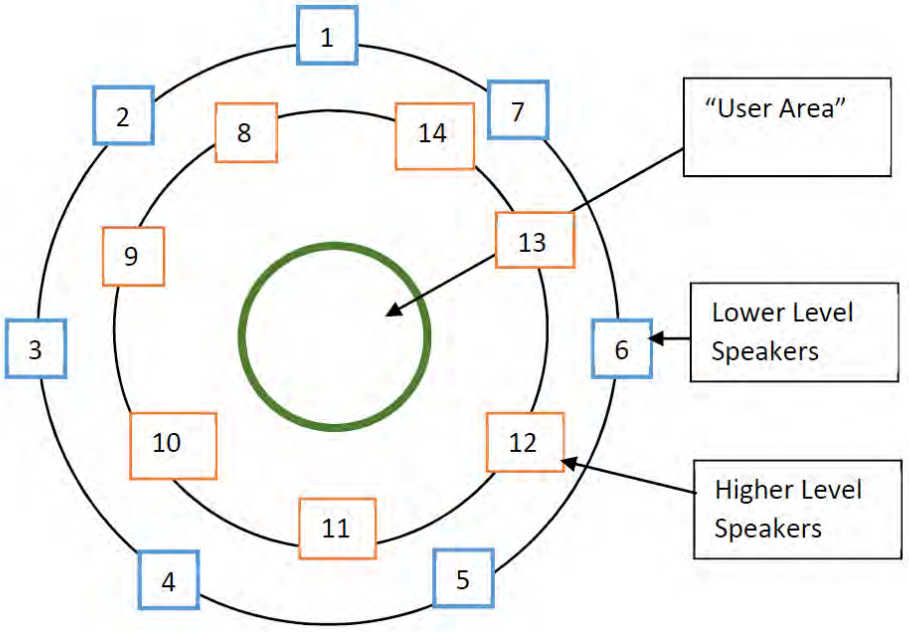

Figure 135: A simple diagram that indicates the physical set up of the speaker array, and where the user should be located within the set-up. In this instance, fourteen speakers are being demonstrated. In the final testing set-up, the speaker count may be higher.

Virtual Environment Interaction

In this application, a user will be able to interact with the environment in two ways; "**Selecting"** audio sources, or **"Clicking"** user interface elements. Importantly, the user will use the same gesture to perform both interactions. The selected gesture is the pinch (see Figure 2 for a diagrammatic representation of the pinch gesture).

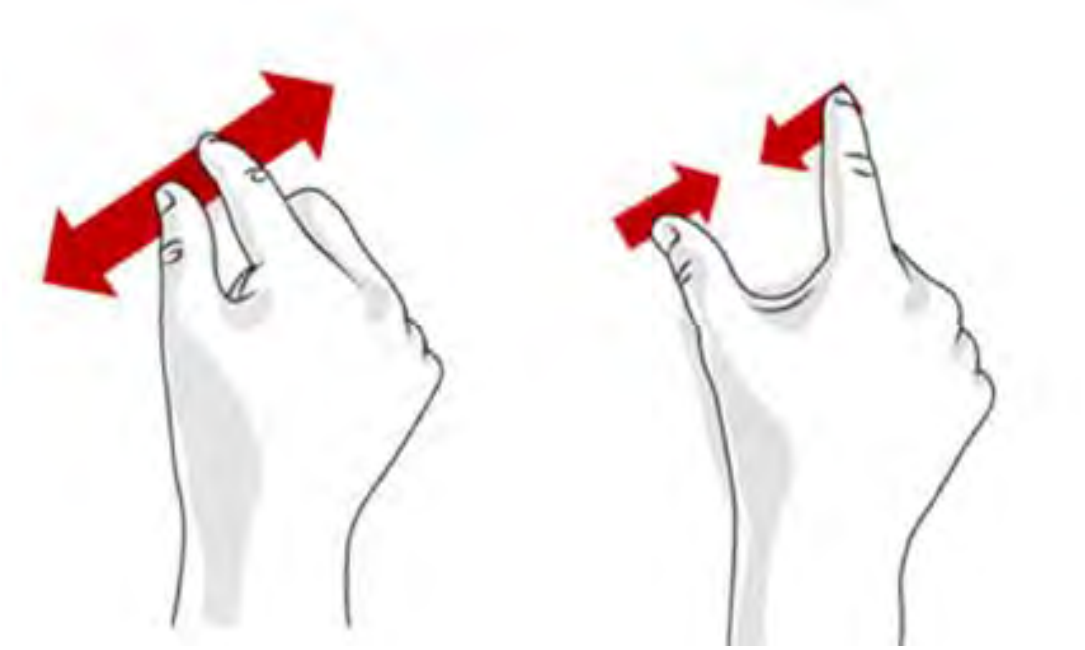

Figure 136: A diagram indicating the pinch gesture.

"**Selection"** involves the movement of an audio source/planet around one. Essentially, this interaction has been enabled through the use of an intuitive three-dimensional cursor. This cursor is visible to a user as a white sphere that moves in relation to how they move their hand (see Figure 3).

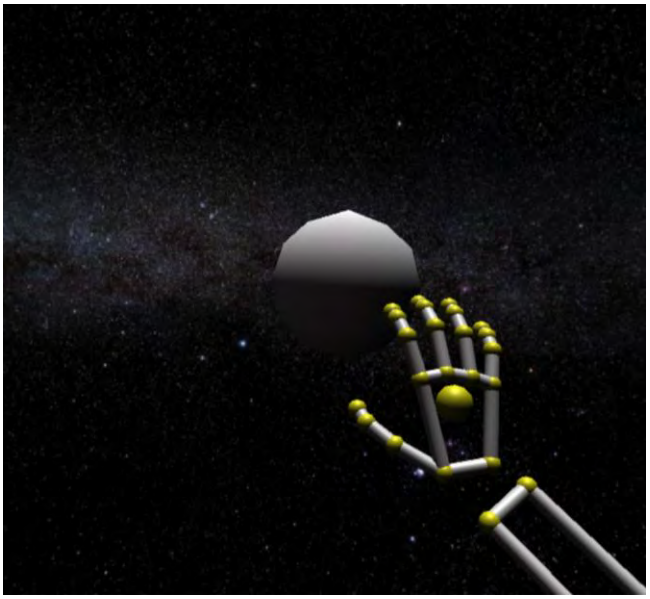

Figure 137: The 3D cursor is visible as the white sphere in the screenshot above. As mentioned, the cursor moves relative to the users' hand movements.

If a user performs the pinch gesture, and holds the pinch (i.e., the left side of Figure 2) they will notice that the sphere turns green, which indicates that the Leap Motion software has recognized the gesture (see Figure 4).

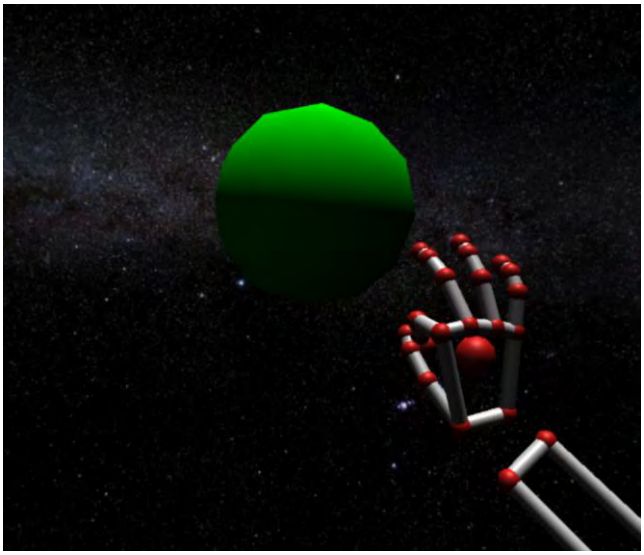

Figure 138: As can be seen in the screenshot, the 3D cursor turns green once the pinch gesture is recognized.

To actually pan the audio source/planet, a user is required to collide the 3D cursor with the planet. Once the 3D cursor overlaps the particular planet, the user should perform the pinch gesture. At this point, the planet is "selected" (see Figure 5). If the user holds their pinch, and moves the 3D cursor to another position in space, they will notice that the planet has bound to the cursor, and will follow it.

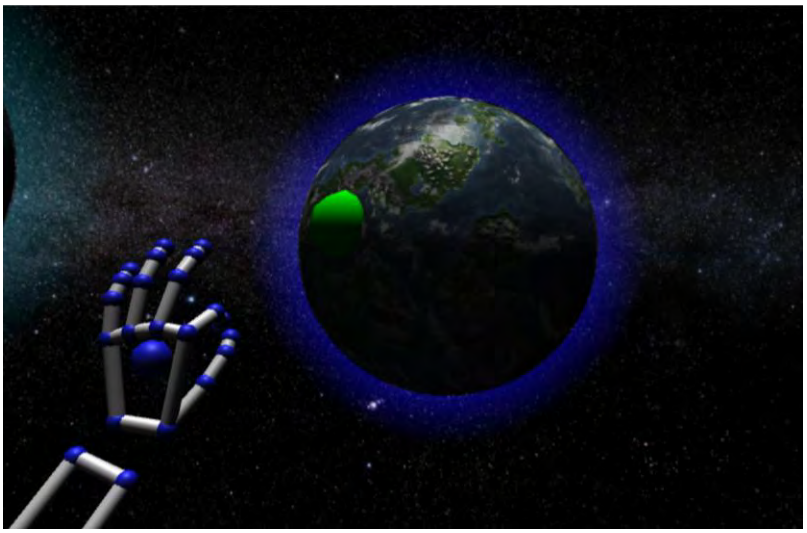

Figure 139: This screen shot indicates the selection of the planet in the scene. Should the user move the cursor while maintaining their pinch, they would notice that the planet is attached to the cursor and can be moved around the scene.

Once they have positioned the planet appropriately, they should simply release their pinch, and the planet will remain where they have left it. At this point they should notice that the cursor is no longer bound to the particular planet, and can be used to pan additional planets in the scene.

Risk Analysis

The user should be aware of the possible outcomes of using Virtual Reality technology. It has been well documented that simulator sickness is something that still hinders the technology. Simulator sickness

has been described to consist of primary symptoms, namely, eye strain, nausea and disorientation. Hettinger (co-author of Vection and Simulator Sickness which was published in the journal of “Military Psychology”) explains simulator sickness as “a form of motion sickness in which users of simulators exhibit symptoms characteristic of true motionsickness.”

Should you be affected by any of the symptoms described above, immediately indicate your discomfort, and the investigator will stop the demo, and help you remove the Head-Mounted Display.

Another potential risk that users may encounter (when using this system) would be that the physical set-up utilizes multiple wires, and cabling. For this reason, there is the chance that users may trip. However, all the cabling and wiring has been secured to the ground, which will minimize this risk.

DEVELOPER'S MANUAL

Having provided a brief overview on the VR application/demo that has been developed to showcase the immersive sound capability, it would be beneficial to now discuss how a developer can employ the capability within their own project(s). As such, the following sections will firstly provide an overview of the current implementation and the operating environment that is required for the capability. Secondly, a section detailing how a Unity developer should employ the capability to provide immersive audio in their project via the ImmerGo audio spatialization system will then be detailed.

Implementation Overview

The final implementation of the capability exists as three discrete components that function in unison. A detailed description of how these components function is omitted, but if the reader is interested, a short paper has been written on the capability that details how things function. However, a brief overview of the components' functions will be provided.

The first component is the ImDSP plugin. It is an FMOD compliant DSP plugin that houses a winsock client within (see Figure 140). The client essentially picks up audio samples of an FMOD event as the FMOD mixer pushes them out, and forwards the samples to the second component of the capability; the ASIO playback server. This server sends requests to the clients for more samples. The clients reply with their unique identifier, and a set of samples for playback.

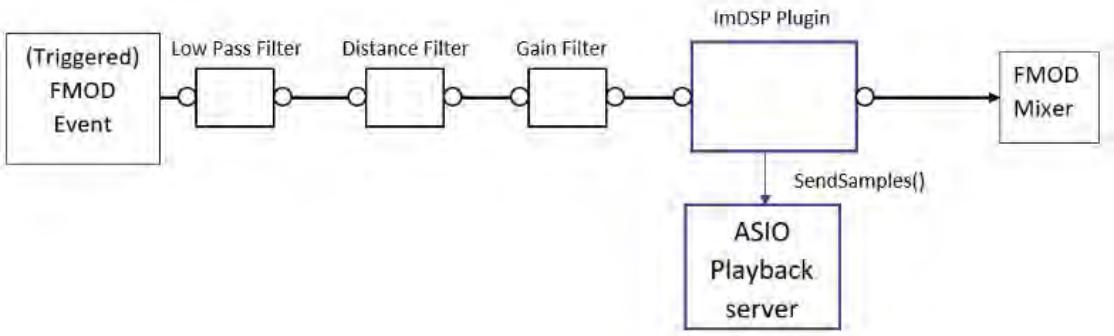

Figure 140: This diagram illustrates an FMOD event with multiple DSP effects forming a DSP "chain". As one can see, the ImDSP plugin should be placed at the end of the chain.

The ASIO playback server functions by using two threads. The first thread is driven by the appropriate ASIO driver, and requests for audio samples as they are consumed. The second thread is used to request for samples from the ImDSP clients. As the server receives samples, it places the samples into corresponding circular buffers. These buffers are then accessed when the driver requests for samples for the output channels via the bufferswitch callback.

In parallel, the third component of the capability, the "socketioclient" script operates from within Unity. This script utilizes a coroutine that recognizes whether an audio source within the Unity project/scene has moved, and if so, transmits the current coordinates of the particular source to the ImmerGo spatialization system, which in turn, performs the requisite 3D audio rendering, and transmits the calculated mix levels to the speaker end points.

The “socketioclient” script exists as a c# script that should be placed within the scene on an object that is active from the start. A good object for the script would be the camera (or parental object that houses the camera) in the scene. In the screenshot below, the script can be seen as a Unity component of the Leap Rig game object. The script firstly enables developers to select which objects within the scene they wish to be tracked. If a developer were to click on one of the boxes next to the “Source” text, they will be provided with a pop-up that enables them to select a game object that is present within the scene. Subsequently, the application’s Maximum and Minimum ranges can be inputted into the textboxes that require coordinate values. This process was introduced so that the developer can express the limits of the ranges of each axis. This then enables a conversion between the units within the Unity application, and the units of the ImmerGo spatialization system (see following figure).

Figure 141: This screenshot of the Unity inspector window is taken from the Immersive Planets demo. The Leap Rig is the game object within the Unity scene that contains the camera (amongst other important functionality for the Leap Motion controller).

Operating Environment

The following section will provide a brief overview of the software environments and applications that a developer will need to employ. After which, the hardware that was utilized in the implementation will also be examined.

The Unity Game Engine

The capability was developed using Unity 2017.1.0f3. However, it should function on any later version of the game engine. Certain knowledge of how the game engine operates should be known.

The developer is required to integrate their project with the FMOD Unity Integration package (available from the FMOD website). This integration will result in an additional drop down menu in the game engine, to change FMOD settings, and supply a valid path for the correct FMOD studio project (see Figure 142 below).

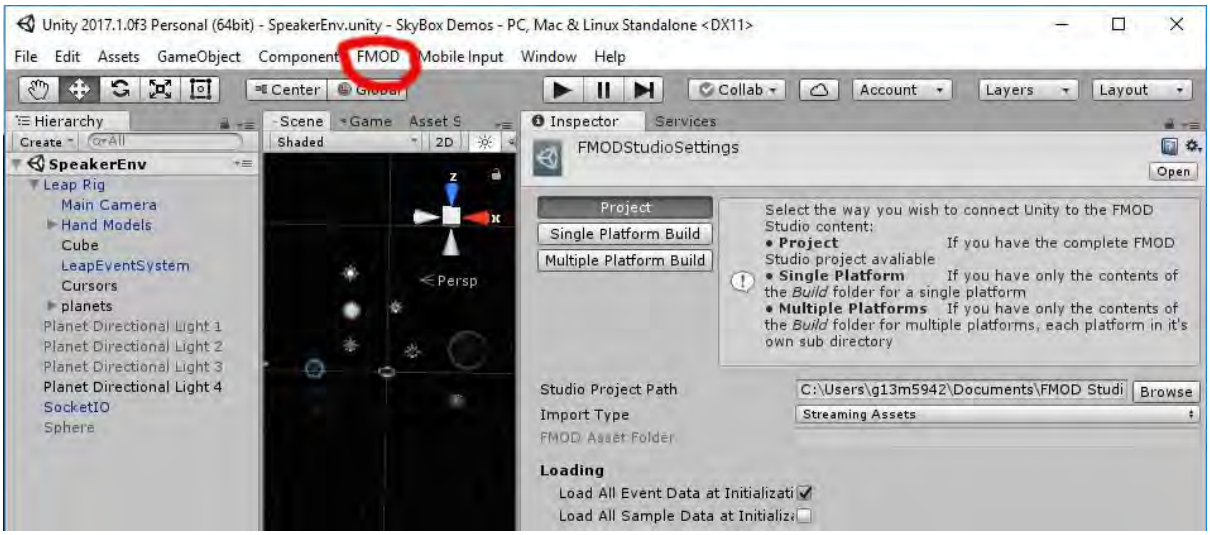

Figure 142: A screenshot indicating how the Unity GUI is altered after a developer integrates their project with the FMOD Unity Integration package. The FMOD project settings are navigated to via the FMOD drop-down menu (circled in red).

FMOD Audio Middleware API and Studio Application

During the implementation of the capability, it was decided that the optimal way to enable advanced Digital Signal Processing functionality would be to integrate the capability into an audio middleware software suite. As such, the FMOD application was selected for this reason. To properly utilize the functionality that the capability provides, it is assumed that the developer understands the FMOD workflow.

In essence, an audio engineer will author FMOD events within the Studio Application. These events are "built" into FMOD audio banks, which can then be accessed from within Unity. The game developer will then refer to the built events from within their Unity project, and decide when they should trigger according to their needs.

The FMOD Studio Application enables a many DSP filters and effects that can be bound to game events (before being built). As such, the capability was developed to reside within this paradigm, and essentially acts like any other DSP filter, i.e., the developer needs to bind the "ImDSP" plugin to the end of their effect chain for a particular event (see Figure 143). The plugin should be placed at the end of the chain, as it sends the audio samples out to the ASIO playback server, and thus any effects that are added to the chain subsequent to the "ImDSP" plugins, will not be rendered before the samples are transmitted to the environment.

Figure 143: This screenshot shows an overview of a FMOD audio event within the FMOD Studio Application. The specific event ("Earth") has a few DSP effects on its DSP chain (Fader, Flanger, Distance Filter).

To add the plugin to your FMOD integrated Unity project, one should simply place the compiled .dll from their filing system into the "Plugins" folder which can be seen in the project hierarchy under *Assets/Plugins/x86_64* (for 64-bit plugins), and *Assets/Plugins/x86* (for 32-bit plugins). The ImDSP plugin is a 64-bit plugin (see following screenshot).

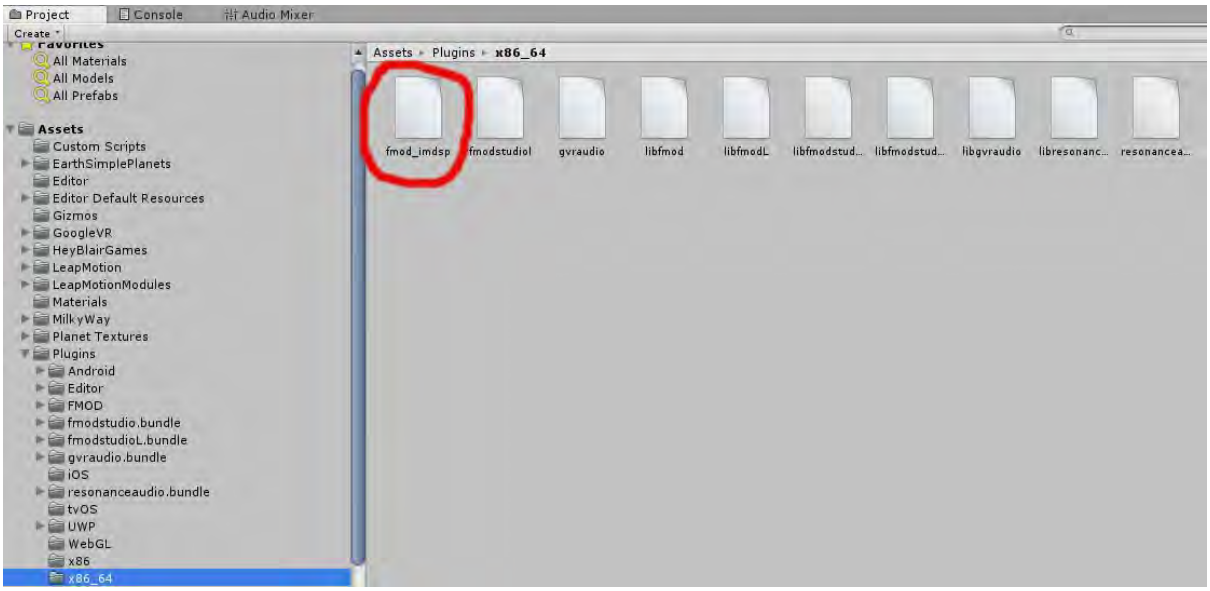

Figure 144: As mentioned in the previous paragraph, FMOD compliant plugins that have been developed should be placed within the Plugins folder of the Unity project's file system hierarchy. In the screenshot above, the ImDSP plugin is highlighted in red.

A final and important requirement of the implemented capability is that developers need to downmix their stereo/multichannel encodings to mono signals (the present implementation does not do this for the developer). If developers omit to do this, the ImDSP plugin will only transmit samples from the left channel of the encoding.

ASIO 4 All SDK (by Steinberg)

The ASIO4All SDK was utilized in the development of the playback server. Essentially, the server functions by selecting the appropriate ASIO driver to use for audio playback (the "ASIO Streamware" driver should be used to enable AVB audio). As the bufferswitch callback operates, a receiver thread constantly calls for the ImDSP plugins to provide more audio samples.

ImmerGo Audio Spatialization System

The ImmerGo audio spatialization system is an AVB speaker-based immersive audio system that encompasses a variety of 3D audio rendering algorithms. The system functions by receiving coordinates of audio tracks, and then calculates the corresponding mix levels for each of the speaker end-points in the array (see Figure 145). In the context of this capability, the server has been altered to only receive coordinates and perform the requisite calculations. Other implementations of the system provide developers and audio engineers with audio playback and control functionality for Digital Audio Workstations (REAPER).

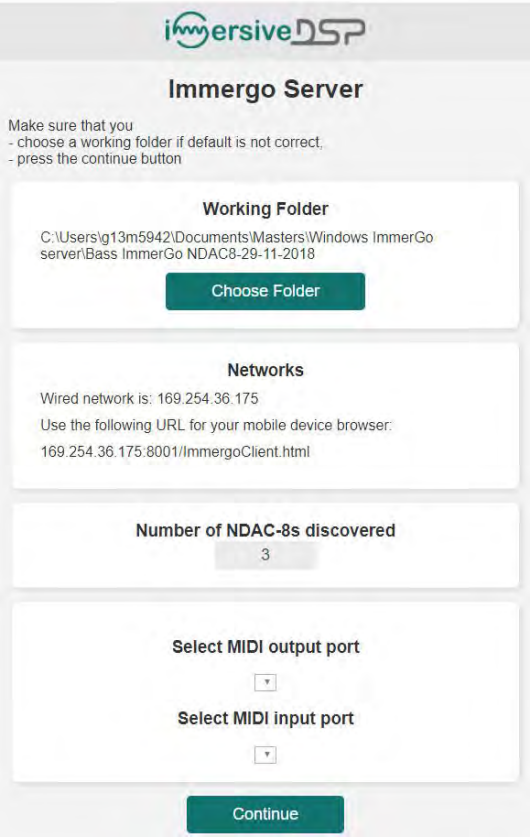

Figure 145: This screenshot indicates how the ImmerGo server's User Interface looks. In this specific image, one should note that the server has already transmitted discovery messages, and has recognized the two NDAC-8s on the AVB network

Echo Streamware Controller Application

The current capability has been implemented to use AVB standards for audio networking. As such, the Echo Streamware card was used for an appropriate Windows compatible Audio Network Interface Card. Users of the controller application should be able to identify, and connect talker streams to the two network DACs from the interface.

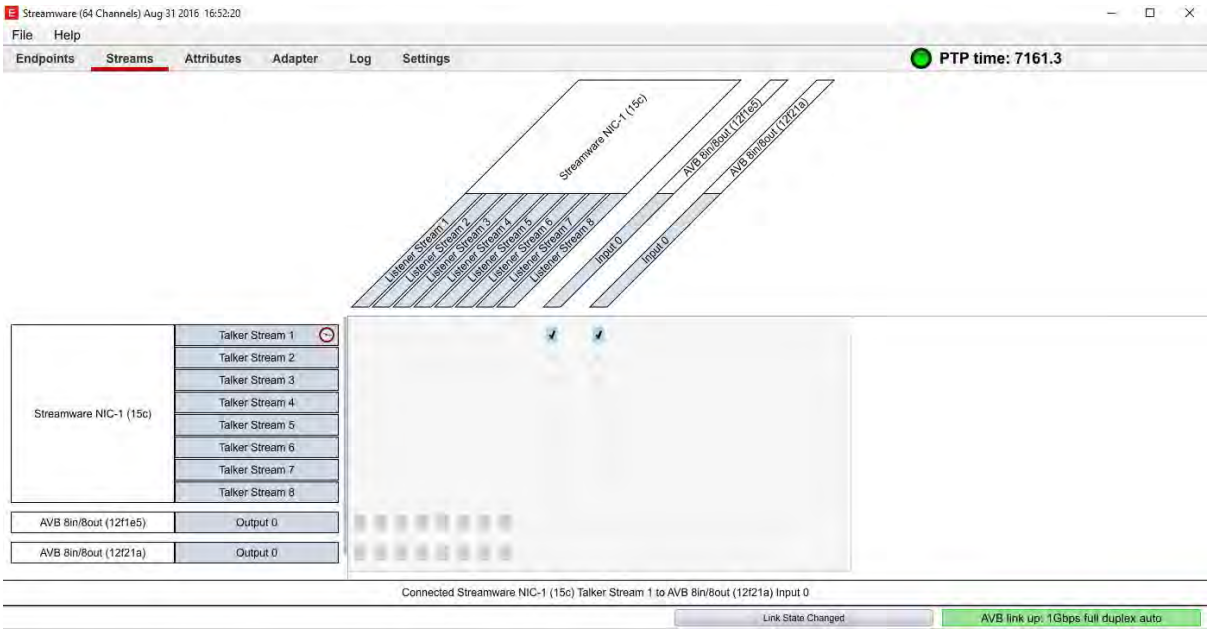

Figure 146: A screenshot indicating the User Interface of the Echo Streamware controller application. As can be seen, the two NDAC-8s are connected to Talker Stream 1.

Project Hardware

- The Echo Streamware card: An AVB network interface card that functions on windows machines.
- Two MiniDSP NDAC-8s
- A MiniDSP PWR-16 amplifier
- A MoTu AVB Switch: used to distribute audio from the Streamware card to the NDAC-8s.
- 15 Speakers mounted in a circular speaker array
- A Yamaha NS-SW200 subwoofer

Steps to Utilizing the Capability

Pre Runtime

I. Author FMOD audio events from within the FMOD Studio Application, and build the events into an FMOD event bank.

II. Access the events from within Unity via a script, and bind the ImDSP instances to the FMOD events (a future consideration is to alter the FMOD event emitter script to, by default, bind an ImDSP instance on each FMOD event). Review the “ImScript” within the custom scripts folder of “Immersive Planets” on how to perform this binding process.

III. Ensure the appropriate game object (audio source) is being tracked by the socketioclient script (see Figure 7).

Runtime

A current requirement of the system is to launch the Unity application first, before launching the ASIO playback server. As the system is still being developed for, the playback server is used as a standalone application for debugging purposes. Once the system has been fully developed, the playback server will

be integrated into Unity, and will therefore operate from within the game engine, and not as an external application.

I. Launch the Echo Streamware controller software and ensure the Talker Streams can connect to the AVB devices (the two NDAC-8s).
II. Launch the edited ImmerGo server. The NDAC-8s should be recognized by the server application, and if the Talker Streams in the Streamware controller have disconnected, the developer should notice these connections re-establishing themselves.
III. Launch the Unity Application.
IV. Launch the ASIO playback server.

IMMERGO UNITY INTEGRATION CAPABILITY DISTRIBUTION MANUAL

These instructions indicate how to launch a sample VR application (“Immersive Planets”), that has been developed to showcase the capability.

The functionality of the capability is detailed in other material, but a brief precise on the capability implementation is provided at the end of this document. To utilize the capability in a separate Unity application, the “Immersive Planets Manual” provides information on the workflow that the capability has been integrated into.

Therefore, the following document indicates the steps required to launch an application that has been developed with the capability, and not how to deploy the capability in other Unity applications. The document should be reviewed linearly, as the steps in launching the application are ordered (i.e., if the ImmerGo server has not been initialized before the demo application is launched, there will not be any spatial audio).

AVB Stream Connection

For the ImmerGo spatialization system to function, the AVB Talker Streams need to be correctly configured, i.e., the Talker Streams are connected to appropriate AVB input devices. Launch the appropriate AVB controller software (in this instance, the Streamware controller software was used). To connect the talker streams, ensure the PTP grandmaster has been correctly determined, and connect streams to input using the GUI. The following screenshots indicate how this process is achieved when using the ASIO AVB Streamware software and Echo NIC (see Figure 146 andFigure 147).

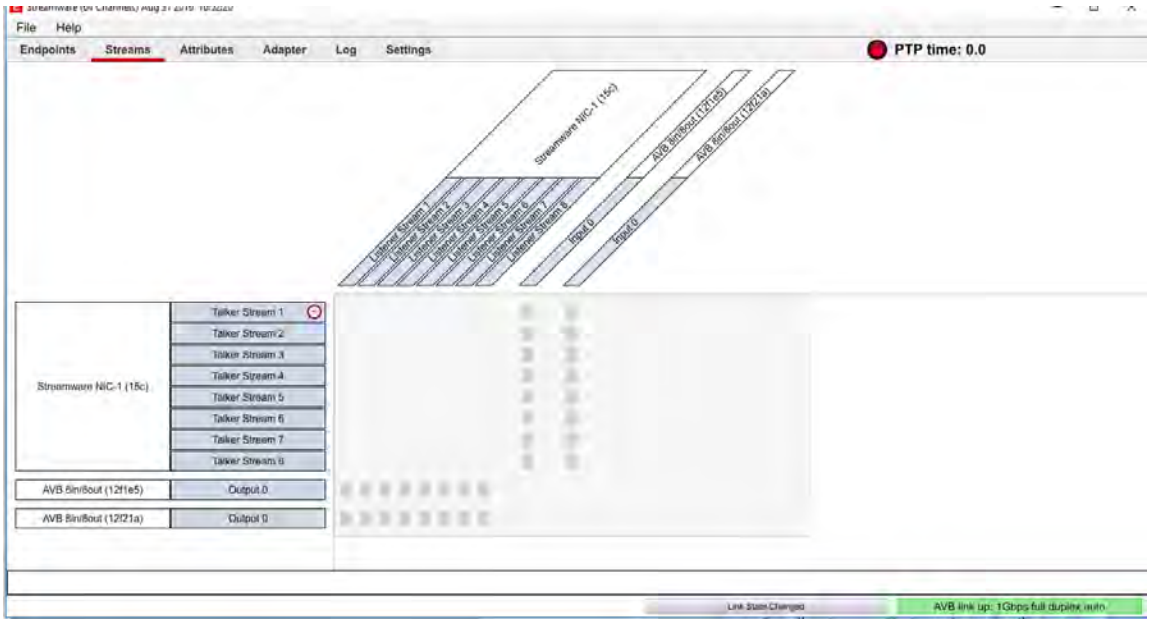

Figure 147: This screenshot indicates the state of the controller software when the talker streams have not been connected, and the PTP grandmaster is unassigned.

ImmerGo Server Initialization

Once the streams have been correctly configured, the ImmerGo server should be launched. The user should ensure that the appropriate speaker configuration has been provided to the ImmerGo server (the

“speakerconfig.xml” file is located in the root of the ImmerGo server file hierarchy). The xml file indicates the positions of each of the speakers in the array (these positions will be relative to the number of speakers in the array and the user-defined distribution of speakers in the array). A screenshot of this file is provided in Figure 148.

```
-->

<config>
  <amplifier macadr="70:b3:d5:12:f1:e5">
    <speaker number = "1" xpos = "-106" ypos = "0" zpos = "106" stype="normal">  </speaker>
    <speaker number = "2" xpos = "0" ypos = "0" zpos = "150" stype="normal">  </speaker>
    <speaker number = "3" xpos = "106" ypos = "0" zpos = "106" stype="normal">  </speaker>
    <speaker number = "4" xpos = "-150" ypos = "0" zpos = "0" stype="normal">  </speaker>
    <speaker number = "5" xpos = "150" ypos = "0" zpos = "0" stype="normal">  </speaker>
    <speaker number = "6" xpos = "-106" ypos = "0" zpos = "-106" stype="normal">  </speaker>
    <speaker number = "7" xpos = "0" ypos = "0" zpos = "-150" stype="normal">  </speaker>
    <speaker number = "8" xpos = "106" ypos = "0" zpos = "-106" stype="normal">  </speaker>
  </amplifier>

<!--
Second NDAC (CHANGE SPEAKER NUMBERS AND POSITIONS)
-->
  <amplifier macadr="70:b3:d5:12:f2:1A">
    <speaker number = "9" xpos = "-75" ypos = "106" zpos = "75" stype="normal">  </speaker>
    <speaker number = "10" xpos = "0" ypos = "106" zpos = "106" stype="normal">  </speaker>
    <speaker number = "11" xpos = "75" ypos = "106" zpos = "75" stype="normal">  </speaker>
    <speaker number = "12" xpos = "-106" ypos = "106" zpos = "0" stype="normal">  </speaker>
    <speaker number = "13" xpos = "106" ypos = "106" zpos = "0" stype="normal">  </speaker>
    <speaker number = "14" xpos = "-75" ypos = "106" zpos = "-75" stype="normal">  </speaker>
    <speaker number = "15" xpos = "0" ypos = "106" zpos = "-106" stype="normal">  </speaker>
    <speaker number = "16" xpos = "75" ypos = "106" zpos = "-75" stype="bass">  </speaker>
  </amplifier>
</config>
```

Figure 148: This screenshot of the “speakerconfig.xml” file indicates the 16 speakers and their positions, that were used in this particular immersive sound configuration. The MAC address of the NDAC8s enables the ImmerGo server to identify the devices on the AVB network, and the positions of the speakers that are connected to the device. The speaker identifier (speaker number), x,y, z positions, as well as “speaker type” are all requisite parameters for each speaker entry in the configuration.

To launch the ImmerGo server, one should simply open up a PowerShell/command line console in the root of the server hierarchy (see Figure 149), and input the command “./nw”. A separate document indicating ImmerGo server control, provides a clear instruction set to initialize the server.

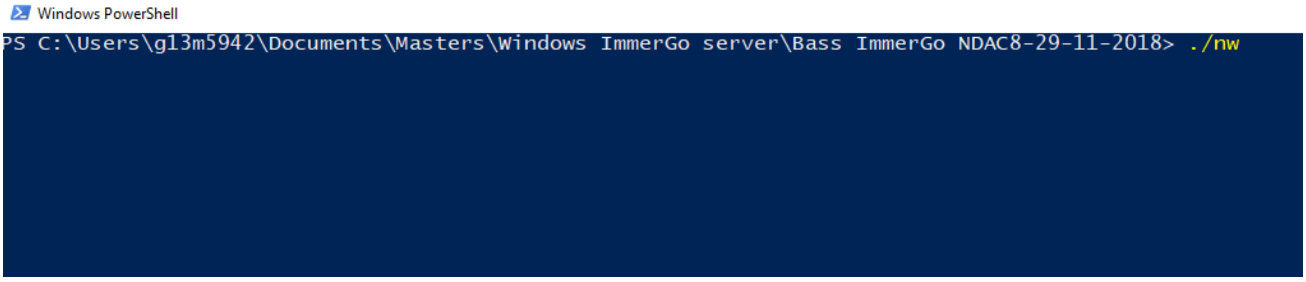

Figure 149: This screenshot indicates a Windows PowerShell console that is operating in the root of the ImmerGo server application. The “./nw” command is the common build and compile command for Node.js and NW applications.

Having executed the launch command, the ImmerGo server application will launch, and provides one with various information, i.e., the working directory for the server to find the “speakerconfig.xml” file, the network interface that the server is using, the number of AVB devices that are discoverable on the

network, and MIDI output and input ports, which are not used in this Unity integration solution (see Figure 150). Make sure that the working directory in which the “speakerconfig.xml” resides is correctly located, by selecting the directory from the pop-up window that is provided once a user has pressed the “Choose Folder” button.

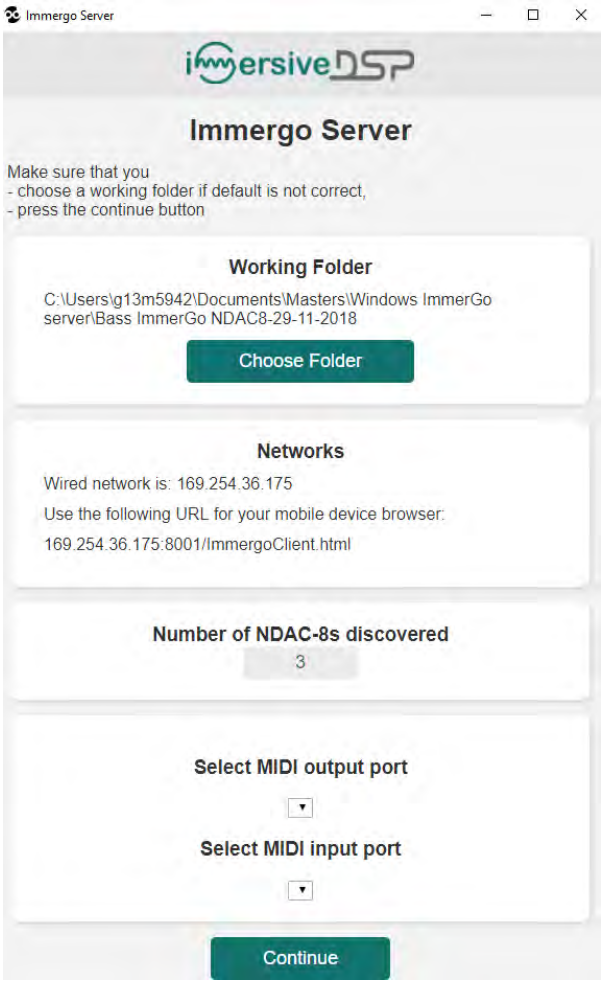

Figure 150: A screenshot of the ImmerGo server application, indicating the various information that the server provides the user.

Once the NDAC8s have been discovered, the server will be put into a running state, after the “Continue” button (see preceding figure), has been pressed. The exact messages and handshaking that is conducted can be reviewed using the “Developer Tools” console for the application. If the server recognizes the an additional NDAC8 on the network, it is because the server also recognizes the Echo Streamware card as an AVB device (so don’t be concerned about an invisible device).

Having correctly configured the AVB streams, and the initialization of the ImmerGo server, the demo application can be launched.

Demo Application Initialization

In the root of the distribution application file hierarchy, launch the “Immersive Planets Development Build.exe” application (see Figure 151). After which, you should be confronted with the classic Unity-developed-game-launcher (see Figure 152). You can then select the appropriate graphical options for

the application before it is launched.

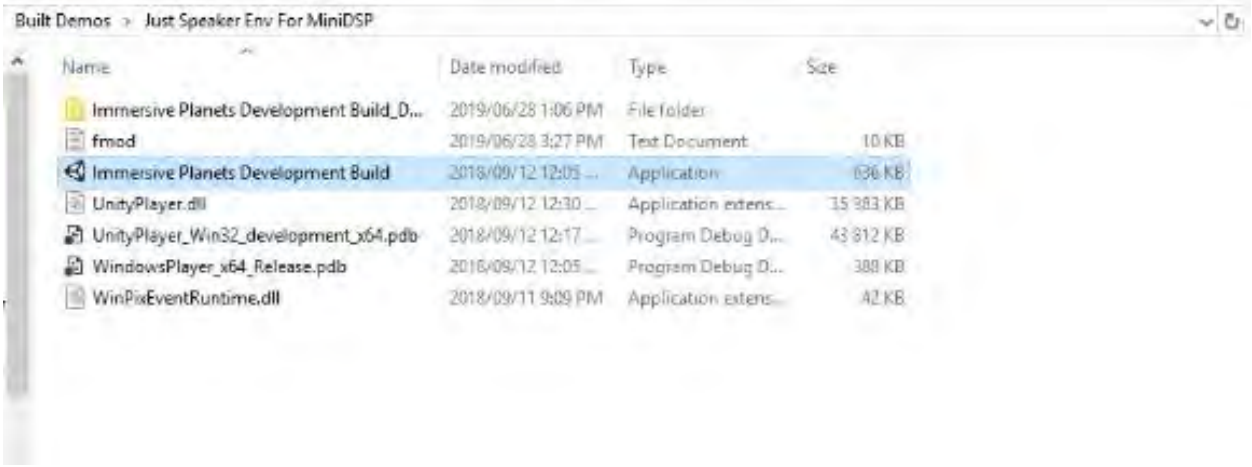

Figure 151: A screenshot indicating the executable and project files for the demo application.

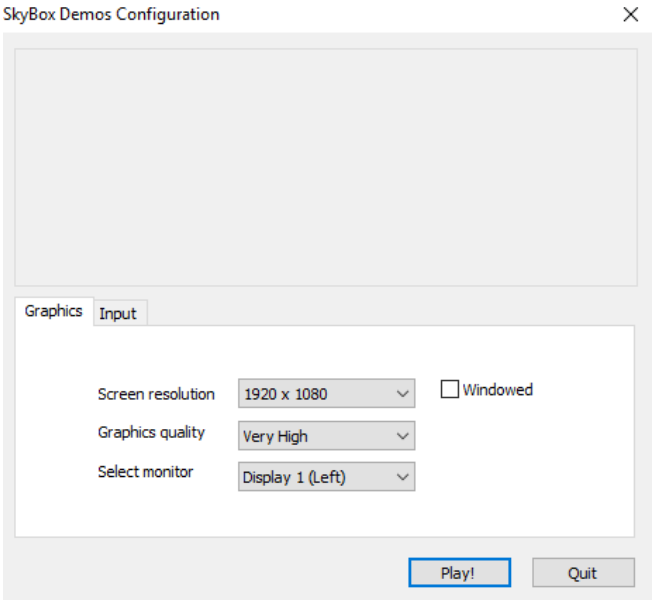

Figure 152: A screenshot of the Unity application launcher, whereby the user can select what resolution and Graphics quality that is optimal for their particular machine, as well as whether they want the application to run in a windowed mode. Of course, this is a VR application, but the display of the HMD is duplicated on another traditional display, so that observers may review what is being seen in the environment.

Once you select the "Play!" button, the VR application will launch after a few seconds. You should ensure beforehand, that your VR headset is fastened correctly, and that you are facing forwards; towards your HMD's positional tracker (see Figure 153). In tandem to the applications launch, the server application that communicates with the ASIO driver needs to be launched. To launch the server, one should press the "w" key once the demo has successfully loaded (see Key Bindings section below). The server application will appear as a console window with various information about the driver that has been selected (see Figure 154).

Figure 153: The positional tracker of the Oculus Rift DK2. The Oculus Rift CV1 uses "Oculus Sensors", which perform the same role.

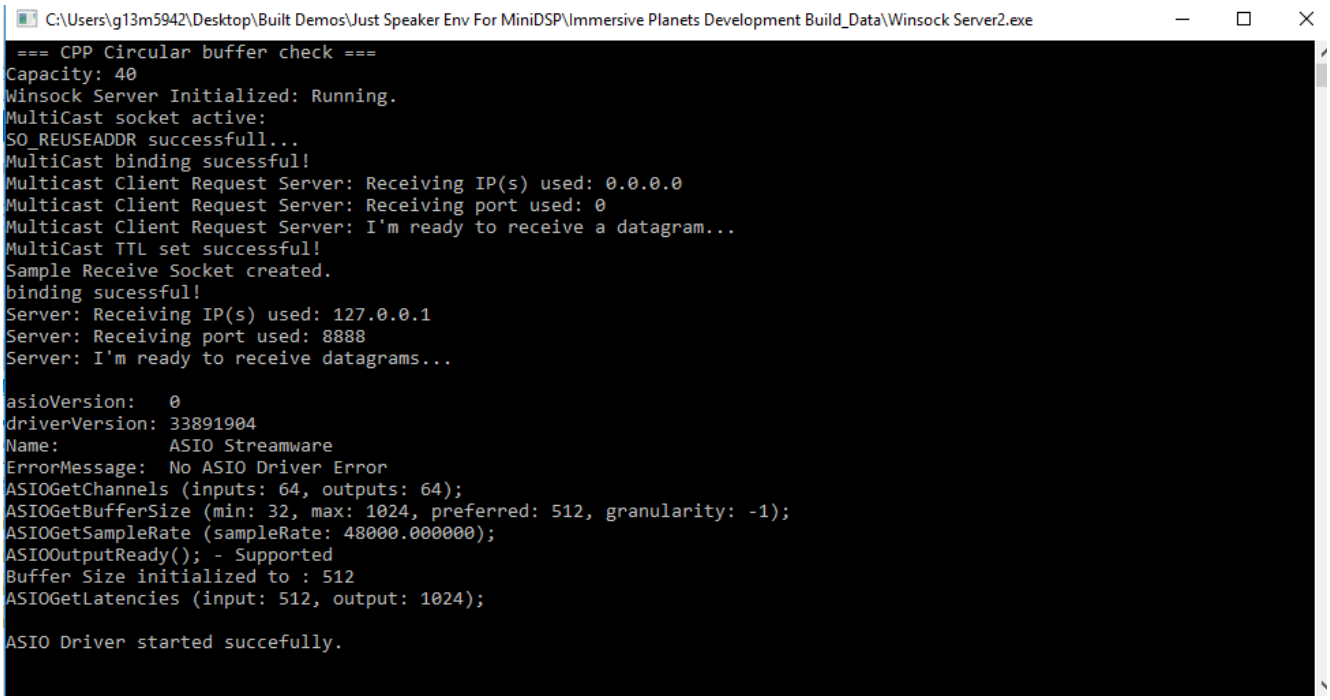

Figure 154: A screenshot that indicates how the ASIO playback server appears as a console application. As can be seen, various handshaking information is provided on launch. In this instance, the server has correctly initialized a handle to the ASIO streamware driver, and is expecting to receive audio samples.

The current application doesn't provide any playback control to the user through any input of the Leap Motion controller. However, the following keyboard bindings indicate how playback of a planet can be controlled, as well as the orbiting function of the planets.

Key Bindings

After playing around in the environment, you can exit and close the application by pressing the "ESCAPE" key.

Solo Planet Bindings:

The following keys function by muting all the other planets, and soloing a specific one.

- ➔ "e" to solo Earth audio.
- ➔ "j" to solo Jupiter audio.
- ➔ "m" to solo Mars audio.
- ➔ "y" to solo Mercury audio.
- ➔ "n" to solo Neptune audio.
- ➔ "s" to solo Saturn audio.
- ➔ "u" to solo Uranus audio.
- ➔ "v" to solo Venus audio.

"w" will launch the winsock server and begin audio playback, after the application has launched.

"o" is used to initiate all the planets' orbits.

"a" is used to stop the planets from orbiting.

"x" can be used to mute all the active audio events (planets).

The user of the application should allow the planets to orbit around them to experience the spatial audio. Remember to solo particular planets if you wish to hear a particular audio track in orbit.

User Interaction

In addition to the orbiting function of the planets, a user can manually pan the planets around them to specific locations via the use of the Leap Motion controller's 3D cursor. A detailed explanation of how to perform this interaction, with diagrams, is provided in the "Immersive Planets Manual" document that was created for participants (who took part in the qualitative research project). Please review this document for a clear understanding of how to interact with the environment using the Leap Motion controller.

The functionality of the capability is also detailed in the aforementioned "Immersive Planets Manual", and provides information on how the capability can be used by developers in their own Unity projects. However, a brief overview of the capability's implementation is outlined in the proceeding paragraphs.

Essentially the capability exists as three components. The first component, the ASIO Playback server is an audio playback server that has been developed to process incoming audio samples from winsock clients and play them back over ASIO output channels.

The second component is the ImDSP plugin, that sits at the end of an FMOD audio event chain, and has an embedded Winsock client within that transmits the audio samples of the particular event to the playback server. These plugins need to be correctly bound to FMOD events before they are triggered (this logic is performed by a C# script).

The final component is the coordinate transmission logic that is housed in various custom scripts within the Unity project. Essentially, the scripts allow a developer to keep track of game objects (that they identify) positions in the scene and transmit these coordinates to the ImmerGo server if any of their positions' change.