TAMPEREEN
AMMATTIKORKEAKOULU

DESIGN AND DEVELOPMENT OF A COLLABORATIVE VIRTUAL REALITY ENVIRONMENT

Mikko McMenamin

Bachelor's Thesis
May 2018
Business Information Systems
Game Development

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Game Development

MCMENAMIN MIKKO:
Design and Development of a Collaborative Virtual Reality Environment

Bachelor's thesis 58 pages
May 2018

The objective of this thesis was to gather information on the design and development of a collaborative virtual reality environment. This was achieved through researching and comparing in detail different technological solutions, such as game engines and networking libraries, for creating an online multi-user virtual reality application. Designing usability and user interface in virtual reality applications was also discussed.

This bachelor's thesis was commissioned by Intopalo Oy, a software company specialized in the digitalization of business. The purpose of this thesis project was to research the development process and to produce a collaborative virtual reality software product. The results of this thesis consist of a collaborative virtual reality application for Planmeca Oy, a customer of Intopalo Oy, and this study report.

Qualitative, secondary research methods were used for this thesis. Data was collected using books, technical documentations and online articles.

The conclusion is that Intopalo Oy and Planmeca Oy were satisfied with the results. Even though the chosen technologies and methods met the requirements, the study suggests that for larger projects different approaches are recommended. Based on this study and its findings, Intopalo Oy can continue developing a collaborative virtual reality platform for its customers. Confidential information was not included in this report.

Key words: virtual reality, collaborative environment, networking

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

MCMENAMIN MIKKO:
Monen käyttäjän virtuaalitodellisuusympäristön suunnittelu ja kehitys

Opinnäytetyö 58 sivua
Toukokuu 2018

Tämän opinnäytetyön tavoitteena oli tutkia, miten voidaan suunnitella ja kehittää virtuaalitodellisuuden monikäyttäjäympäristö. Opinnäytetyössä tutkittiin ja vertailtiin yksityiskohtaisesti eri teknologisia ratkaisuja, kuten pelimoottoreita ja tietoverkkokirjastoja ja niiden hyödyntämistä verkossa toimivan monen käyttäjän virtuaalitodellisuussovelluksen kehittämiseen. Lisäksi käsiteltiin käytettävyyden ja käyttöliittymän suunnittelua yhteistoiminnallisissa virtuaalitodellisuussovelluksissa.

Opinnäytetyön toimeksiantaja oli tamperelainen ohjelmistoyritys Intopalo Oy, joka on erikoistunut liiketoiminnan digitalisointiin. Opinnäytetyön tarkoitus oli tuottaa Intopalolle tietoa kehitysprosessista ja suunnitella sekä toteuttaa Intopalon asiakasyritys Planmeca Oy:lle virtuaalitodellisuuden monikäyttäjäsovellus.

Tutkimusmenetelmänä käytettiin kvalitatiivista tutkimusta ja työssä hyödynnettiin pääasiallisesti toissijaista tutkimustietoa. Tietoa kerättiin kirjoista, teknisistä dokumentaatioista ja verkkoartikkeleista.

Yhteenvedossa todetaan, että Intopalo Oy ja Planmeca Oy olivat tyytyväisiä tuloksiin. Siitä huolimatta, että valitut teknologiat ja menetelmät vastasivat vaatimuksia, tutkimustyön perusteella laajempiin projekteihin suositellaan vaihtoehtoisia menetelmiä. Opinnäytetyön pohjalta monikäyttäjäympäristöä ja sen toteutusta voidaan kehittää pidemmälle ja mahdollisesti tuotteistaa siitä laajemmalle asiakaskunnalle soveltuva tuote. Luottamuksellinen aineisto on poistettu raportista.

Asiasanat: virtuaalitodellisuus, monikäyttäjäympäristö, tietoverkot

TABLE OF CONTENTS

ABBREVIATIONS

AWS	Amazon Web Services
HLAPI	High-Level API
HMD	Head-Mounted Display
IDE	Integrated Development Environment
IK	Inverse Kinematics
IP	Internet Protocol
LAN	Local Area Network
LLAPI	Transport-Level API
MOG	Multiplayer Online Game
NAT	Network Address Translation
P2P	Peer-To-Peer
RPC	Remote Procedure Call
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE4	Unreal Engine 4
UX	User Experience
UI	User Interface
VoIP	Voice over Internet Protocol
VR	Virtual Reality

1 INTRODUCTION

Virtual reality, or VR, has taken the world by storm in the last couple of years. After the release of multiple high-end consumer VR devices such as HTC Vive and Oculus Rift in 2016, VR applications have become a multi-billion-dollar market globally. So far gaming has been the main focus for most content providers, partly because VR development is very similar to game development and gamers have been the early adopters of VR. However, VR is increasingly being used in media and education and there is a significant, fast-growing market in the industrial and professional sectors.

In the internationally best-selling book Ready Player One, and its 2018 record-breaking movie adaptation, VR is intertwined with every aspect of life. People work, socialize, compete and spend their free time in the virtual world. The China President of HTC Vive has named this the 'VR-First Future' and thinks that a form of this will become a reality in the not too distant future. (Road to VR 2018) Oculus founder Palmer Luckey has also stated that he believes that social applications will be the most popular virtual reality experiences in the long term and that distance between people will become irrelevant (Johnson 2015).

Besides consumer use, VR is used for productivity enhancement, visualization, R&D, spatial awareness, healthcare, simulations and marketing. Designers, engineers and architects can visualize and interact with photorealistic models from a first-person perspective in real-world scale, creating endless possibilities for industrial development. Almost any scenario that can be imagined can be implemented in VR as an interactive environment. In these kinds of professional settings, collaborative work is essential. In a collaborative VR environment, it is possible to work efficiently with colleagues around the world in the same 3D space.

This thesis provides insight into the design and development of a collaborative virtual reality environment. The reason behind implementing this project was the growing industrial interest in the aforementioned collaborative VR solutions. The purpose of this thesis was to gather information of the development process and to produce a collaborative VR application. This thesis study was commissioned by Intopalo Oy, a software company that focuses on digitalization. The results consist of this report and a collaborative virtual reality application for Planmeca Oy, a leading high-tech dental company, and a prototype platform for Intopalo Oy.

The thesis was conducted using the constructive research methodology and includes a case study. The research was primarily conducted in a qualitative manner. Although some parts of the thesis required specific primary research, including communication with the customers, the thesis is mainly based on literature and theory.

This thesis report is divided into nine chapters. After the introduction chapter, an overview of VR is given: what VR is, the state of the VR ecosystem and different use cases. Insight is also given for VR design and usability aspects. The third chapter consists of theory, where relevant technologies such as game engines and networking libraries are examined. The design of the software framework for collaborative VR development is researched in chapter four and the differences between game engines and networking libraries are analyzed. In the fifth chapter the programming methods for VR networking are explained and detailed examples are given. The sixth chapter focuses on a customer case: implementing a collaborative VR solution for Planmeca Oy. The results of this thesis project are presented in the seventh chapter and analyzed in the eight chapter. The eight chapter also includes the discussion on the pros and cons of the project and provides ideas for future development. Finally, a conclusion is drawn in the ninth chapter.

2 VIRTUAL REALITY

In this chapter the concept of virtual reality is introduced and insight to the current state of the VR economy is provided. This chapter explores the possibilities of VR and why it has become an emergent technology. Designing a comfortable experience is probably the most important and challenging consideration in VR development and therefore the design methods and guidelines are also considered and discussed.

2.1 Virtual Reality Defined

According to Charlie Fink (2018, 276), virtual reality is an experience that requires a headset to completely replace a user's surround view with a simulated, immersive, and interactive virtual environment. In the book Virtual Reality, Steven Lavalle (2015, 2) defines virtual reality in broader terms as inducing targeted behavior in an organism by using artificial sensory stimulation, while the organism has little or no awareness of the interface. He also states that VR technology is evolving rapidly and therefore defining VR in terms of specific devices would result in unstable definitions.

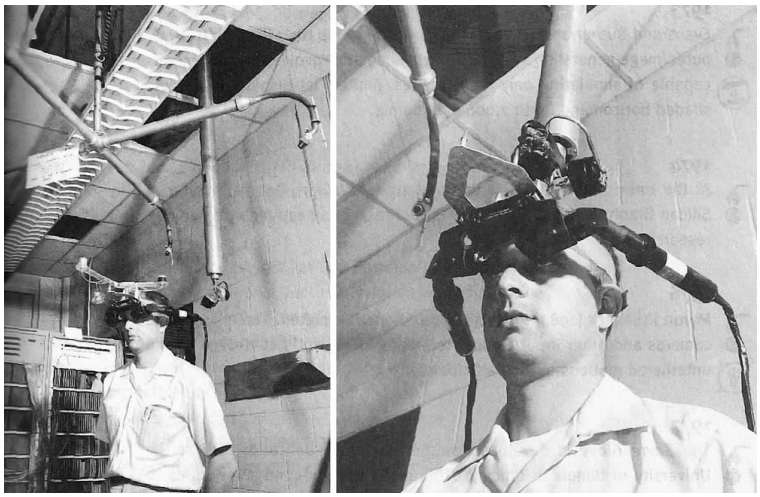
> Virtual Reality is about humanity's quest for immersion. It provides presence and agency in other worlds, in stories and myths, and it stretches from Plato's cave to religious rituals, theater, dark rides, theme parks, film, television, and video games. These experiences require our willing suspension of disbelief. In short, VR is a new reality. (Fink 2018, 329)

In practice, the most common way of experiencing VR in 2018 is using a HMD (head-mounted display), such as the HTC Vive Pro (Picture 1).

PICTURE 1. HTC Vive Pro HMD (HTC 2018)

2.2 The Rise of VR

Virtual reality is not an entirely new concept. In 1960, Morton Heilig invented the Telesphere Mask and it was the first example of a head-mounted display (HMD), albeit without any motion tracking (VRS 2017). Before that, inventors had been experimenting with different kinds of stereoscopic 3D displays and viewer devices since 1851, when David Brewster impressed Queen Victoria with his magical invention, the stereoscope (Fink 2018, 676). In 1968, MIT computer scientist Ivan Sutherland created the first HMD that was connected to a computer instead of a camera (Picture 2).



PICTURE 2. Sword of Damocles – a VR HMD from 1968 (VRS 2017)

However, the term virtual reality was not widely used before the late 1980s. This was when the first wave of enterprise VR computers and HMDs began to emerge. Back then the technology costed upwards of 40 000 dollars. The early 1990s was the era of first consumer headsets and public VR arcade machines. In 1995 Nintendo's Virtual Boy was marketed as the first ever portable VR gaming console. Technical difficulties and uncomfortable user experience were the reasons why most of the devices did not reach commercial success and were eventually discontinued. (VRS 2017)

For two decades, the VR industry stagnated, even though engineers and developers continued working on the technology out of the public eye. The advent of powerful smartphones with high-definition displays and 3D graphics capabilities finally enabled lightweight and affordable VR devices with impressive features. In 2011, Palmer Luckey created the Rift, a prototype VR headset, and soon after launched a successful Kickstarter campaign. This lead to Facebook purchasing his company Oculus in 2014 for 2 billion

dollars. The success of Oculus sparked a new era of VR: in 2016 multiple high-quality consumer devices were released, including the Oculus Rift consumer version and HTC Vive. (VRS 2017)

Since 2016, the global market for VR applications has grown steadily. Sony, HTC and Oculus have sold millions of units and together they make up about 86 percent of the global market share for high-end VR headsets. (TechCrunch 2017)

Various forecasts predict that the global VR market size will grow fast during the next few years. Consumer uptake of VR gear and content, such as games, drives the initial growth. Consumer use will be followed by the industrial uptake of VR technology and services. The market size for VR and AR (Augmented Reality) combined is predicted to grow 20-fold within the next 3 years, as illustrated in figure 1 (Statista 2018a).

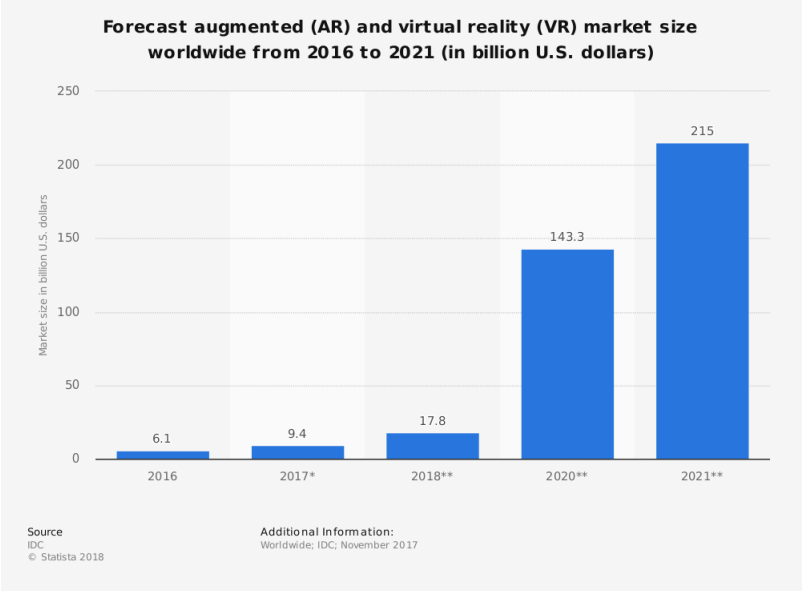


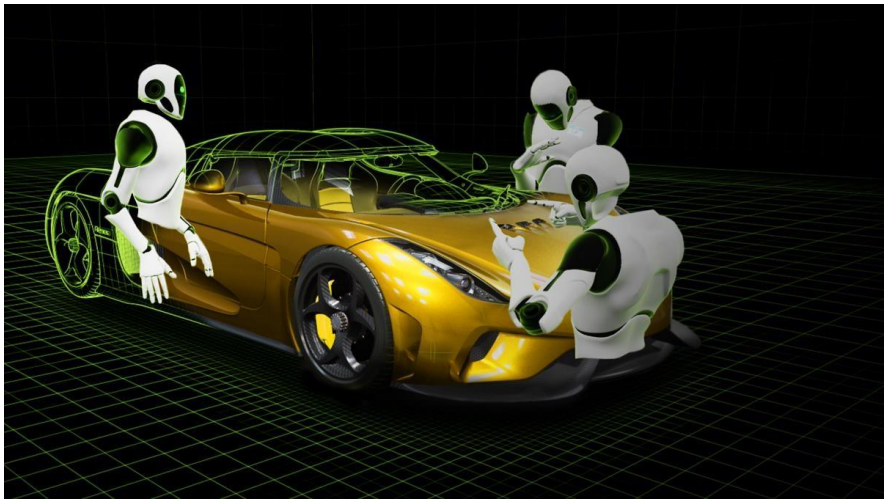
FIGURE 1. Market size forecast for VR and AR (Statista 2018a)

2.3 VR Enterprise Landscape

In 2017, the amount of VR companies in Europe went from 300 up to 487 between February and August. The growth has mainly been driven by startups creating applications for specific industry sectors. VR is already changing the business landscape in remarkable ways and specialists say that it will not take long before VR is integrated into day-to-day activities. (VentureBeat 2017)

The industries already utilizing VR include architectural visualization, simulation, product visualization, professional training, R&D, design, modeling, marketing and advertising. The VR industry is young and businesses are constantly experimenting with new diverse strategies to harness the medium and incorporate it into their business. “In every profession or trade, knowledge retention is a key issue in education and training. People retain 10% of the information they read, 20% of what that they hear but 90% when they physically interact with the subject matter” (Lawrence 2016). VR enables professionals to completely immerse themselves in the virtual world and learn new skills by training in environments that emulate the real world. Organizations can train their employees to perform complex tasks for a fraction of the cost and time.

Collaborative VR allows for virtual remote offices, where the company’s personnel can meet and interact in new and diverse ways. This will result in improved global collaboration between people and companies and lower costs for hiring employees. As more of everyday objects and tools will be connected to the internet, the sensor data can be used to create a fully immersive, shared experience where the monitor screen or smart phone is no longer needed. (Lawrence 2016) Large organizations such as NASA, Volkswagen and Nvidia are already exploring and creating collaborative VR platforms. Picture 3 demonstrates Nvidia’s collaborative platform Holodeck. (Weinstein 2017).



PICTURE 3. Nvidia’s collaborative VR platform Holodeck (Weinstein 2017)

2.4 Social VR: Enter the Metaverse

Palmer Luckey, the founder of Oculus, has declared that "virtual reality has never been about gaming. It's about the metaverse" (Johnson 2015). The term **metaverse** refers to a persistent collective virtual shared space, where people all over the world can connect and interact with each other. Physical realities converge seamlessly with virtual realities and this creates a virtual universe.

Social VR platforms are already being built. Facebook released the beta version of their social VR application Spaces in April 2017. In Spaces, users are represented by self-created avatars and they can spend time with their friends. The company behind the largest social media platform is working hard to bring Spaces to a wide array of VR platforms and devices. (Eadicicco 2017) Other popular social VR platforms include VRChat, AltspaceVR and Sansar (Kim 2018).

VR may enable platforms where people can connect in ways that are not possible in other online environments. The problem of feeling isolated in VR is best solved when people experience VR socially. Collaborative VR environments could spur a major change in how people interact with each other in social settings. (Park 2017)

2.5 VR Hardware

The global VR hardware market is currently experiencing rapid growth. In 2017, Oculus, HTC and Sony were the highest-grossing manufactures in the high-end VR market. Samsung's Gear VR and Google's Daydream View were the most sold mobile VR devices. The most sold VR headsets of 2017 in both high-end and mobile VR categories are listed in table 1. (Statista 2018b)

TABLE 1. Most sold VR headsets of 2017 (Statista 2018b)

MOST SOLD VR HEADSETS 2017							
Type	VR Headset	Units sold 2017	Resolution	Refresh Rate	Other	Motion Controller(s)	Headphones
High-End	Oculus Rift	850 000	1080 x 1200	90hz	6DoF	Yes	Yes
	HTC Vive	950 000	1080 x 1200	90hz	6DoF	Yes	No
	Sony Playstation VR	2 550 000	960 x 1080	120hz	6DoF	Yes	Yes
Mobile VR	Samsung Gear VR	3 650 000	(varies)	60hz	3DoF	No	No
	Google Daydream View	2 250 000	(varies)	60hz	3DoF	Yes	No

In 2018, there are numerous new and upcoming VR headsets with improved features. The unit sales for Windows Mixed Reality immersive headsets such as the Samsung Odyssey have been increasing steadily since the launch of the platform in late 2017. HTC released the Vive Pro, a follow-up to the original Vive, in April 2018, with better resolution and pixel density, integrated headphones and improved form factor. A wireless adapter is expected later this year. Upcoming high-end headsets include Varjo's HMD, that promises a high resolution Varjo Bionic display and Pimax 8K with a wide field of view and sharp resolution that helps in reducing virtual reality sickness.

Other new or upcoming VR headsets of 2018 are the standalone VR headsets Oculus Go, Oculus Santa Cruz, Lenovo Mirage Solo, Vive Focus and Pico Neo. Standalone headsets are wireless and have all the needed hardware integrated into the headset. Because there is no need for a powerful PC, complicated set-ups or external hardware, standalone VR technologies present numerous new opportunities for enterprise and collaborative VR applications.

2.6 Adverse Health Effects

Designing a comfortable VR experience requires paying attention to safety and health effects for the user. If a VR application ignores fundamental best practices, adverse health effects may occur and cause discomfort. Iterative user testing is key to creating a comfortable user experience. Following current industry guidelines is recommended, but as many aspects of VR are constantly changing and haven't been studied enough, experimentation is required. (Oculus 2018a)

Virtual reality sickness, also known as **simulator sickness**, is a subset of motion sickness, where the vestibular system's sense of real-world movement disagrees with what is perceived visually. The vestibular system is responsible for the sense of balance, spatial orientation and acceleration, based on inner ear fluid. The term **vection** describes the feeling of motion that is perceived visually, and if vection conflicts with the motion perceived in the user's vestibular system, simulator sickness ensues. As the vestibular system detects acceleration, virtual reality applications should be designed with little artificial acceleration in the visuals to prevent conflicting sensory information in the brain and thus to prevent nausea. **Optic flow** is all the visual motion in the scene that the user perceives.

More optic flow usually translates to more vection, which again might lead to VR sickness. Total amount of optic flow can be reduced with vehicle cockpits that stay still in the view of the user, or by reducing the visual information with narrower field of view, tunneling, fading and other techniques while in motion. (McCaffrey 2017, 184)

Adverse effects include the **vergence-accommodation conflict**. This effect is more or less unavoidable in the current generation of HMDs and is caused by the disparity between **vergence**, eyes targeting the physical screen surface, and **accommodation**, result of the focal point in the virtual world where the user is looking at. The vergence-accommodation conflict might induce nausea and headache for some users. (Jerald 2015, 173)

Low frame rates or dropped frames, latency, flickering images and low-resolution geometry are potential causes for eye strain, discomfort and VR sickness. A well-designed VR application takes performance and visual content into consideration to improve user experience and reduce adverse effects. (Oculus 2018a)

2.7 Locomotion in VR

The term locomotion describes the way of moving in the virtual world. Designing the locomotion method carefully to suit the experience is one of the core design principles in VR development. It is also one of the most difficult mechanisms to get right. Most locomotion types are still experimental, each with their advantages and disadvantages. Some induce less VR sickness, whereas others are more immersive and allow for greater freedom of movement.

The most natural way of moving is walking around within the range of the physical tracking area. Real-world movement is translated one-to-one to movement in the virtual scene. Natural locomotion methods are least likely to induce VR sickness as they reduce vection, but they are also heavily limited by the physical tracking area. (McCaffrey 2017, 185)

Teleportation usually combines natural locomotion with instant traveling from point A to point B. This allows for long distance movement in VR. Teleportation does not normally induce nausea, as there is no visible motion or acceleration. The disadvantage is that teleporting can be disorienting and immersion-breaking. (McCaffrey 2017, 185)

According to McCaffrey (2017, 186), **vehicle locomotion** can reduce the optic flow by encapsulating the user in stationary geometry and therefore reduce VR sickness. For the user it feels as though the world is moving around the user rather than the user moving through the world, which again is consistent with the vestibular system. The disadvantage in vehicle locomotion is that it is not suitable for many use cases.

Physical locomotion includes some of the most experimental locomotion methods. Examples include the user performing a skiing or jogging motion with the motion controllers to move forward. Because of the real-world physical action that translates into motion in VR, the vection caused by artificial movement is reduced as the user's vestibular system is able to anticipate it to some extent. This can be a fun mechanic for games, but for professional applications the experience can feel awkward and induce sickness. (McCaffrey 2017, 186)

Artificial locomotion methods do not rely on real-world movement or physical action. Instead, the user moves by using for example the thumbstick on a controller. This is referred to as passive movement and it can easily induce nausea, as the vestibular system disagrees with the movement that the user sees. For the developer artificial locomotion is usually straightforward to implement. (McCaffrey 2017, 187)

2.8 Optimizing Performance

VR rendering is a demanding process. Every frame must typically be rendered twice. This includes drawing the mesh and bounding the texture twice. (Oculus 2018b) The HMD display is very close to the eyes of the user and therefore the resolution needs to be high: the current display resolutions vary from 1920x1080 to 2880x1600 pixels (1440x1600 per eye). **Motion-to-photon time** refers to the time that the system takes from the detected motion to the user seeing it displayed on the screen. Ideally motion-to-photon time should be less than 20ms to avoid VR sickness. In practice this requirement means 90hz or even 120hz refresh rates. To make matters more complicated, the rendered resolution needs to be 1.5 times higher than the HMD native resolution to compensate for the apparent loss of resolution caused by barrel distortion, a technique that is used to compen-

sate for lens distortion caused by the Fresnel lenses of the HMD. 2880x1600x1.5 translates to over 600 million rendered pixels per second at 90hz refresh rate. This is why optimizing performance becomes an important aspect of designing a VR application. (McCaffrey 2017, 198)

There are general guidelines for optimizing performance for the current generation of hardware and software. Each frame should be limited to 500-1000 draw calls and a maximum of 2-3 million triangles if no advanced optimization techniques, such as clustering or batching, LODs (Level of Detail) and culling are used. By using geometry-optimizing techniques and algorithms the achieved polygon count may be much higher. Script execution should take 1-3 milliseconds at most. Performance-heavy post processing effects and shaders should be avoided. Some techniques such as instanced single pass rendering help improve performance by going through the render loop only once instead of twice. Scenes should be divided into smaller scenes and streamed in asynchronously. Throughout the development process it is advised to use performance profiling to mitigate possible bottlenecks. (Oculus 2018b)

3 RELEVANT TECHNOLOGY

In this section, the advantages and disadvantages of different game engines and networking solutions, as well as the underlying concepts, are discussed. This section also presents the principles of networking in multi-user applications.

3.1 Restrictions of Collaborative VR Development

Developing a networked, collaborative VR-application is very similar to developing a multiplayer online game (henceforth MOG). All MOGs utilize the Internet and mainly use the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) protocols for network communication. Developing a MOG requires knowledge of authoritative servers, cloud systems, databases, states and replication. (Ignatchenko 2017b, 6) Consequently, most of the same concepts and rules that are considered when developing and deploying a multiplayer online game apply to developing a networked multi-user VR-platform.

On the other hand, when developing a collaborative VR environment, most of the common pitfalls of online gaming, such as cheating, need not be considered. This makes designing the architecture simpler. For example, network traffic encryption is important when dealing with cheaters in games, but it is also hard to implement. (Ignatchenko 2017a, 107) The networking model also does not need to be fully authoritative, like in competitive online first-person-shooter games, as replicating the application state with millisecond precision is not required. As such, considering online multiplayer gaming, including VR gaming, is beyond the scope of this thesis and I will only examine the development of a collaborative, non-gaming VR platform. This thesis will not go through the fundamentals of VR development and instead the focus is solely on VR and networking.

3.2 The Engine Wars

Rendering immersive experiences in VR with high framerates is a complex task. The games industry has been solving complex rendering problems for over four decades and that makes commercial game engines well-suited for VR development. (Epic Games 2018a). There are numerous different game engines to choose from, so only the three most popular commercial game engines of 2017 are considered (Ignatchenko 2017b, 38). Only network- and VR -related features of the different engines are discussed. Everything else is beyond the scope of this chapter.

3.2.1 Unity

Not only is Unity arguably the most popular game engine among indie developers (Ignatchenko 2017b, 313), it is also the most widely used VR development platform (Unity 2018b). Unity provides a highly-optimized stereoscopic rendering pipeline that is optimal for VR. Unity supports all the major VR platforms, the documentation is extensive and the online community is vast. Unity can also be extended easily with 3rd party plugins and assets. (Unity 2018b)

Communication support in Unity, known as UNet, is split into two separate API levels: High-Level API (HLAPI) and Transport-Level API (LLAPI). HLAPI provides most functionalities needed in developing a networked application, whereas LLAPI is a low-level networking library. (Ignatchenko 2017b, 315)

3.2.2 Unreal Engine 4

Epic Games' Unreal Engine 4 competes in the same market as Unity. While Unity is the most popular game engine, Epic Games' CEO Tim Sweeney has stated that Unreal Engine's market share is "double the nearest competitor in revenue". In short, even though more games are developed with Unity, games developed with UE4 create more revenue. This is partly because UE4 is more geared towards AAA game development, meaning state-of-the-art games with high production value and big budgets. (Takahashi 2017)

While Unity caters to a wide range of needs, UE4 tends to focus on first- and third-person games, with great emphasis on graphics quality and performance (Ignatchenko 2017b, 333). Fortuitously, VR development benefits a lot from this approach. Since 2015, Epic Games has been investing heavily in Unreal Engine's VR capabilities and features. Popular VR titles such as Batham Arkham VR, Robo Recall and Farpoint have been created with UE4. Unreal Engine is also known for its networking features, showcased in for example the popular Battle-Royale mode of Fortnite. (Epic Games 2018a)

3.2.3 Amazon Lumberyard (beta)

In 2015, Amazon licensed Crytek's CryEngine software and developed a proprietary game engine called Lumberyard. Lumberyard features integration with Amazon Web Services (AWS) which allows developers to build and host dedicated servers for their applications. In Lumberyard, developers can include different Virtual Reality Gems to activate the integrated VR features. (Amazon 2018a)

The integration of Amazon GameLift, the dedicated game server hosting and matchmaking solution makes Lumberyard a solid candidate for building networked VR applications. The caveat with Lumberyard is that the license prevents you from using other leased servers or cloud hosting than AWS. (Amazon 2018b) Lumberyard is free to use and its business model revolves around income generated from AWS.

3.3 Networking Overview

The most important part of creating a collaborative VR environment is to make the users feel like they are in the same virtual world. When a person moves, others can see the same movement, and when a person throws a ball, others can see the flying ball. To make this possible, each computer client needs to construct a world state. If anything changes in the world, hosts exchange the necessary information to maintain the same world state on each host. In this section I will discuss the underlying architecture and protocols to make that possible. (Glazer & Madhav 2015, ch.6)

3.3.1 Network Topologies

There are two prominent networking topologies for building a multiplayer online game, or as in our case, a multi-user VR application: the client-server model and peer-to-peer model. In figure 2, the clients are connected using the client-server model.

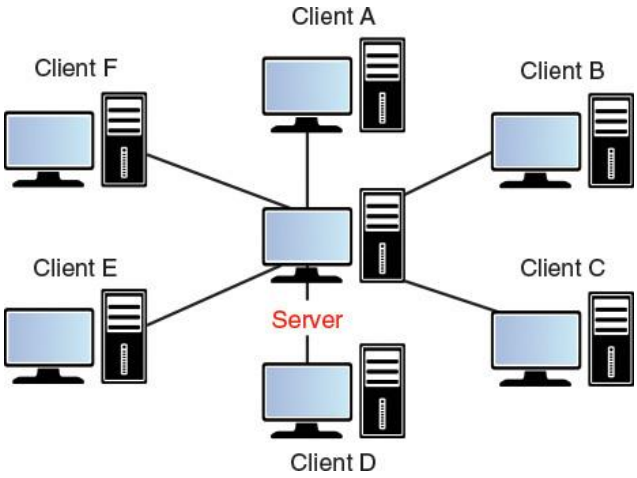


FIGURE 2. Client-server network topology (Glazer & Madhav 2015)

Most games and applications that are based on the client-server model utilize an authoritative server. This means that the server's state of the world is considered correct, and all communications are transferred via the server. The server processes requests from clients and if they are valid, the server sends the updated world state to the clients. This means that at any given moment the state of the world on a client is only an approximation of the "true" world state. (Glazer & Madhav 2015, ch.5) The client-server architecture has two sub-categories: some servers are dedicated servers, meaning they only process the game state and communicate with clients, and others are listen servers, where the server is also a participant in the game or application (Glazer & Madhav 2015, ch.5).

In a peer-to-peer topology, every host is connected to each other (figure 3). This makes managing authority more complicated. One option to synchronize game states is that every host sends their input to others, and each host updates the world state based on that. This usually requires the virtual world to be fully deterministic (all events happen in a mathematically precise and predictable manner), which is not trivial to achieve. In VR many interactions are physics-based, and physics are notoriously difficult to implement in a deterministic manner. Because of these limitations, peer-to-peer model is not very

common in most games or applications. Numerous networking libraries that erroneously market themselves as peer-to-peer, are not really peer-to-peer in the traditional sense. Instead, one peer acts as the master client and all communications are routed via the master client. This is essentially the same concept as in the listen server model. (Glazer & Madhav 2015, ch.5)

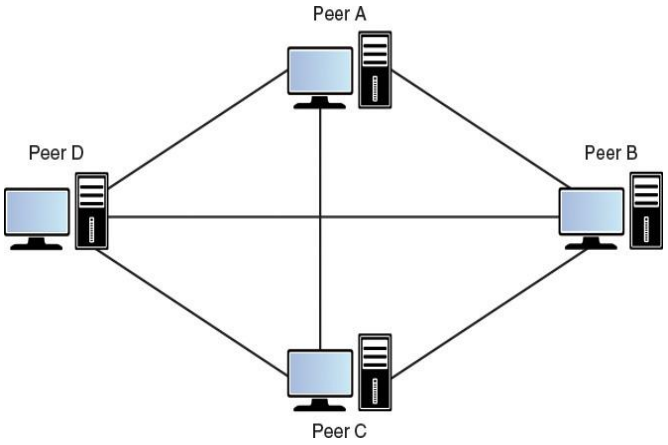


FIGURE 3. Peer-to-peer networking topology (Glazer & Madhav 2015)

3.3.2 TCP and UDP

TCP and UDP are both protocols that belong to the transport layer of the OSI model in computer networking. The transport layer and its ports makes communication between different hosts possible. UDP packets are light, fast and unreliable, and they do not have error- or flow-control. TCP requires stateful connections and guarantees reliability, which means that all data gets delivered and in correct order. Because of this TCP packets are slower and they require more bandwidth.

UDP's small footprint makes it the go-to protocol for synchronizing game states between hosts, especially when communications need to be fast, like in most FPS games. The TCP protocol is used e.g. for important events, such as connecting players and for reliable RCPs. Some libraries also utilize RUDP (reliable UDP) for reliable data transfer. (Mullins 2001)

3.3.3 State Replication

Glazer and Madhav (2015, ch.6) have defined that “the act of transmitting an object’s state from one host to another is known as replication”. Essentially replication usually means transferring the authoritative state contained in a server to all clients. First, a packet must be marked as containing an object state. Then it must be uniquely identified and its class declared. After the preparations the information of the object’s state is serialized from its RAM (Random Access Memory) format into a linear series of bits and sent over the network to all receiving clients. Usually the clients will update their world state based on the replicated delta value, i.e. the change in the object’s state. (Glazer & Madhav 2015, ch.6)

3.3.4 Remote Procedure Calls

Remote Procedure Calls, or RPCs, are remote function calls. Essentially a client can cause a procedure to execute on one or more remote clients. RPCs are needed when a client wants to transmit something else than object state to another client (Glazer & Madhav 2015, ch.6). This can be a specific event in the application, such as a sound effect, or a door opening, or a new user joining the session.

The RPC layer can usually be built on top of the object replication system. The networking libraries discussed in this thesis use non-blocking void RPCs, which means that the remotely callable functions have been specified signatures and they cannot return a value or throw an exception (Ignatchenko 2017a, 237).

3.3.5 Client-Side Prediction

In the authoritative server model clients send information, commands and RPCs to the server, the server verifies them and then sends the updated state of the world back to the client. This flow of information is illustrated in figure 4.

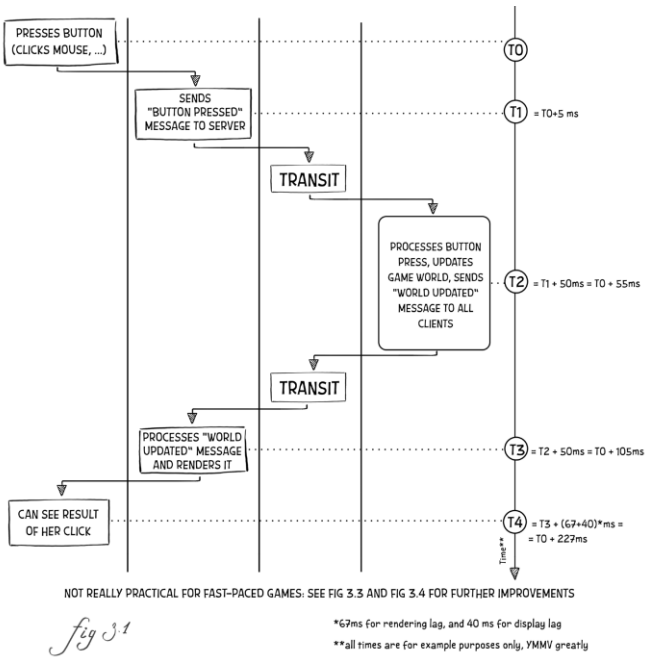


FIGURE 4. Data-flow in the authoritative server model (Ignatchenko 2017a)

The problem with this is that the network round-trip takes time and if there is any noticeable delay in between the user committing an action and seeing the results of that action, the experience will feel sluggish and laggy. According to numerous sources, the maximum input lag should be around under 100-200ms for the experience to feel responsive (Ignatchenko 2017a, 145). With **client-side prediction** the user client will start executing an action, for example moving an object, immediately when the button is pressed to negate the effect of the high-latency connection. Instead of waiting for the server to update the true state of the world, the client tries to predict the future state. When the server update eventually arrives, the possible desync is corrected by using a smoothing algorithm. This allows for the input of the user to feel more instantaneous. (Ignatchenko 2017a, 168)

3.4 Networking Libraries

3.4.1 Unity HLAPI and LLAPI

Unity's network communication system, also known as UNet, has two separate API layers: HLAPI (High-Level API) and LLAPI (Transport-Level API). HLAPI is for developers who prefer networking tools that work without custom algorithms. In HLAPI, there are two ways of replicating the application state across clients: state synchronization and

Remote Actions, Unity's version of RPCs. Variables can be synced from Server State to clients by using the **[SyncVar]** tag. This works for primitive types and because of this it is a rather limited way of replicating state. HLAPI has other shortcomings as well: for instance, Unity's synchronization methods lack proper bit compression. HLAPI does support interest management to some extent, but the developer needs to write custom logic for comprehensive prioritization. Compression and scoping algorithms usually save bandwidth and help reduce lag.

HLAPI does not provide lag compensation or client-side prediction methods out-of-the-box. Client-side prediction does affect the application's responsiveness somewhat, so the developer will likely have to implement it in some way. All in all, HLAPI might work well for prototyping and for smaller projects, but for more ambitious applications, a third party solution or custom network code will provide better performance. (Ignatchenko 2017b 315; Unity 2018a)

LLAPI is a low-level layer that provides only the very basic mechanisms for network communications. For instance, there is no support for RPC calls or state synchronizations. LLAPI is built on top of the UDP socket and only transmits low level messages. The advantage LLAPI is that it is built with performance and flexibility in mind and provides the developer with nearly all the networking possibilities – if they are implemented by the developer. (Ignatchenko 2017b, 312; Unity 2014)

3.4.2 Photon Engine

Of all the 3rd party networking libraries for Unity, Photon is the most popular (Ignatchenko 2017b, 320). Photon offers several products for different uses. They are split into two categories: Products based on Photon Realtime SDK (Software Development Kit) that utilize the Photon Cloud and the Photon Server SDK that requires a self-hosted standalone server. The higher-level SDKs for Photon Realtime SDK are called PUN, Bolt and Quantum. (Photon Engine 2018a) Because Quantum is currently not publicly available, it will not be considered in this chapter.

Photon Server is targeted for custom backends that run on self-hosted servers. Implementing custom server logic is possible and the SDK includes a load balancing framework for

high scalability as well as server-to-server communication for large projects. Photon Server does not provide state synchronization as-is, but instead sends a **BroadcastEvent()** to all connected peers. The clients will then need to manually manage these events. RPCs also need to be handled in a somewhat complicated manner: first you need to create a peer, then send a request using Photon.SocketServer.Rpc and finally register an operation handler for the response. This is a lot more involved process than RPCs in most libraries. (Ignatchenko 2017b, 322)

PUN features close Unity integration and it is the SDK of choice for many Unity developers. PUN is based on a client-to-server architecture and provides high scalability. PUN features serialization of game states, RPCs and matchmaking. (Photon Engine 2018b)

Photon Bolt is very similar to PUN, but it is a higher-level API with a few notable differences. Bolt provides state of the art bit compression and lag compensation, as well as automatic replication of states, all of which are features not included in PUN. Although this means less flexibility than what PUN offers, as you do not have total control of serialization routines like in PUN, it can be a great time saver. Bolt features NAT punch-through, which means clients can communicate directly with each other. Instead of RPCs, Bolt uses events to achieve the same functionality. Bolt also utilizes the Photon Relay servers to relay communications. Bolt is not a suitable library for building a fully authoritative server with custom server plugins and code, but for client-hosted applications it has many useful features. (Photon Engine 2018d)

3.4.3 Forge Networking

There are numerous other 3rd party networking libraries available for Unity: DarkRift 2, TNet 3 and SmartFoxServer to mention a few. One particularly interesting networking library is the open-sourced Forge Networking. The library includes RPCs, automatic state serialization for common types, NAT punch-through, relay servers, client-hosted servers as well as dedicated servers. Hosting and joining a server is a straightforward process. Forge is completely multi-threaded while preserving thread-safety at the application level. Forge also features advanced bit compression and other bandwidth-saving techniques, such as selective network variable updating. (Forge Powered 2018)

3.4.4 Unreal Engine Networking

“Networking is one of the many areas where Unreal Engine 4 shines” (McCaffrey 2017, **page**). On the surface, UE4’s networking capabilities are very similar to Unity’s HLAPI. Instead of Unity’s SyncVars and Remote Actions, Unreal provides state synchronization, i.e. **Actor** replication (Actor is the base game object class) and RPCs. (Ignatchenko 2017b, 334). In both UE4 and Unity the supported built-in network topology is the client-server model. In UE4, developers can focus on writing higher-level gameplay code that works in a networked environment and not worry about lower-level details. (Glazer & Madhav 2015, ch.11)

The main difference to Unity is that the developer has access to the full source code of the engine and thus all the low-level networking code. In UE4, it is possible to configure the rules for replication so that properties are only replicated when certain conditions are met. UE4 incorporates a powerful system for remote procedure calls by providing three types of RPCs: server, client, and multicast and they can be set to be either reliable or unreliable. Conveniently, UE4 provides client-side prediction that replicates and predicts actor movement on clients based on the replicated actor’s velocity. (Glazer & Madhav 2015, ch.11; Neukirchen 2017)

3.4.5 Amazon GridMate

Amazon’s networking technology in Lumberyard is known as GridMate and it resembles the previously discussed higher-level architectures, such as HLAPI and Unreal Networking. GridMate implements state synchronization and void RPCs. Synchronized objects and states are called replicas and RPCs are primarily used to modify their state.

What makes Amazon Gridmate stand out of the crowd is GameLift, a tight integration with Amazon’s cloud service for games. GameLift facilitates creating new cloud instances, load-balancing server traffic, and optimizing performance geographically. (Ignatchenko 2017b, 341) Overall, Gridmate provides powerful networking tools and features, but still lacks support and documentation compared to other libraries.

4 DESIGNING THE SOFTWARE FRAMEWORK

The collection of the required components and applications to enable the development of a project can be considered as the software framework. In this section, the different framework options for developing a collaborative VR application are assessed and evaluated in practice. This chapter also provides reasoning for the choice of the game engine platform, networking library and networking architecture that together form the software framework.

4.1 Application Requirements

Depending on the scope of the project and the amount of people working on it, there are many factors to consider when choosing the game engine, networking library and server architecture. Are the developers familiar with the development platform and programming language? Is there a need for cloud hosting services? Will the application run on a dedicated server or as listen server? Is decoupling of the client and the server in a clear way needed or can development be approached from an engine-provided server perspective? Load balancing, server-to-server communication and bandwidth usage are important considerations if there are a great number of concurrent users. Some solutions scale better and have more robust networking features so that the platform does not collapse under heavy traffic, but they take more time to implement. (Ignatchenko 2017b, 320)

4.2 VR Setup on Different Platforms

As discussed in chapter 3, modern game engines act as the basis for VR development. In this subsection configuring the different game engines for standard VR development is examined in detail and comparisons are made.

Unreal Engine 4

Getting started with VR development in UE4 is very straightforward. UE4 offers a VR Template Project that includes a VR Pawn (in UE4 a Pawn is an Actor controlled by the user) with motion controller and HMD tracking, rendering optimizations and ready-made functions such as teleportation and grabbing objects. The VR template supports HTC Vive, Oculus Rift and Playstation VR. In UE4 external plugins for setting up VR are not needed, as most of the VR SDKs such as **OpenVR**, **SteamVR** and **Oculus SDK** are built into the engine and are enabled by default. (Looman 2016) Compared to other platforms UE4 provides the fastest way of setting up basic VR functionality.

Unity

VR setup in Unity is slightly more complicated than in UE4. A common way to set up VR is to integrate the SteamVR SDK from the Unity Asset Store into the project (Van de Kerchove, 2016). At the time of writing this thesis, Unity's abstraction layer for motion controller inputs is not as comprehensive and easy-to-use as in UE4, so to access all motion controller functionalities such as haptic feedback, the SteamVR controller classes were utilized. Although Unity does not include a template for VR interactions or locomotion, the SteamVR plugin includes mechanics for teleportation and basic interactions in VR. Many VR developers also utilize **VRTK**, a VR toolkit for rapid prototyping and development (VRTK 2018).

Lumberyard

Configuring Lumberyard for VR development includes adding one or more Virtual Reality Gems to new or existing projects from the Project Configurator. Lumberyard also offers a Virtual Reality Samples project that demonstrates most common VR functionalities and how a VR project should be configured. (Amazon 2018c) Unfortunately, that is the extent of support currently available for VR development in Lumberyard. Amazon’s documentation only describes rudimentary VR configuration and VR Lua Functions. As of writing this thesis, an online support community for Lumberyard VR development does not exist and there are few showcases available.

Researching VR development in Lumberyard in more detail revealed several critical issues. The Virtual Reality Samples Project crashed on all tested computers with a fresh install of Lumberyard, even though other sample projects could be opened without issues. The VR rendering performance was tested with a basic scene without any complex 3D

models using an Oculus Rift and this resulted in constant frame drops and low overall framerate. There were also some mismatches in rendering geometry to both eyes. On top of performance problems, Lumberyard suffered from motion controller configuration issues. Even though the integration with AWS GameLift offers great potential for collaborative VR, based on this research Lumberyard is not a suitable platform for professional VR development yet. Therefore, the networking evaluation and experiments will only focus on Unity and UE4.

4.3 Examining the Unreal Engine 4 Networking Architecture

Unlike Unity, UE4 is very specific about the application architecture. UE4 features sample projects that demonstrate how the different models should be used in development. One of them is **Multiplayer Shootout**, which is a simple blueprint multiplayer showcase. The sample project is built with blueprint visual scripting only and it establishes how session nodes are used in creating a networked multiplayer game. (Epic Games 2018b) In researching the networking capabilities of UE4, a sample project was constructed using a similar architecture as in the Multiplayer Shootout project.

The architecture was divided into two different branches: the server-side and client-side. In the server-side logic, the GameInstance blueprint class creates a session and opens the correct level, as shown in figure 5. When the level has loaded the GameMode class receives an OnPostLogin callback for setting up the session and users. A **switch on authority** node determines if the GameMode class is running on server or client, and proceeding actions are taken accordingly. On the client host, the GameInstance class finds a list of active sessions and joins the selected session. After that, the GameMode receives the same OnPostLogin callback as on the server, and from there continues to do the client-side setup routines, such as calling the ClientPostLogin function in PlayerController and RespawnPlayerEvent in the GameMode class. (Epic Games 2018b) The networking setup described above allows two hosts to connect to each other in a local area network or by utilizing a networking subsystem such as Steam, Playstation Network, or Xbox Live. (Helios Interactive 2016)

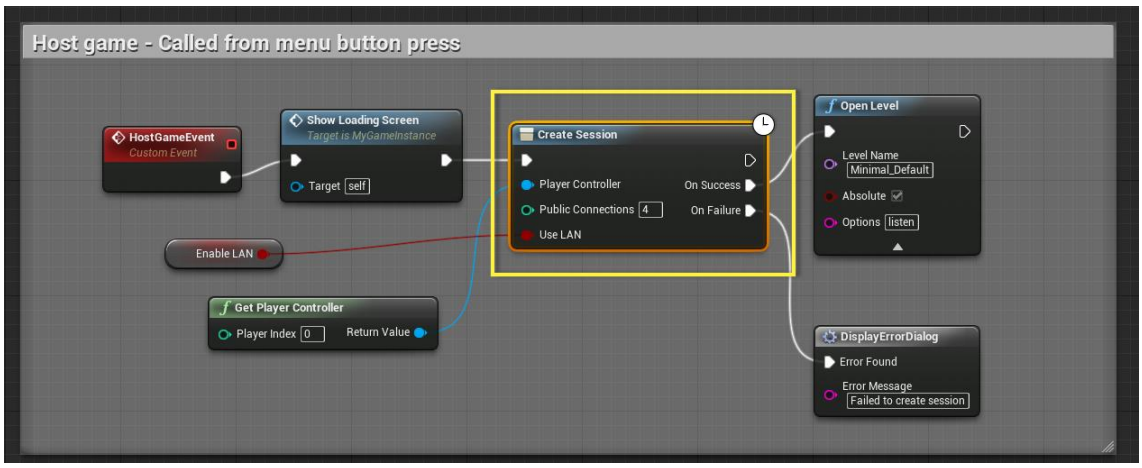


FIGURE 5. Blueprint function for creating a session (Epic Games 2018b)

UE4 normally provides automatic replication functions for Pawn classes. However, the experiments revealed that Epic Games have not yet implemented automatic replication of the motion controllers and the HMD in UE4. Implementing this can be cumbersome and when testing different ways of replicating them correctly, several problems ensued: either the client spawned too many motion controllers or the client did not get the position and rotation updates correctly. When the client finally received correct replicated values for the motion controllers, the server host did not see the updates of the client. When researching a solution, a multiplayer VR template project for UE4 named Proteus VR Template was found on Github. However, at the time of the research, the template project version 3.1 presented similar problems for replicating the HMD and motion controllers and therefore it did not solve the issue. Eventually, a working solution was found by utilizing the **VRExpansionPlugin** by the developer Mordentral. The plugin is intended to ease the creation of VR games and experiences in UE4 and it provides automatic replication, interpolation and client-side prediction for the relevant VR components and actors. The plugin also provides numerous other useful features for any VR development. (Mordentral 2018) Accompanied with this plugin, the Unreal networking framework is well suited for collaborative VR development.

4.4 Comparison of Unity Networking Libraries

As mentioned in subsection 3.4.1, Unity comes with its own networking library UNet. UNet does not support NAT punch-through and instead uses relay servers to connect clients to each other. This increases the possible latency in the network drastically, as each client might go through several relay relays before reaching the host. (Unity 2017) Also

the synchronization mechanisms UNet provides are not very efficient in optimizing traffic. This is mostly because UNet lacks some advanced compression methods, such as incremental delta compression and restricting precision of state variables (Ignatchenko 2017b). Due to these reasons, the network performance of UNet does not compare to Photon Bolt and Forge, which provide NAT punch-through and optimized bit compression techniques (Unity 2017, Forge Powered 2018). If the application has a high number of concurrent users, the pricing of bandwidth usage in UNet will be significantly higher than in Photon Bolt or Forge (Unity 2017).

Photon PUN lacks numerous features that are implemented in Photon Bolt, such as NAT punch-through, automatic replication of variable states, advanced bit compression, client-side prediction and host migration (Photon Engine 2018d). Comparison of networking features of different libraries are presented in table 2.

TABLE 2. Networking Features in Different Libraries

Networking Features in Different Libraries

Feature	UNet	UE4	Bolt	PUN	GridMate	Forge
LAN	✓	✓	✓	~	✓	✓
NAT Punch-Through	✗	✗	✓	✗	✓	✓
Automatic Replication	~	✓	✓	✗	✓	~
Bit Compression	✗	✓	✓	✗	✓	✓
Interest Management	✗	✓	✓	✓	✓	✗
Client-side Prediction	✗	✓	✓	✗	✓	✓
Room / Lobby support	✓	✓	✓	✓	✓	✓
RPC's	✓	✓	~	✓	✓	✓
Host Migration	✓	✓	✓	✗	~	✓
Encryption	✗	✗	✗	✗	✓	✗

✓ Implemented ~ Partially implemented ✗ Not built-in or needs configuring

4.5 Testing During Development

In multi-user VR development, the developer cannot launch and test two clients on the same computer. This is because HMDs such as HTC Vive and Oculus Rift cannot be used by two applications at once because the performance drops very low. For practical multi-

user VR application development two computers, two sets of VR hardware and two developers are required. One drawback in using UE4 for creating a collaborative VR application is that multi-user applications cannot be run and tested from the UE4 editor, as this creates a map version mismatch. The solution is to utilize the Unreal Frontend to build and deploy the project for different machines. (Helios Interactive 2016) This is an involved process and requires a great deal of setting up. The other option is to package the project and manually transfer it to different machines for testing. Unfortunately, this process makes collaborative VR development in UE4 much slower and less agile than in Unity, which allows networked VR applications to be run and tested straight from the editor.

Unity also offers Collab, a built-in version control system. As multi-user VR development requires two computers, syncing the project with Collab and running the application from the editor without the need to build it is a significant advantage in Unity's favour. In projects with limited budgets and timelines, this feature becomes an important factor when considering the development platform.

4.6 Online Subsystems

The other major obstacle in using Unreal is that most of the viable online subsystems are meant to be used with games only. For example, the Steam subsystem requires a SteamAppId, which can only be acquired by releasing the game or app on Steam. For testing purposes, the official testing id 480 can used and this makes the app run as "SpaceWar". (Epic Games 2018b) For private enterprise applications using the Steam subsystem is not therefore a viable option. To circumvent this, the application would have to run on a dedicated master server that clients can connect to. Compared to the client-hosted listen server architecture that Photon Bolt and Forge offer, this is a much more complex model and normally requires a separate backend team for set up. Unity Multiplayer Services, Photon Cloud and Forge Cloud all provide online systems for both games and other applications.

4.7 Summary

Because of the lack of options for online connectivity and the added development time, creating enterprise multi-user VR applications with UE4 was not considered a viable option this time. The testing process became time consuming due to the lack of editor testing functionality and the project timeline would have extended too much. A practical solution for implementing an online subsystem for enterprise use without the need for a dedicated server and cloud hosting was not found during this research.

At the time of writing, most VR projects at Intopalo have been developed with Unity. This coupled with the fact that this project was a direct continuation to a previously made Unity VR application made Unity a natural choice for this thesis project. A dedicated server was not a requirement and building one would have impacted the scope. Instead, a client-hosted listen server architecture networking model was chosen. UE4 Networking does not support features like NAT punch-through natively, which means that to support communications over the internet, a dedicated server, preferably on a cloud hosting service such as AWS, would be required. The other option would be to use an online subsystem like Steam and utilize the listen-server model. However, for a non-gaming application this is not the optimal choice, as an official Steam App Id can only be acquired for games and other Steam applications.

Because of the advantages of Photon Bolt's compression algorithms, automatic state replication, lag compensation and most importantly support for client-hosted networking, Bolt was chosen as the networking framework. The research suggests that feature-wise Forge would have been a suitable and valid option as well.

SteamVR SDK for Unity was used as the toolkit for standard VR functionality. SteamVR provides developers with numerous advanced tools for VR development and allows developers to target a platform that works with almost every popular headset. Even though HTC Vive was this project's targeted hardware, using SteamVR makes it possible to use the application with other headsets as well. As shown in this chapter, the design of the software framework for collaborative VR development depends on numerous factors and requirements and as such must be evaluated on a case-by-case basis.

5 NETWORK DEVELOPMENT

In this section, the development techniques for creating a collaborative VR platform using Unity and Photon Bolt are examined and examples are given. The network architecture is analyzed and the most important considerations for VR networking are studied.

5.1 Bolt Network Topology

The network session can be set up so that the first client that connects becomes the master client for the session and creates a room where other users can connect to. It is also possible to choose which client creates the listen server. The latter approach was used in this project. Bolt's session discovery allows clients to find servers running on a Local Area Network (LAN). The network topology in LAN is uncomplicated: all clients can communicate directly with the master client, as visualized in figure 6.

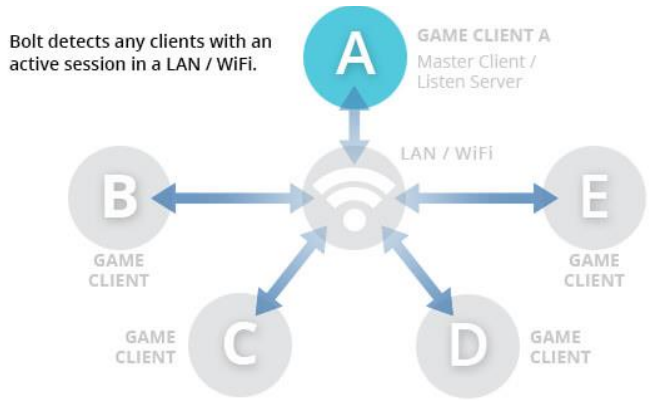


FIGURE 6. Bolt network topology in LAN (Photon Engine 2018c)

When the clients are connecting with the master client through the internet, Bolt uses **NAT punch-through** to connect the clients. NAT is short for network address translation. Routers use NAT to map addresses behind the router to a single public IP (internet protocol) address. The problem with NAT is that if two computers behind a router try to connect to each other, neither will connect, because no mapping exists yet. As shown in figure 7, Bolt's NAT punch-through technology allows the clients to connect to each other. (Photon Engine 2018c)

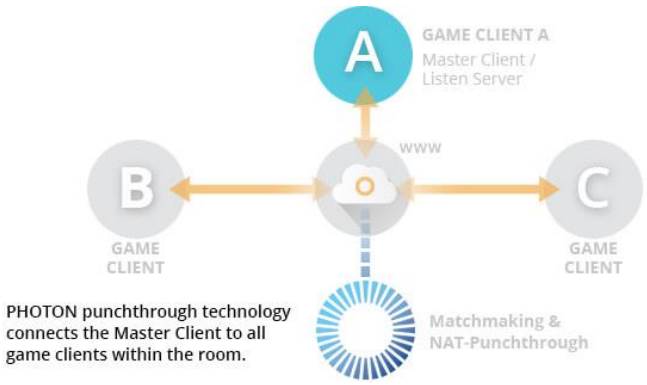


FIGURE 7. NAT-Punchthrough in Bolt (Photon Engine 2018c)

If the clients are not able to connect directly to each other or it would slow down the connection, the connection is relayed via Photon Cloud, Photon's global relay network. Figure 8 illustrates this topology. (Photon Engine 2018c)

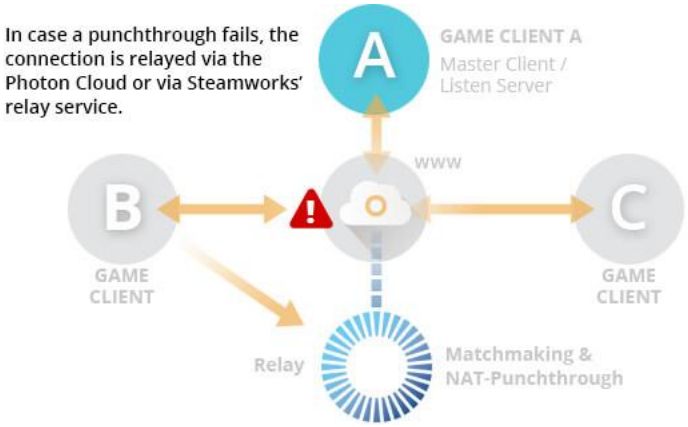


FIGURE 8. Relaying connection via Photon Cloud (Photon Engine 2018c)

5.2 Bolt Assets

The installation of the Bolt plugin creates a new editor menu in Unity. Under Window/Bolt/Assets there is a Bolt Assets window containing everything that Bolt replicates (figure 9).

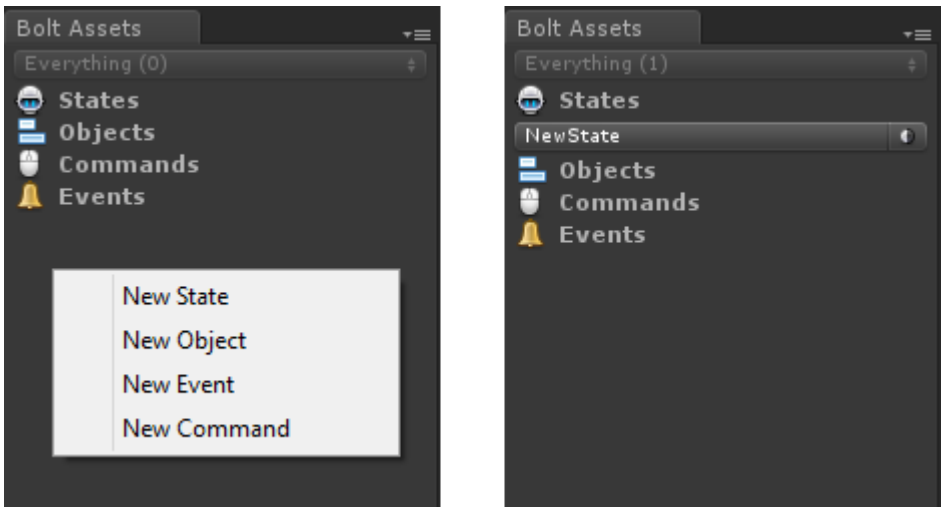


FIGURE 9. Bolt assets window

All assets belong to one of four categories: States, Objects, Commands and Events. The content of the replicated data, including transforms, strings, floats etc. is defined in a State. Objects are data structures consisting of sub-items that can be used in the State object. Commands are used to send input data to the authoritative host for updating the state of the world. Commands are essential in the implementation of client-side prediction and correction of predicted state. Events are comparable to traditional RPCs. (Photon Engine 2018f)

5.3 Creating a Session

As discussed in chapter 4.6, Bolt uses the client-server network model, which means that one computer acts as the listen server and everyone else as clients. All network traffic is routed via the server host, either directly from computer to computer with NAT punch-through or via the Bolt relay servers. As depicted in figure 10, the clients only know about their connection to the server client, and conversely, the server holds the information of the connections to the clients.

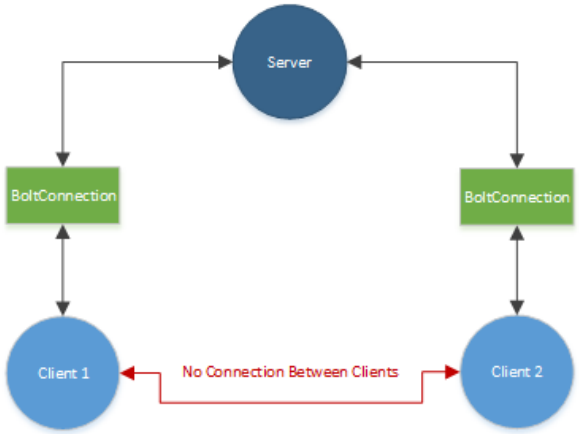


FIGURE 10. Bolt's Client-Server networking model

A server is created by calling the **StartServer()** method in the **BoltLauncher** class. Conversely, a client will be initialized with the **StartClient()** method, as shown in figure 11.

```
public void CreateServer()
{
    BoltLauncher.StartServer(new UdpEndPoint(UdpIPv4Address.Any, (ushort)_serverPort));
}

public void JoinServer()
{
    BoltLauncher.StartClient(UdpEndPoint.Any);
}
```

FIGURE 11. Methods for creating and joining a server

After Bolt has established a connection, a **BoltStartDone()** callback method is called and the main scene is loaded. As shown in figure 12, for LAN connections to work, the **EnableLanBroadcast()** method must be called.

```
[BoltGlobalBehaviour(BoltNetworkModes.Host)]
public class ServerCallbacks : Bolt.GlobalEventListener
{
    public override void BoltStartDone()
    {
        BoltNetwork.LoadScene("Main");
        BoltNetwork.EnableLanBroadcast();
        BoltNetwork.SetHostInfo("Intopalo", null);
    }
```

FIGURE 12. Loading the main scene in the BoltStartDone() callback method

5.4 BoltEntity

BoltEntity is central to the Bolt networking library. A BoltEntity is a Unity **GameObject** (a base class for Unity scene entities) that has the BoltEntity.cs script component added to it. By adding the BoltEntity component, the game object can serialize and synchronize state over the network automatically. To synchronize state, a **State** object must be created in Bolt's Editor Window. (Bolt Cheat Sheet 2018; Photon Engine 2018e)

A BoltEntity should be instantiated at runtime using the **BoltNetwork.Instantiate()** method. By making the GameObject a Unity **Prefab** (an asset class and template), the entity to be instantiated can be selected using a **BoltPrefabs.<EntityId>** parameter. (Bolt Cheat Sheet 2018; Photon Engine 2018e) In the example below, the user's VR avatar prefab is instantiated in the **NetworkCallBacks** class (figure 13). This results in each user instantiating an avatar entity that they are an Owner of.

```
[BoltGlobalBehaviour]
public class NetworkCallbacks : Bolt.GlobalEventListener
{
    public override void SceneLoadLocalDone(string map)
    {
        BoltNetwork.Instantiate(BoltPrefabs.NetworkPlayer);
    }
```

FIGURE 13. Instantiating a BoltEntity

5.5 Owner Controls State

A BoltEntity always has an **Owner** (entity.isOwner == true). This is the host that instantiated the BoltEntity prefab in the network. The Owner has full control of the entity State and ownership cannot be transferred or assigned to another host. The Owner has a **SimulateOwner()** method that only runs on the Owner entity. (Bolt Cheat Sheet 2018; Photon Engine 2018e)

In authoritative networking topologies, the server is usually the Owner for all objects. In games, this usually includes the player character. However, in a non-competitive collaborative application, ownership of the user entities can be given to the users themselves.

5.6 Syncing Motion Controller and HMD Transforms

The SimulateOwner() method is the best place to perform movement updates for the VR avatars in the network. Because a collaborative application does not require a fully authoritative server, we can let the users have authority of their own movement. As the position and rotation of the motion controllers and HMD are locked to their position and rotation in the real world, we cannot update their positions directly. Instead, proxy transforms are created. Proxy transforms are just empty GameObjects replicating the position and rotation of the motion controllers and HMD. At first, we inform Bolt in the Attached() method that these transforms should be set to replicate automatically in the network (figure 14).

```
public override void Attached()
{
    state.SetTransforms(state.HeadProxyTransform, _headProxyTf);
    state.SetTransforms(state.HandLProxyTransform, _handLProxyTf);
    state.SetTransforms(state.HandRProxyTransform, _handRProxyTf);
}
```

FIGURE 14. Assigning the proxy transforms to the Bolt state object

After setting the transforms in the Bolt state object, we update the position and rotation of the proxy transforms to represent the position and rotation of the user's motion controllers and HMD. In this project, the InputTracking class of Unity was used to get the position and rotation of the motion controllers and HMD (figure 15).

```
public override void SimulateOwner()
{
    _headProxyTf.rotation = InputTracking.GetLocalRotation(XRNode.Head);
    _headProxyTf.position = InputTracking.GetLocalPosition(XRNode.Head);

    _handLProxyTf.rotation = InputTracking.GetLocalRotation(XRNode.LeftHand);
    _handLProxyTf.position = InputTracking.GetLocalPosition(XRNode.LeftHand);

    _handRProxyTf.rotation = InputTracking.GetLocalRotation(XRNode.RightHand);
    _handRProxyTf.position = InputTracking.GetLocalPosition(XRNode.RightHand);
}
```

FIGURE 15. Updating the position and rotation of the proxy transforms

5.7 Controller and Proxy

A BoltEntity can also have a **Controller** (entity.hasControl == true). This is a person who has been assigned control of an entity by the Owner. The Controller can control the entity by calling a **QueueInput()** method inside a **SimulateController()** method, which is used for sending commands to the Owner of the entity. The Owner then verifies the queued commands and performs the operations inside the **ExecuteCommand()** method on both the Owner itself and the Controller. On the Owner, the method only runs once for each command. On the Controller (if it is not the owner also), ExecuteCommand() behaves differently: it handles both the client-side prediction and the state update from the owner. This happens by first executing the last verified state from the owner, and then running all commands not yet verified by the owner. In the following example, the position change of a Unity GameObject is sent from the Controller. If the **resetState** Boolean is **true**, Bolt sets the local state as it has been received from the Owner. Using **cmd.Result** instead of **cmd.Input** allows Bolt to correct the local movement. If resetState is false, ExecuteCommand() will update the object position with the non-verified position updates. This key mechanic of Bolt enables e.g. lag compensation / client-side prediction. Figure 16 illustrates the correct use of SimulateController() and ExecuteCommand() methods.

```
public override void SimulateController()
{
    IModelCommandInput input = ModelCommand.Create();
    input.BodyPos = _bodyAttributes.Position;
    entity.QueueInput(input);
}

public override void ExecuteCommand(Bolt.Command command, bool resetState)
{
    ModelCommand cmd = (ModelCommand) command;

    if (resetState)
    {
        _bodyTf.position = cmd.Result.BodyPos;
    }
    else
    {
        _bodyTf.position = cmd.Input.BodyPos;
        cmd.Result.BodyPos = _bodyTf.position;
    }
}
```

FIGURE 16. Sending input commands from Controller to Owner

The Photon Bolt documentation (Photon Engine 2018f) states that incorrectly/incompletely resetting state is one of the most common problems with jittery movement on the Controller's side. Therefore, care must be taken when implementing this behavior.

The Proxy is a host that is neither the Owner or the Controller of the entity. The only callback method which the Proxy executes is **SimulateProxy()** and it is only used on certain edge-case scenarios. Bolt updates the entity state for proxies automatically. (Bolt Cheat Sheet 2018; Photon Engine 2018e)

5.8 Events

Events are a way of implementing RPCs in Bolt. Events are divided into global events and entity events. Global events can be either reliable or unreliable, however in general global events are usually set as reliable. Entity events are always unreliable and they are intended for effects and other cosmetic actions that are not essential to synchronizing the world state between users. Events are created in the Bolt Editor and then Bolt compiles each event into a class that can be instantiated at runtime. Each event can have a set of arguments that are sent with the event. In the following code example, an event for changing the color of an object is created and sent over the network (figure 17).

```
var evnt = ChangeColorEvent.Create();
evnt.NewColor = new Color(Random.value, Random.value, Random.value);
evnt.Send();
```

FIGURE 17. Creating and sending an event

Global events are received by everyone in the network. A class needs to inherit from the Bolt.GlobalEventListener class to be able to execute the OnEvent() callback methods. The event object parameter holds all the sent information that can be used, like in the following code snippet (figure 18).

```
public override void OnEvent(ChangeColorEvent evnt)
{
    bgColor = evnt.NewColor;
}
```

FIGURE 18. A global event is received in the OnEvent() callback method

5.9 Voice Chat

Implementing voice chat is an important feature in a collaborative VR application. Currently there are numerous ways to create a voice chat system. One of them is a third-party library Photon Voice. Photon Voice is a free voice chat library that can be integrated with the Photon PUN and Bolt networking libraries and it uses the Photon Cloud for relaying voice data. (Photon Engine 2018a) Another option is Dissonance Voice Chat, which works with UNet, Forge and Photon networking. Audio quality, voice positioning and speaker priority are features that help in creating an immersive VR experience.

6 CASE: PLANMECA

In this section the implementation of a collaborative VR application for Planmeca Oy is studied and explained. This section describes the design choices and patterns that were used and aims to give insight into the design and development process.

6.1 Inverse Kinematics for Rigged Avatars

One way or another, the bodies of the users need to be represented in VR. McCaffrey (2017, ch.7) explains that replicating the users' real-world movement to their VR characters, or **avatars**, can improve immersion significantly if done right. There are different approaches to this. The simplest solution is to completely separate the head and hand objects and map their positions and rotations to the HMD and motion controller positions and rotations. This results in a floating head and floating hands that represent the user. Though easy to implement, it is also the least realistic and immersive approach.

Another approach is to **rig** the avatar model and in a modeling software (figure 19) and use **inverse kinematics** (IK) to let the system interpolate the correct pose based on the known information of the HMD and motion controller positions and rotations. When rigging the model, **armatures** are used to create the flexible joints for the mesh objects. In some cases, it is better not to model and rig legs for the avatar, as solving the correct position of the virtual legs without actually knowing their position in the real world is difficult to get right. In this case, the full body avatar was specified in the application requirements, and therefore a commercial IK solver plugin Final IK was used to produce accurate movements for the user's avatar.

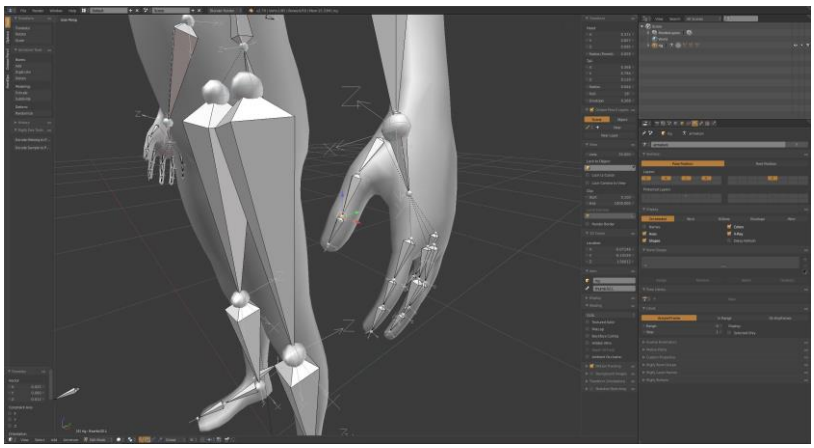
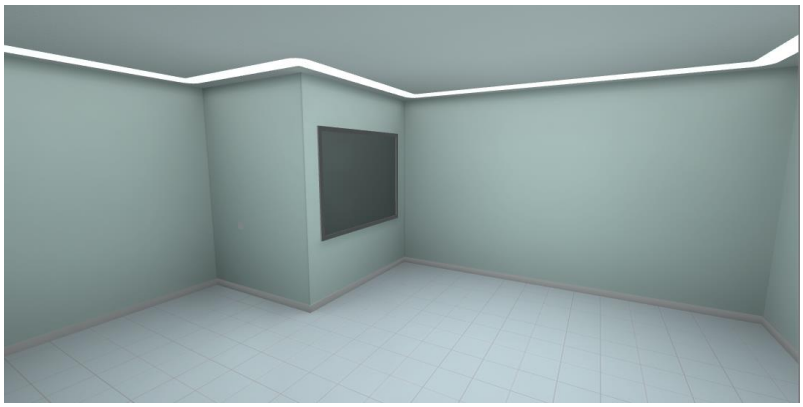


FIGURE 19. Rigging the avatar model in Blender

6.2 Environment Design

When designing VR environments, the scale of spaces and objects need to match their real-world counterparts or otherwise the experience does not feel realistic. This might be an intended effect for games and VR entertainment, whereas for professional applications realism is usually desired.

The requirements of the customer specified that the virtual environment should be well-lit with ambient light. Real-time lighting in VR consumes a lot of resources, which is why emissive materials and baked lighting were mainly used for this project. The chosen colors were similar to colors used in healthcare design to promote tranquility and to simulate real-world environments where the customer's products are used (picture 4).

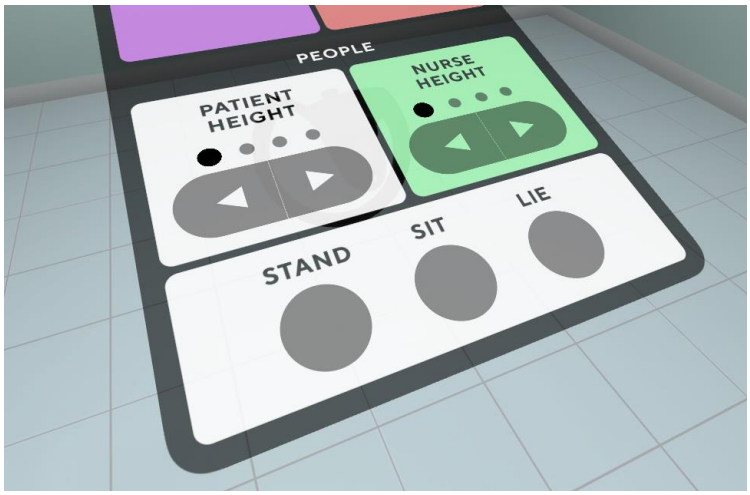


PICTURE 4. Room and lighting

6.3 User Interface and Interaction

Traditional two-dimensional UI (User Interface) design does not really translate to VR applications. In first person games, many UIs are head-locked, so when the camera moves, the UI moves with it. In VR, looking at an orthographically projected UI that is layered on top of everything else creates a conflict in the focus of the user's vision, leading to possible discomfort. (McCaffrey 2017, 108) Having the UI constantly in sight also blocks the user's view of the virtual world and might break immersion.

A common way of implementing UI in VR is to have a three-dimensional object, usually a quad, onto which the two-dimensional UI is projected (McCaffrey 2017, 108). The user can then use motion controllers to point and interact with the UI in the same way as with traditional UIs. In this project, an external plugin **CurvedUI** was used for this. The library's **CurvedUIInputModule** class provides a custom **Ray** object that can be used for interacting with native Unity UI canvas components, provided that the canvas is first initialized with the CurvedUI plugin. Instead of having the UI object stationary in the world, it was parented to the left motion controller. This gives users complete control of the position of the UI and they can comfortably use the interactions with their right hand. This design pattern is known as **the tool palette** and it has been used in applications such as Google Tilt Brush and Oculus Medium and Quill (Shahar 2017). Picture 7 demonstrates the use of the tool palette UI in this project.



PICTURE 5. The tool palette UI

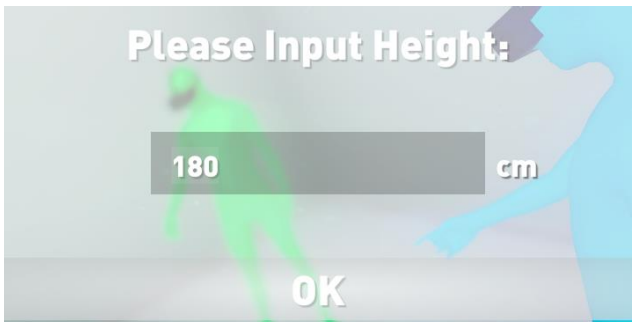
Another concept of interaction design in VR is **affordances**. Affordances describe how an object can be used, i.e. the relationship of the properties of the object and our ability

to act on it. If there is a button, the user can push it, and if there is a lever, the user can pull it. Because of the three-dimensionality and increased immersion of VR, many interactions that have traditionally been placed in the UI can now be embedded into the virtual world. (McCaffrey 2017, 108) In this project, the pose and position of the nurse and patient models in the application can be modified by grabbing one of the transparent interaction spheres and by moving it to a desired position (picture 6).

PICTURE 6. Interactable joints

6.4 Avatar Height

If the user's avatar size is not scaled to match the height of the user, there will be complications in their virtual representations. Because of the way the character rig and IK were set up, tall users looked like they were standing on their toes and constantly reaching, and conversely, short users looked like they were crouching. This is because the IK solver only knows the Y-position, or height, of the HMD, and tries to figure out the correct positions for the bones of the default avatar. If the default avatar height is 180 cm and the user's height is 160 cm, the IK solver will cause the knees of the avatar to bend. There are several ways of addressing this. One option would be to automatically detect the height of the HMD when it is placed on the user's head and scale the avatar size accordingly. However, if the HMD is replaced while sitting, the resulting height will be incorrect. Therefore, it was decided that the ideal approach for this project was to make height an adjustable setting in the main menu, as shown in picture 7.



PICTURE 7. Setting for height

6.5 Avatar Customization

In Bolt, customizing the avatar properties is optimally done by utilizing the IProtocolToken interface. In this case, the only customization options that were needed were the avatar color and height. The inherited token class is activated with the BoltNetwork.RegisterTokenClass<NameOfTokenClass>() method. The token object can then be used as an argument in the BoltNetwork.Instantiate() method. When the avatar prefab is attached in the Bolt network, custom settings and values can be derived from the token object. Figure 20 demonstrates how a customized token class can be created.

```
public class AvatarCustomization : Bolt.IProtocolToken
{
    public Color charColor;
    public int height;

    public void Write(UdpPacket packet)
    {
        packet.WriteColorRGB(charColor);
        packet.WriteInt(height);
    }

    public void Read(UdpPacket packet)
    {
        charColor = packet.ReadColorRGB();
        height = packet.ReadInt();
    }
}
```

FIGURE 20. Avatar customization class

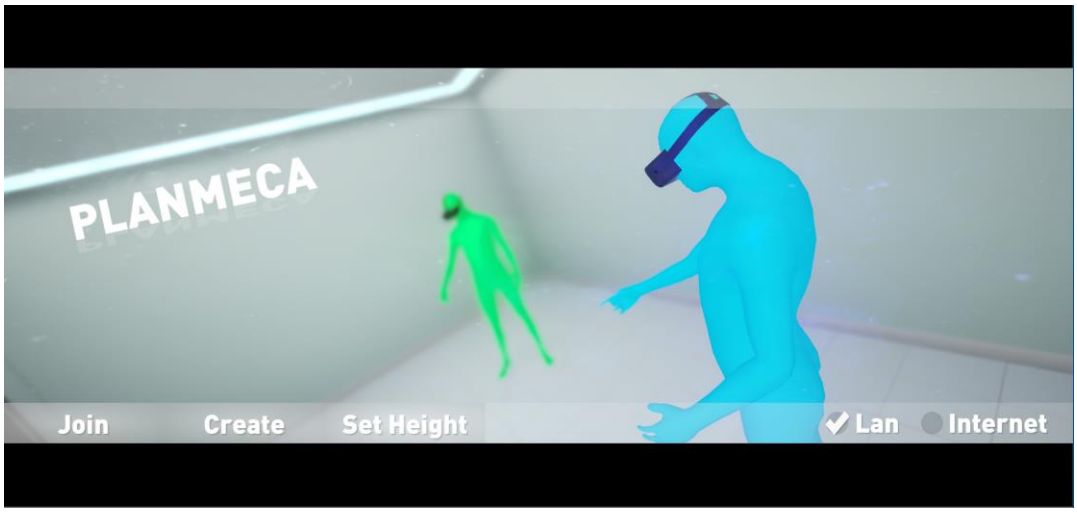
6.6 Teleportation

Based on the discussion with the customer, their preferred locomotion method in VR was teleportation. This was a mechanic they were already accustomed to and satisfied with. Normally teleportation works by changing the values of the transform component of the VR camera rig. This way the entire VR rig, and not just the avatar, locates to a new position in the scene.

As discussed in chapter 4.11, Unity provides a InputTracking.GetLocalPosition(XRNode.Head) method for getting the position of the HMD. Because the method only gets the local position, i.e. the position of the HMD in relation to the root of the camera rig, the position of the avatar will not be replicated over the network correctly if the position of the camera rig is altered. To correct this, an offset vector that represents the vector between the origin of the scene and the VR rig can be added to the replicated position of the user. Alternatively, the code can be changed to replicate the world position of the HMD and motion controllers.

7 RESULTS

This thesis project resulted in the creation of a collaborative VR application for Planmeca Oy (picture 8). The application was designed to support Planmeca's research & development operations. The application starts from a main menu, where users can set their height, choose between LAN and internet connection and connect to the network as the server or the client. Users are connected to a room where they can operate a digital twin of Planmeca's technology. This allows the company to make design choices and iterate faster and more efficiently. Researchers can connect to the same virtual scene and experience their design in an immersive and collaborative way.



PICTURE 8. Planmeca collaborative VR application main menu

The results also include all the collected information about developing a collaborative VR application. The chosen software framework for this thesis project was composed of Unity, SteamVR SDK and Photon Bolt networking library. The methods described in this thesis project have been used to produce a prototype collaborative VR platform for Intopalo Oy (picture 9). In the application users can connect to the same virtual space and interact with a car model in numerous ways.

PICTURE 9. Collaborative VR prototype environment for Intopalo Oy

8 DISCUSSION

Based on the results, this thesis project succeeded in reaching its objectives. A collaborative VR application was produced and insight into the design and development process was gained. Customer feedback and internal testing sessions were used as the main method of validation throughout the development process. An agile approach was used and the customer could participate and collaborate in the development of the software. In the end the customer was satisfied with the outcome.

In the final sprint review the customer stated that the collaborative VR environment has generated a myriad of new opportunities for the company. Planmeca Oy is in the business of dental and imaging device technologies, and in this application the users can act as real-world patients and nursing staff. This helps in making design choices regarding the technologies in development. The customer also expressed that they can use the VR application in the same tracking space without complications and collisions, making it possible for the users to be in the same room. They experienced some difficulties with the LAN connection, and this was considered to result from the dual ethernet card setup on one computer. Most of the design choices were considered as well-executed.

Future work and opportunities were discussed with the customer and one particularly interesting use case emerged: collaborative VR could be used to present the company's products in a virtual showroom. With the advent of affordable standalone VR headsets such as the Oculus Go and Vive Focus, the customers of Planmeca could easily and cost-effectively experience a guided virtual tour of their products in an immersive social setting. Combining the possibility of importing CAD models into a collaborative environment at runtime was also discussed and explored. This is a feature that Intopalo Oy will evaluate in detail to further extend the capabilities of a collaborative VR environment for industrial design.

Considering the technical implementation, Photon Bolt's authoritative networking model was at times too restrictive for a collaborative application. Bolt is clearly designed with first person shooter (FPS) gaming in mind. In FPS games, server authority is important to prevent cheating and to enable equal and accurate competitive gaming. That is why Bolt uses the Controller paradigm to move and affect objects. This model is designed for

traditional input methods and for controlling a player character. The caveat is that for a collaborative application not designed for gaming, being able to change the ownership or authority of a specific object at runtime would make the architecture a lot simpler.

In addition, Bolt does not allow to change the owner of an entity once it has been instantiated in the network. This leads to scenarios where replicating the state of different objects that can be controlled by anyone becomes complicated with the authoritative model, as all shared objects need to use the QueueInput() and ExecuteCommand() methods instead of setting the new state for an object as the authority and replicating it to other clients automatically. Because of this, the networking code was more complicated than it would have been with other networking solutions, like for instance with Unreal Engine Networking or Forge Networking, and therefore lead to some non-elegant solutions to project-specific problems.

Even though the documentation of Photon Bolt states that network entities can be created in the scene editor and do not necessarily have to be instantiated at runtime, in practice this resulted in network errors. Instantiating all network objects at runtime can make level design more difficult. Bolt also did not always automatically replicate all state changes, especially when the changed floating-point values during a frame were small. On the other hand, creating and joining a server was trivial to implement with Bolt. Photon Cloud combined with Bolt's NAT punch-through technology made connecting via internet simple. Bolt's interpolation and lag compensation techniques made the experience smooth and responsive. Different tools have different advantages and disadvantages, and it is the position of this author that the present thesis produced detailed insight about what aspects should be taken into consideration when selecting between the aforementioned tools.

Unity proved to be an excellent platform for developing multi-user VR applications. In traditional multiplayer game programming, the developer can usually launch several session windows on the same PC and test that everything replicates correctly on all clients. VR headsets and motion controllers work differently, and so testing a multi-user VR application cannot be performed without at least two connected PCs and headsets. Unity provides an integrated version control system called Collab, which makes syncing changes between computers fast and agile. Unity also allows the networked application to function when played in the editor. This is not always possible – for example Unreal Engine 4 requires the application to be built and packaged to be able to connect to other

clients. This is because non-VR applications are usually tested on the same PC by launching several session windows from the editor.

In this thesis, only qualitative research methods were used, and thus evaluating the trustworthiness of the results is not required. Outcomes were assessed empirically based on testing executed by the developers and customer feedback. The sources used in this thesis, such as books and technical documentations, were of reputable kind.

In my view, the significance of this thesis stems from its unique standing point: the research revolves around multiplayer gaming concepts, yet instead of gaming, the study focuses on using these concepts to build an immersive, professional VR application. Multiplayer gaming has been extensively studied, but professional, immersive multi-user applications with similar mechanics have not been investigated in such detail. As the global enterprise VR market continues to grow rapidly, similar studies will surely emerge. Collaborative VR solves many real-world problems and thus it is hard to not believe in its importance in the future. Instead of Skype calls, remote meetings can be organized in VR, where people can interact with each other on an entirely new level compared to two-dimensional screens. Designers can work together in visualizing full-scale 3D models and make changes on the fly. Instead of drawing on a whiteboard, people can draw in the air using all three dimensions, and then export their creations for more accurate modifications.

This project was mostly restricted by its limited scope and budget. Building a foundation for a large-scale collaborative VR platform is a complex endeavor that requires a dedicated team with specialized skills. This is why the focus of the thesis was on commercial tools that were readily available and provided the fundamental features out-of-the-box. For large-scale enterprise platforms developing customized networking technologies will most likely provide better performance and functionality in the long run.

I have reasons to believe that the techniques used in this thesis can be taken and applied to a larger-scale project. For Intopalo Oy, the commissioner of the project, this study provides a stepping stone into developing collaborative VR applications that can be scaled. Possible future features could include support for hundreds of concurrent users, secure connections, mission-critical foundations in the networking technology, runtime importing and manipulation of CAD models and photorealistic immersion.

9 CONCLUSION

The purpose of this thesis was to gather information on the design and development of a collaborative VR environment and to produce a collaborative VR software application for Planmeca Oy. The aim was to consider carefully different technological solutions, such as game engines and networking libraries and study the implementation of said solutions.

A collaborative VR software application for Planmeca Oy was produced. Planmeca Oy, a high-tech dental company, is a customer of Intopalo Oy, the commissioner of this thesis project. The requirement specifications for the product were met and the customer was satisfied with the results. The results also included a prototype collaborative VR platform for Intopalo Oy.

Based on the findings of this study, Intopalo Oy can develop the collaborative VR platform further using the information and methods presented. As mentioned in the discussion chapter, for larger industrial projects a different solution for implementing online networking is recommended. The thesis project also provides a foundation for a possible productization of a collaborative VR platform in the future.

REFERENCES

Amazon. 2018a. Creating Virtual Reality Games in Lumberyard. Accessed 29.3.2018. https://docs.aws.amazon.com/lumberyard/latest/userguide/virtual-reality.html

Amazon. 2018b. Using Amazon GameLift. Accessed 29.3.2018. https://docs.aws.amazon.com/lumberyard/latest/userguide/network-gamelift-using.html

Amazon. 2018c. Creating Virtual Reality Games in Lumberyard. Accessed 19.5.2018. https://docs.aws.amazon.com/lumberyard/latest/userguide/virtual-reality.html

Eadicicco, L. 2017. Inside Facebook's Plan to Take Virtual Reality Mainstream. Accessed 8.4.2018. http://time.com/4881487/facebook-vr-spaces-preview/

Epic Games. 2018a. Unreal Engine for AR, VR & MR. Accessed 29.3.2018. https://www.unrealengine.com/vr

Epic Games. 2018b. Multiplayer Shootout. Accessed 19.5.2018. https://docs.unrealengine.com/en-us/Resources/Showcases/BlueprintMultiplayer

Fink, C. 2018. Charlie Fink's Metaverse – An AR Enabled Guide to AR & VR.

Forge Powered. 2018. User Manual (Forge Networking Remastered). Accessed 28.3.2018. http://docs.forgepowered.com/

Glazer, J., Madhav, S. 2015. Multiplayer Game Programming. USA: Addison-Wesley Professional

Helios Interactive. 2016. Project Artemis: UE4 Multiplayer VR. Accessed 19.5.2018. http://heliosinteractive.com/project-artemis-ue4-multiplayer-vr/

HTC. 2018. Vive Pro. Accessed 18.4.2018. https://www.vive.com/us/product/vive-pro/

Ignatchenko, S. 2017a. Development and Deployment of Multiplayer Online Games volume I. Wien: ITHare.com Website GmbH

Ignatchenko, S. 2017b. Development and Deployment of Multiplayer Online Games volume II. Wien: ITHare.com Website GmbH

Jerald, J. 2015. The VR Book: Human-Centered Design for Virtual Reality. USA: Morgan & Claypool Publishers.

Johnson, E. 2015. Oculus Rift Inventor Palmer Luckey: Virtual Reality Will Make Distance Irrelevant (Q&A). Accessed 5.4.2018. https://www.recode.net/2015/6/19/11563728/oculus-rift-inventor-palmer-luckey-virtual-reality-will-make-distance

Kim, S. 2018. List of the Most Popular Social VR Platforms. Accessed 12.4.2018 https://www.vrandfun.com/popular-social-vr-platform-list/

LaValle, S. 2016. Virtual Reality. Illinois: Cambridge University Press.

Lawrence, C. 2016. No Longer Science Fiction: How VR is Infiltrating the Business Landscape. Accessed 14.4.2018. https://www.enterprise-cio.com/news/2016/jul/18/no-longer-science-fiction-how-vr-infiltrating-business-landscape/

Looman, T. 2016. VR Template Guide for Unreal Engine 4. Accessed 20.5.2018 http://www.tomlooman.com/vrtemplate/

McCaffrey, M. 2017. Unreal Engine VR Cookbook. USA: Addison-Wesley Professional.

Mordentral. 2018. VRExpansionPlugin. Accessed 29.3.2018. https://bitbucket.org/mordentral/vrexpansionplugin

Mullins, M. 2001. Exploring the Anatomy of a Data Packet. Accessed 27.3.2018. https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/

Neukirchen, C. 2017. Multiplayer Network Compendium. Accessed 14.3.2018. http://cedric-neukirchen.net/2017/02/14/multiplayer-network-compendium/

Oculus 2018a. Introduction to Best Practices. Accessed 9.4.2018. https://developer.oculus.com/design/latest/concepts/book-bp/

Oculus 2018b. Optimizing Your Application. Accessed 9.4.2018. https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-performance/

Park, M. 2017. 3 Reasons Why VR's Killer App Will Be Collaborative. Accessed 5.4.2018. https://venturebeat.com/2017/10/11/3-reasons-why-vrs-killer-app-will-be-collaborative/

Photon Engine. 2018a. Photon. Accessed 28.3.2018. https://www.photonengine.com/

Photon Engine. 2018b. PUN. Accessed 28.3.2018. https://www.photonengine.com/PUN

Photon Engine. 2018c. Bolt. Accessed 8.3.2018. https://www.photonengine.com/en/BOLT

Photon Engine. 2018d. PUN vs. Bolt. Accessed 28.3.2018 https://doc.photonengine.com/en-us/bolt/current/reference/pun-vs-bolt

Photon Engine. 2018e. BoltEntity. Accessed 2.4.2018. https://doc.photonengine.com/en-us/bolt/current/reference/boltentity

Photon Engine. 2018f. Bolt 101 – Getting Started. Accessed 4.4.2018. https://doc.photonengine.com/en-us/bolt/current/getting-started/bolt-101-getting-started

Proteus. 2018. Proteus Template. Accessed 20.5.2018. https://github.com/ProteusVRpublic/ProteusTemplate

Road to VR. 2018. Vive China President Shares 16 Lessons for a VR-First Future From 'Ready Player One'. Accessed 5.4.2018. https://www.roadtovr.com/16-lessons-for-vr-first-future-ready-player-one-vive-china-president-alvin-wang-graylin/

Shahar, E. 2017. VR UI Design Pattern—The Tool Palette. Accessed 11.4.2018. http://vrux.design/vr-ui-tool-palette/

Statista. 2018a. Forecast Augmented (AR) and Virtual Reality (VR) Market Size Worldwide From 2016 to 2021. Accessed 4.4.2018. https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/

Statista. 2018b. Global VR Headset Sales by Brand. Accessed 12.4.2018. https://www.statista.com/statistics/752110/global-vr-headset-sales-by-brand/

Takahashi, D. 2017. Game Engine CEOs Talk Past Each Other When it Comes to Statistics. Accessed 2.4.2018. https://venturebeat.com/2017/03/01/game-engine-ceos-talk-past-each-other-when-it-comes-to-statistics/

Unity. 2014. All About the Unity Networking Transport Layer. Accessed 27.3.2018 https://blogs.unity3d.com/2014/06/11/all-about-the-unity-networking-transport-layer/

Unity. 2017. Photon vs UNet: Multiplayer Architecture Explained. Watched 4.5.2018. https://www.youtube.com/watch?v=Y1my5bKhKJY

Unity. 2018a. Multiplayer and Networking. Accessed 8.3.2018. https://docs.unity3d.com/Manual/UNet.html

Unity. 2018b. Unity for VR and AR. Accessed 28.3.2018. https://unity3d.com/unity/features/multiplatform/vr-ar

TechCrunch. 2017. Virtual Reality Headset Unit Sales Are Slowly Improving. Accessed 6.4.2018. https://techcrunch.com/2017/11/28/virtual-reality-headset-unit-sales-are-slowly-improving/

Van de Kerckhove, E. 2016. HTC Vive Tutorial for Unity. Accessed 20.5.2018. https://www.raywenderlich.com/149239/htc-vive-tutorial-unity

VentureBeat. 2017. Europe's VR Sector Has Grown to Nearly 487 Companies. Accessed 14.4.2018. https://uploadvr.com/europes-vr-sector-grown-nearly-487-companies/

VRS. 2017. History of Virtual Reality. Accessed 6.4.2018. https://www.vrs.org.uk/virtual-reality/history.html

VRTK. 2018. Virtual Reality Toolkit. Accessed 20.5.2018. https://vrtoolkit.readme.io/

Weinstein, D. 2017. NVIDIA Reveals Holodeck, Its Groundbreaking Project for Photorealistic, Collaborative VR. Accessed 8.3.2018. https://blogs.nvidia.com/blog/2017/05/10/holodeck/