Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

AEROSPACE
INFORMATION
TECHNOLOGY

Bachelor's Thesis

Developing a virtual Control Room for future satellite missions

Submitted by

Marlene Corinna Busch

Examiner:
Prof. Dr. Sergio Montenegro, Julius–Maximilians–University Würzburg, Germany

Supervisor:
Prof. Dr. Sergio Montenegro, Julius–Maximilians–University Würzburg, Germany

May 12, 2020

Abstract

This thesis deals with the first part of a larger project that follows the ultimate goal of implementing a software tool that creates a Mission Control Room in Virtual Reality. The software is to be used for the operation of spacecrafts and is specially developed for the unique real-time requirements of unmanned satellite missions. Beginning from launch, throughout the whole mission up to the recovery or disposal of the satellite, all systems need to be monitored and controlled in continuous intervals, to ensure the mission's success. Mission Operation is an essential part of every space mission and has been undertaken for decades. Recent technological advancements in the realm of immersive technologies pave the way for innovative methods to operate spacecrafts. Virtual Reality has the capability to resolve the physical constraints set by traditional Mission Control Rooms and thereby delivers novel opportunities. The paper highlights underlying theoretical aspects of Virtual Reality, Mission Control and IP Communication. However, the focus lies upon the practical part of this thesis which revolves around the first steps of the implementation of the virtual Mission Control Room in the Unity Game Engine. Overall, this paper serves as a demonstration of Virtual Reality technology and shows its possibilities with respect to the operation of spacecrafts.

Declaration of Authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Marlene Corinna Busch

Würzburg, May 12, 2020

.....

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective

The aim of this thesis is the implementation of a software tool for the creation of a virtual Control Room that enables the operation of satellites in space. The Oculus Rift system, with a virtual reality headset and two hand controllers, provides the underlying hardware support. This paper commences by delineating Virtual Reality concepts and technologies. Then the fundamental principles of the functionality and structure of a Mission Control Room are discussed. These insights are later used to extract design principles for the implementation of the software. Constituting the basis of the communication system within the virtual Control Room, UDP (User Datagram Protocol) services are outlined in the following chapter. The focus of the thesis lies upon the implementation of the Control Room in the Unity Game Engine. Overall, the generated software is required to be real-time capable in order to support continuous and reliable satellite operations. For this purpose and by implication of this thesis the architecture of the Control Room was implemented, text-only telemetry interfaces were created and the support for Virtual Reality was configured, alongside of various other considerations. How the implementation of the Control Room was established in practice is thoroughly discussed throughout chapter 5. To conclude, the last chapter discusses future possibilities and regions of development for the virtual Control Room.

1.2 Motivation

The desire to build a virtual satellite Control Room, as opposed to a traditional Control Room, originates in the prospect of resource optimization and the reduction of expenses. Mission Control Rooms are highly complex to build. While regulations and security measures for a virtual Control Room are of equal elaboration as they are for traditional Mission Control Rooms, the Virtual Reality software presents certain advantages. With the help of modular programming individual elements in the Virtual Reality application can be replicated effortlessly. This promotes smooth growth and the expansion of the Control Room if necessary. The constraints of physical rooms are obvious, as they cannot simply be expanded. Overall the initial building costs, rental costs, in-office utilities and supply expenses as well as transportation and travel costs are greatly reduced through the use of virtual workplaces. Another main advantage of the virtual Mission Control Room is the

irrelevancy of geographical location. This is especially relevant for space missions because they are often joint ventures. Scientific and technological support during the space mission is often offered by international suppliers. Experts from all over the world have the possibility to work together in a virtual room, even if they are physically separated by large distances.

Chapter 2

Virtual Reality

2.1 Immersive Technologies

Due to the novelty of the Extended Reality (XR) technology and its early stage of development the surrounding semantics are not yet globally defined and hard boundaries are not yet set. Immersive technologies include numerous slightly different technologies with flexible terminology. The generic term Immersive Technology refers to a technology that tries to emulate a physical world using digital stimuli. The creation of a sensory environment then induces a feeling of immersion in the user. The extent to which the physical world is changed or simulated differs greatly from technology to technology. Thus the hard- and software requirements vary as well, creating different kinds of immersive experiences. [1] For the purpose of this thesis the following distinctions will be made.

Virtual Reality

The goal of Virtual Reality (VR) is full immersion. The decisive feature of Virtual Reality is that the real world is wholly concealed from the user as immersion into the virtual world takes place. In Virtual Reality it is possible for the user to alter the viewing angle and perception of objects by moving, standing up or bending down. The emulated world adapts accordingly and creates a completely separate surrounding sensory feeling. The simulation is carried out either on screens, in special projection rooms or through headsets. With the help of input devices like controllers, the user can interact in real-time with this environment. Thus the user develops the impression of actually being in the virtual environment. [1]

Augmented Reality

In Augmented Reality the natural surroundings of the user are still visible. The user views the natural world through a camera. The image of the environment is enriched with computer generated content. Virtual objects are integrated into the view. No specific hardware is needed for a simple Augmented Reality experience. For instance a smartphone with a camera suffices. An Augmented Reality software renders additional objects on top of the camera's recordings. Interactions between the digital layer and reality are not possible. [1]

Mixed Reality

As the name suggests, Mixed Reality mixes the real world with a virtual reality. It is a combination of Virtual Reality and Augmented Reality. While Augmented Reality only shows additional content on top of the camera view, Mixed Reality takes additional measures to further integrate virtual objects into the scene. The computer generated content in Mixed Reality is responsive and adapts to the natural surroundings. For instance, computer generated virtual objects can be obscured by real-world objects if they are placed in front of each other. [1]

2.1.1 Immersion

In Virtual Reality, the concept of immersion describes the extent to which a human perceives an artificial world, which is created through the use of technology, as real. Immersion is the state a user assumes when mind and perception are separated from the physical world. Adequately stimulating more senses results in a deeper feeling of immersion and thus a more successful Virtual Reality experience. Key elements to induce immersion include sensory stimulation, feedback and interactivity. Sensory stimulation is achieved through visual and audio input as well as haptic feedback. Feedback is essential for the feeling of immersion as it empowers the user with the ability to perceive the results of his or her actions. An environment that reacts to the user's input seems more realistic. Interaction is only valuable if the virtual system has the ability to understands the user's intent and delivers adaptable feedback in real-time. [2]

2.2 Virtual Reality

Virtual Reality tries to emulate real environments through digital stimuli by addressing human senses and thereby creating digital worlds. The digital world is conceived through a computer system which consists of several components. The system encompasses a virtual and a physical environment as well as a software and a hardware interface, which allows interaction between the human and the computer. A fundamental part of the virtual world is responsiveness. This implies user control. Inside the virtual environment, the user can easily interact with the world and manipulate it by navigating through it or moving and picking up objects. Inside the simulated environment the user is an active and integrated character, rather than a passive observer. This generates the illusion of 'being there' within the participant. Virtual environments are realized through a combination of high performance real-time 3D computer graphics as well as interactive devices such as VR headsets and input controllers. These features shape Virtual Reality technologies as powerful Human Computer Interfaces. [3]

2.2.1 Virtual Reality technologies

The most common technology used to distribute Virtual Reality content is the Head Mounted Display (HMD). It is a visual output device worn on the head that presents images on small screens directly in front of the eyes, while blocking out the the users physical surroundings. HMDs can be tethered like the Oculus Rift, which means they rely on external computers to generate the 3D content and are connected to them via cables. Alternatively they can be mobile, like the Oculus Quest, with on-board operating systems for more freedom in movement. HMDs are equipped with Inertial Measurement Units (IMUs) and other tracking technologies to acquire location and

positional data from the user. This enables a matching representation of the user's actions in the Virtual Reality. Since the experience becomes more immersive, the more senses are addressed, the Head Mounted Display may be complemented with haptic hand-held devices like controllers, mechanical devices such as omnidirectional treadmills or headphones to provide sound [4]. A different method to create a virtual environment is through the use of spatially immersive installations such as CAVE Automatic Virtual Environments. Here, the virtual content is projected onto room-sized screens to fully encompass the user. CAVE like systems overcome some limitations of HMDs. For instance, as they are much less sensitive to tracking errors, simulator sickness becomes a smaller issue. Furthermore the feeling of disembodiment is reduced and a less isolating experience is created since the user can see his own body in the scene and therefore feels more present. However, CAVE technologies are not widely used among standard consumers as they are restrained by their large size and high costs. [5]

2.3 Hardware - Oculus Rift - Consumer Version 1

The Oculus Rift Consumer Version 1 is a Virtual Reality headset developed and manufactured by Oculus VR. In March 2019, the CV1 was replaced by its successor, the Oculus Rift S, however the CV1 remains to be supported in software and was used to establish this project. A field of view of 110 degree diagonally and 90 degree horizontally is achieved through two OLED displays running at 90 Hz with a resolution of 1080x1200 per eye. The total resolution of 2160x1200 Pixel, combined with the large field of view makes it possible to display sharp images, in which the edges are barely perceptible. This deepens the feeling of immersion in the user. The displays focus and reshape the image for each eye and create a stereoscopic 3D image by angling the two 2D images to mimic human 3D vision of the environment. [6]

The Head Mounted Display is used in combination with the Oculus Rift Constellation sensors. They are the headsets rotational and positional tracking system. Constellation uses external infrared tracking sensors to optically track the VR headset and other compatible VR devices. Each device has a predefined subsurface constellation of infrared LEDs that are identified by the Oculus Constellation sensors. They detect the light of the LED markers frame by frame. The Oculus Software also receives information about the acceleration of the device from its accelerometer and about its rotation from a gyroscope. The software then determines changes in position and rotation of the devices, to establish 6 Degrees of Freedom tracking and thus allowing the user to move freely through the virtual space by moving the head [7]. With a sampling ratio of 1000 Hz, a near zero latency between the actual movement and the image rendering is achieved. The power supply and information transfer for the Head Mounted Display is realized via only one main cable that splits into a USB 3.0 and a HDMI cable. The headset includes integrated 360-degree spatial audio headphones. Furthermore, the Oculus Rift Consumer Version 1 is compatible with the Oculus Touch Hand Controllers. The Touch Controllers are tracked by the Constellation sensors in the same way as the Head Mounted Display. They are supplied with power through standard AA batteries. In addition to the hand position tracking, different hand gestures and finger poses are recognized and represented in the virtual environment. Each controller consists of three buttons, one analog stick, two triggers and sensors to detect the hand poses. They also feature rumble motors to provide haptic feedback. [6]

(a) Oculus Rift CV1 - Head Mounted Display [8]

(b) Oculus Touch Hand Controllers [9]

(c) Constellation Sensor for tracking [10]

Figure 2.1: The Oculus VR Setup used in this project

2.3.1 Hardware specifications for the Oculus Rift

	Recommended specifications	Minimum specifications
Graphics Card	NVIDIA GTX 1060 / AMD Radeon RX 480 or greater	NVIDIA GTX 1050Ti / AMD Radeon RX 470
CPU	Intel i5-4590 / AMD Ryzen 5 1500X or greater	Intel i3-6100 / AMD Ryzen 3 1200, FX4350
Memory	8GB+ RAM	8GB+ RAM
Video Output	Compatible HDMI 1.3 video output	Compatible HDMI 1.3 video output
USB Ports	3x USB 3.0 ports plus 1x USB 2.0 port	1x USB 3.0 port, plus 2x USB 2.0 ports
OS	Windows 10	Windows 10
source: [11]		

Table 2.1: Hardware specifications for the Oculus Rift - Consumer Version 1

2.4 Virtual Reality Applications

Ivan Sutherland, a pioneer in Virtual Reality technology, developed one of the first computer graphics technologies with the ability to create virtual environments in the mid 1960s. Since then Virtual Reality has widely broadened its range of application. As for aerospace technology, an early form of Virtual Reality was used by NASA for telerobotics and scientific visualization since 1986 [12]. Another benefiting use for Virtual Reality in space technologies lies in astronaut training. NASA astronauts train on virtual models to internalize complex procedures. For example models of the Hubble Space Telescope and virtual models of the Space Shuttle [13]. Likewise, ESA developed a

virtual representation of the ISS COLUMBUS module that is used for astronaut training and as a tool for familiarization [14]. JAXA uses Virtual Reality to study the effects of microgravity on human orientation [15]. Today one of Virtual Reality's main purposes is education. VR offers immersive On-the-Job learning experiences. For this purpose, NASA created an immersive technology called OnSight, that allows users to virtually explore the surface of Mars. This concept can be transferred to planetary exploration in general [16]. Another example for Immersive Education is Apollo 11 VR, an application enabling any user to experience the Apollo 11 mission in great detail while interacting with the simulation [17].

Next to the mentioned aerospace and educational applications Virtual Reality became an integral part in aspects of the daily life such as medicine, architecture, engineering, entertainment or in the military. Through Virtual Reality, physical distances between users become less important as advances in telecommunication are made. Physical presence is mostly irrelevant for VR applications. Thus users from all over the world can connect in a virtual space and interact with each other over large distances. The Start-up company "VirBELA" uses this idea to create next generation virtual workplaces. Their virtual workstations offer new methods of handling information and provide intuitive means of interaction. [18] The concept of remote workplaces is transferable to satellite missions and Mission Control Rooms.

Chapter 3

Mission Control Center

3.1 General

A satellite mission exceeds the functions of the space segment, which generally consists of the satellite bus system and the payload. The space segment is complemented by a launch system and the ground segment. The ground segment includes a ground station with antenna sites, the Mission Control Center with a Mission Control Room, a Data Center responsible for data warehousing, processing and data distribution, and finally the contractor of the mission. The Mission Control Center (MCC) acts as the central point of control for all Ground Support elements and thereby accomplishes a mission's operational tasks. A dependable, adaptable and cost-efficient Mission Operations department is one of the main warrantors for a safe and successful space flight mission. The Mission Control Center performs specific procedures during the entire satellite lifetime, to realize the mission's goals. Associated operating concepts have to be developed simultaneously to the creation of the space segment to ensure consistency and compatibility to all other elements. [19] It is necessary to differentiate between Control Centers for manned and unmanned missions. However this paper focuses on unmanned satellite missions only.

3.2 The Mission Control Room

3.2.1 Requirements

Mission Control Rooms (MCR) are bound to support the real-time operation of spacecrafts. For instance, tracking and telemetry data require immediate availability in the Control Room. The speed of response is not solely defined by the design of the communication architecture, between the space and ground segments, which handles great amounts of communications entering and leaving the Control Center. The speed of response is also determined by the composition of the Mission Control Room staff. Faster responses are achieved by shorter chains of command and autonomous teams rather than large hierarchies. This also reduces the overall number of operators in the room and thus leads to resource optimization, which is another main consideration concerning the design. The goal is to offer an efficient work environment by providing unique data displays, interfaces and room arrangements that are adapted to each specific mission, while still maintaining operational flexibility. Extending beyond the initial training, typical space missions involve further continuous

education during the mission. Hence the need for duplicated systems emerges. A separate Mission Control Room and an identical simulation tool allow for operational work and for concurrent further training. For long term operations, each mission disposes over an individual Control Room. However MCRs can also be shared and reused across multiple missions [19]. To achieve effortless transitions, a generic operational process, which remains unchanging throughout different missions, is detached from mission specific elements and technologies. As a conclusion Mission Control Rooms and all other elements need to balance security for the success of the mission with efficiency. Future Mission Control Rooms benefit from advanced technologies and should take advantage of new operating tools to facilitate global cooperation in a dependable and reliable manner.

3.2.2 Purposes and tasks

The basic tasks for the operation of spacecraft systems can be divided into four categories. They are mission planning, mission operations, education and training and lastly scientific and technical support. The Mission Control Room claims responsibility for all operational activities. To begin with this means constant monitoring of all spacecraft subsystems including trend analysis. The satellite and payload data is send to the ground segment. Afterwards the Control Center is in charge of processing, distributing and analyzing this data. The responsibilities further include the planning of operating procedures and the commanding of the spacecraft. Besides the tracking of the spacecraft, the desired orbits are calculated and orbit maneuvers are executed to correct the flight path and attitude if necessary. Hereby the Mission Control Room staff stands in close communication with the contracting authority. Apart from operating procedures on board of the spacecraft, all terrestrial operating procedures are coordinated from the Mission Control Room. In addition, the Mission Control Room manages organized communication between all elements. [20]

3.3 Mission Control Center staff

The employees working in a Mission Control Room are called flight controllers. Their composition in the Control Room varies depending on the mission's specific architecture and needs. However certain positions and a general structure are commonly found in almost all Mission Control Rooms.

3.3.1 Flight Director

The Flight Director (COL Flight) leads the flight controllers and is the supreme authority in the room. The overall responsibility for a safe and successful mission and the final decision-making is inherited by this position. [21]

3.3.2 Flight Control Team

The Flight Control Team (FCT) is subordinate to the Flight Director. The team is composed of highly skilled professionals who direct the mission's progress and monitor individual parameters. In case of a satellite mission this comprises all satellite subsystems, i.e. Power Subsystem, Thermal Control Subsystem, Communication Subsystem, Attitude Control Subsystem, the On-Board Computer and the individual Payload systems. Based on their observations and analysis, members of the Flight Control Team deliver recommendations for further operations to the Flight Director while carrying out the Flight Director's decisions. [21]

3.3.3 Staff Support Team

The Flight Control Team stands in close communication with the Staff Support Team, which is composed of 'backroom' Flight Controllers. They do not physically sit in the Mission Control Room. However they participate in the real-time operating of the spacecraft in adjacent rooms. Both, the Flight Control Team and the corresponding Staff Support are specialized for certain functions of the spacecraft. While the FCT regards individual parameters in the overall context, a single Staff Support member deeply concentrates on their assigned parameter and is therefore responsible for detailed data evaluation and recommendation. Depending on the scale and difficulty of the mission, the Staff Support Team is only active in periods of high intensity or not at all. [22]

3.3.4 Extended Support Teams

In addition to the Staff Support Team an Extended Support Teams exist. They are not part of real-time mission operations but rather provide long term scientific and technical support. This team includes hardware and software designers, data and system analysts, as well as engineering specialists.

3.4 Mission Operations

Spaceflight missions encompass four phases. The first phase constitutes the mission planning phase. Here the Mission Control Room is configured and the personnel is trained. The second phase is the Launch and Early Orbit Phase (LEOP) with a duration of approximately two to four weeks beginning with the launch. The Mission Control Center assumes full control of the satellite once it is detached from the rocket. The first contact with the satellite in space is established and the orbit is determined and possibly altered. During the LEOP, extensive supervision by the Flight Controllers is required to increase the probability of the mission's success. This phase is followed by the Commissioning Phase in which the satellite's payload is activated. In the case that external clients exist, the control of the satellite is handed over to them. After this, Mission Operations is transitioned into the Routine Operation Phase. This last phase requires less staff as the performed activities become repetitive and more predictable. For grand scale missions the Control Room is staffed 24 hours every day throughout the whole mission and the teams work in shifts. As missions become smaller, staff in the Mission Control Room is successively reduced during the first weeks after assuming control over the satellite. Periods of high activity besides the launch and the first acquisition of the spacecraft occur during and shortly before flyovers. A flyover happens when the spacecraft is visible to the ground segment and direct communication via the antenna site is possible. Within the Mission Control Room, acoustic and visual signals indicate such periods. For instance, in the German Space Operations Center (GSOC) the light is dimmed to announce critical phases and to induce high concentration. [23]

3.5 Mission Control Room elements and design

3.5.1 Examples

Worldwide several space agencies maintain Mission Control Centers to exchange telemetry data with respective spacecrafts from the launch throughout the entire course of the mission, up to the recovery or disposal of the spacecraft. The most prominent of these Control Centers are the following. The European Space Operations Centre (ESOC) is located in Darmstadt, Germany and serves as the main Mission Control Center for the European Space Agency (ESA). It supports missions like Mars Express and it operates the European Space Tracking network (ESTRACK). [19] NASA operates their manned spaceflight missions from the Johnson Space Center (JSC) in Houston, Texas. The JCS also serves for astronaut training and the supervision of missions to the ISS. [22] The German Space Operations Center (GSOC), in Oberpfaffenhofen, Germany is among other missions, responsible for the maintenance and controlling of the Columbus module of the ISS. [21]

(a) ESOC [24] (b) JSC [25] (c) GSOC [26]

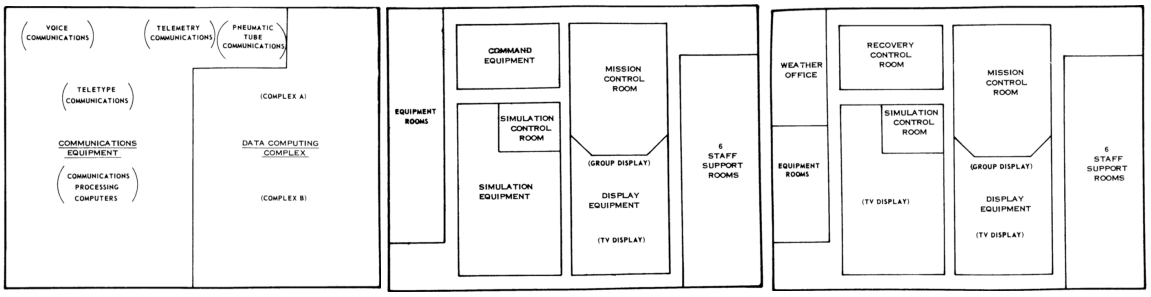
Figure 3.1: Mission Control Rooms from various Mission Control Centers worldwide

Reflecting on the demands for functioning Mission Control Rooms it is not surprising that the general appearance of the different rooms is notably similar. The large monitors in the front of the rooms originate from the generic felt need for a connecting element between the different teams. The large monitors are useful for briefings and problem solving in large groups. At the same time, individual workplaces are placed in various smaller groups to physically highlight the separate teams and their particular functions. [22]

3.5.2 Redundancy

Equally to most other mission elements redundancy is a critical part of a Mission Control Centre. Therefore multiple Mission Control Rooms typically exist even though only one is used for real-time Mission Operations. Duplicated Control Rooms exist as backups or for the purpose of simulations and training. Backup Control Rooms may be found in the same building, neighboring buildings or even on completely different sites to be prepared for all emergency situations. Excessive redundancy is especially important for manned missions during which human life is at risk. Redundancy further includes multiple but separate power supply systems and network accesses. [21]

3.5.3 Mission Control Center example layout



(a) First floor [22] (b) Second floor [22] (c) Third floor [22]

Figure 3.2: Room arrangement in the Johnson Space Center in Houston, Texas

Besides the Mission Control Room and its backup and simulation replications, Mission Control Centers also house the Telecommand and Telemetry systems, the Communication network, data archives, Off-line Analysis tools, the Mission Planning element and further equipment rooms. [22]

3.5.4 Mission Control Room communication system

The communication within Mission Control Rooms is realized via headsets by using voice-loops, a real-time communication system. Multiple loops exist for differing topics. A Flight Controller can broadcast a message to all other coworkers, listening on that loop. Participating in multiple conferences by monitoring several loops at the same time is possible. However only authorized personnel, such as the Flight Director, my talk on more than one loop at once. The communication system needed for real-time mission support is required to be highly reliable, easy to maintain and continuously available. This is the case for voice-loops. The spoken language in the Control Room is very formal and is characterized by the fact that confirmations must be explicitly requested. [27]

Chapter 4

User Datagram Protocol

4.1 General

The User Data Protocol (UDP) is a communication protocol that is used as a pruned alternative to the Transmission Control Protocol (TCP). It is defined for the use with the IP network layer protocol. UDP works on layer 4 which is the transport layer of the OSI-Model. The protocol is regarded as an unreliable service as it does not provide any guarantees for the safe and intended delivery of the data. For instance it does not provide protection against duplication or loss of data which may occur due to software failures in the system. [28]

4.1.1 UDP architecture

The protocol differentiates itself from other communication techniques by not establishing an end-to-end connection between the communicating endpoints. Therefore UDP is considered to be a connectionless protocol. This is highly advantageous for applications that require efficient and fast communication methods. The efficiency results from the possibility to immediately commence the data transfer. This is achieved because no connection needs to be build up and the waiting time for confirmation messages is avoided. Moreover, the UDP Protocol manages with a much smaller header that is only 8 bytes long. The header contains the source port, the destination port, the length of the packet and lastly a checksum. The data section of a UDP Datagram is limited in size because it is carried by a single IP packet and it therefore cannot exceed 65507 bytes for IPv4 or 65527 bytes for IPv6. [28]

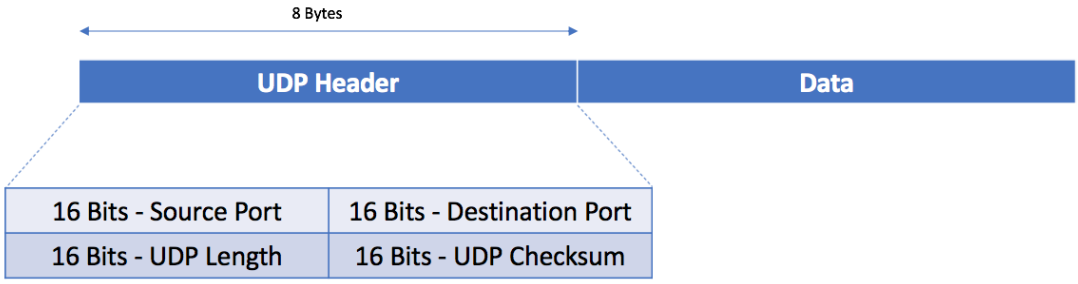


Figure 4.1: A UDP Datagram with the UDP Header and the Data Section.

With a total of 8 bytes, the UDP Header is very slim and can be processed with little computing power. The header consists of four segments, each is two bytes in length. The Source Port indicates the sender's port number and is only required if replies to the sender are expected. Likewise, the Destination Port identifies the recipients Port number, however it must always be specified. Source and Destination Ports both reveal Service Access Points (SAP). The length of the combined UDP Header and payload section is inscribed in the UDP Length field and can be used to determine the completeness of the UDP Datagram. Finally, the UDP Checksum is utilized to determine whether the transmission of the Datagram was completed successfully. [28]

4.1.2 Port structure

The User Data Protocol uses a port structure to ensure that a Datagram is transferred to the correct application on a certain device. The ports are software abstractions that are part of the network addressing system to distinguish between multiple destination endpoints on a single host computer. Specific ports are managed by the Internet Assigned Numbers Authority (IANA). These ports are used for official practices and range from 0 to 1023. Because of their restricted use, they are also called System Ports. The so-called User Ports range from 1024 to 49151. They are also registered with IANA and are reserved by requesting parties for specific services. At last, ports ranging from 49152 to 65535 are Dynamic or Private Ports and cannot be registered with IANA. Their purposes are temporary applications and customized services. [29]

4.1.3 UDP characteristics

The User Data Protocol does not provide extensive control mechanisms and is thus limited in its functionality but beneficial for its simplicity. The protocol does not track the Datagrams and does not show confirmation reports. Furthermore Datagrams used by UDP are not sequenced. This means the order in which the segments arrive at the destination might be disturbed. In comparison to other protocols like TCP this characteristic marks UDP as an unreliable communication method. Applications using UDP services need to provide the security on their own. This can be achieved by additional, customized protocol mechanisms without being limited by inconvenient features of more

secure protocols. Set side by side with TCP, the User Datagram Protocol has a lower latency, as it does not wait for lost Datagrams. Additionally, UDP enables Multicasting and Broadcasting of messages which is highly useful for applications including numerous participants. [30] Due to the trade-off between faster data processing in return for less security, UDP is only suitable for specific applications that are fault tolerant and rely on data transmissions with low latency. This includes real-time applications where continuous data flow is essential. This is not possible with TCP as the data only flows if acknowledgements are send back and forth. UDP uses much less network resources and is not limited by the connection management. Since UDP requires only a minimal amount of computing power it is the ideal communication method for this project.

Chapter 5

Implementation

5.1 Unity Editor

The implementation of the virtual Control Room was realized using the Unity Editor which is a cross-platform runtime and development environment. For this project the Unity Editor version 2019.2.10f1 is used. The project is based on a new and empty 3D project, using the standard render pipeline.

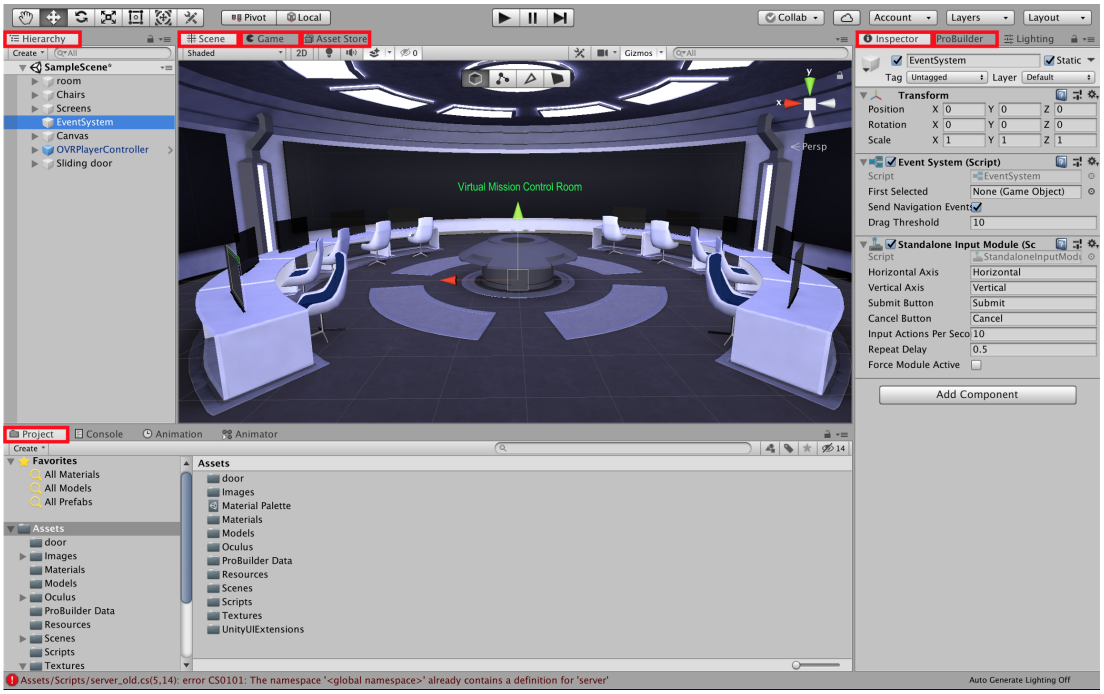


Figure 5.1: Screenshot of the Unity Editor in the Default Layout with additional windows

5.1.1 Windows

Project window

The Project window displays all files and directories related to the current Unity project. The files which are collectively known as Assets include all used or unused models, scripts, textures and other elements. From the Project window it is not only possible to search assets of the current project but also to search the Unity Asset Store.

Asset Store

Unity's Asset Store offers a growing library of free and commercial assets, of which some were created by companies and others were created by community members. Differing types of assets are available. They include models, animations and important tools for VR development like the Oculus Integration or ProBuilder.

ProBuilder

ProBuilder is a Unity asset downloaded from the Asset Store. The tool provides developers with the ability to create, edit and texture custom geometry in the Unity Editor. Moreover, it offers UV editing which was the primary reason to use ProBuilder for this project.

Scene view

Unity projects are developed by creating scenes. Each scene represents a different environment and is constructed by placing objects into it. The scene is organized as a scene graph from the so-called GameObjects. A GameObject's position, rotation, scale and general appearance in the scene can be changed in the Scene view. The Scene view also operates as a preview of the application.

Hierarchy window

All currently used assets appear with their GameObject name in a listing in the Hierarchy window to the left of the Unity Editor. The Hierarchy window provides scene visibility controls, to hide certain GameObjects from the Scene view, without altering their in-game visibility.

Inspector window

When a GameObject is selected from the Hierarchy window or directly from the Scene view the GameObject's Inspector tab opens. The Inspector window displays detailed information about the selected asset. The GameObjects functionality is altered by adding and removing components such as scripts from the object. Selecting an asset from the Project window results in the Inspector showing the objects import settings and run-time options.

Game view

The image seen in the Game window is rendered from the camera in the scene. Once the Play Mode is entered, the Game view automatically appears and all changes made in the Unity Editor are only temporary.

5.1.2 Toolbars

Various menus and forms allow the manipulation of the scene and the camera view. GameObjects in the scene can be moved, rotated and scaled with the mouse through the help of Unity's standard Toolbar. This is supported for the 2D and 3D view. An additional toolbar is provided by ProBuilder. ProBuilder enables more detailed editing to manipulate individual vertices, edges and faces of objects.

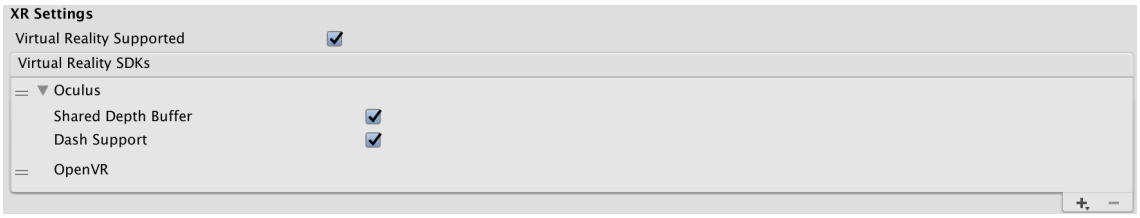
(a) Unity Toolbar. From left to right: Pan around the scene, move, rotate, scale, Rect Transform and Transform objects.
(b) ProBuilder Toolbar. From left to right: select and manipulate GameObjects in Object mode, Vertex mode, Edge mode or Face mode.

Figure 5.2: Screenshot of the Toolbars in the Unity Editor

5.2 Unity Integration and first steps

The Oculus Integration for VR development in Unity is available free of charge in the Unity Asset Store. The downloaded package complements Unity's built in Virtual Reality support for the Oculus Rift, Oculus Go and Oculus Quest by adding scripts, prefabs and samples to the project. These resources offer useful interfaces for the management of VR Camera behavior, first-person controllers and the Oculus Touch Controllers. The built in VR support is established through the OVR Utilities Plugin (OVRPlugin). It is recommended to always use the latest available version of the OVRPlugin regardless of which Unity Editor version is used. [6]

For the initial setup of the Oculus Unity Integration it is essential to enable Virtual Reality support in the Player Settings. Additionally the Oculus SDK needs to be among the Virtual Reality SDKs.



Enable Virtual Reality support by checking the checkbox. Integrate the Oculus SDK into the Project.

Figure 5.3: Screenshot of the Player Settings (Edit > Project Settings > Player > XR Settings)

At this point Unity automatically renders the regular Unity camera in the scene to the connected Virtual Reality device. The camera's transform is overridden by the positional and head tracking data which are supplied by the Head Mounted Display. This configuration only provides basic VR support. To gain more control and to take advantage of more advanced features, an access to the OVRManager which is Unity's main interface to the VR hardware, is needed. The OVRManager script contains functions to shape the VR camera behavior and is added to the OVRCameraRig which is a prefab from the Oculus integration. A prefab in Unity is a template of a GameObject that synchronizes all its instances. [6] It is a useful concept for easy and global changes throughout the whole process of development and comparable to the inheritance concept in object oriented programming.

5.2.1 OVRCameraRig

The conventional Unity Camera in the project is replaced by the OVRCameraRig for a more advanced Virtual Reality experience. The OVRCameraRig also contains a Unity camera but exceeds the functions of the regular camera by providing control mechanisms for stereoscopic rendering and tracking. Stereoscopic rendering creates a set of two images from slightly different perspectives that are combined to establish a 3D image. The camera from the OVRCameraRig tracks the Head Mounted Display. The OVRCameraRig handles three child "anchor" Transforms. The first two Transforms reference the position and rotation of the left and right eye, while the third poses as a virtual center eye in the middle of them. The Tracking Origin Type for this project is set to "Floor Level" tracking which enables 6 Degrees of Freedom tracking and thereby gives the player the ability to move around or work in a seated position. In order to navigate through the virtual world, the OVRCameraRig is attached to a movable GameObject. [6]

5.2.2 OVRPlayerController

The OVRPlayerController is a simple character controller to which the OVRCameraRig prefab is attached. While the OVRCameraRig is responsible for the stereoscopic rendering and tracking, the OVRPlayerController allows the user to move around in the virtual world through keyboard, mouse or controller input. It is represented by a physics capsule and is modeled by the OVRPlayerController script. This script controls the physical properties of the controller and establishes a movement system. [6]

5.2.3 Locomotion

Locomotion refers to the active movement of an individual. In Virtual Reality locomotion enables the player's movement from one location to another location within the virtual environment. A well implemented locomotion system is imperative for a comfortable VR experience. Therefore it is a core design feature of any VR application. It needs to respond to the user's intent while triggers of discomfort need to be eliminated. A mismatch between the senses is such a trigger. If the user feels his movement in the real world but the virtual environment portrays a different motion, the user may feel uncomfortable. For example the acceleration that a user feels must agree with the movement seen in the Virtual Reality.

In this project two types of locomotion system were combined to imitate motion from the real world using a high amount of available input. Physical Movement is the type of locomotion with the highest ratings of comfort as it matches exact physical movement in the Virtual Reality. It

thus eradicates all triggers of discomfort. Nevertheless, this locomotion system is limited in its application because of limited space in the real world. Not all users have access to large areas that are not obstructed by real life objects. Besides not all users have the mobility to navigate through the virtual world using the Physical Movement locomotion system. Therefore Avatar Movement supplements additional features to move around in the virtual environment. This locomotion system uses a combination of varying controller inputs to manage movement in VR. For instance the input, collected from the left and right thumb sticks, controls the user's movement in any direction and the possibility to turn the head. All while remaining in a fixed position in the real world. With the use of two locomotion systems a balance between immersion and comfort is achieved while implementing a functioning movement system. [31]

5.2.4 Hand presence

Hand presence is the primary method of interaction with the virtual world. Virtual hands can be added to the character controller in the scene by including the CustomHandLeft and CustomHandRight prefab from the Oculus Integration. To synchronize the actual hand movement with their virtual representation, the hand prefab's transform needs to be overridden by the Oculus Touch Controller's position and tracking data. This is achieved by setting the left and right CustomHand prefab as a child of the left or right HandAnchor GameObject respectively. In the Inspector window their position must be set to (0,0,0). The CustomHandLeft/Right prefab has two scripts attached to itself. The Hand script controls the animations of the hand movement. Whereas the OVRGrabber script is responsible for the logic behind grabbing other objects in the virtual scene. It is important to ensure that the virtual hands align correctly with the real world hands of the user to provide quality hand presence and to grant an immersive and comfortable VR experience. This process is called hand registration. [6]

5.3 3D Modelling

To create the virtual environment a detailed model of the Control Room is needed. In general Virtual Reality applications require 3D modeling to build a virtual world with which the user may interact. There are three major building blocks of 3D model development. First of all modelling an object means building its basic structure by creating a mesh. An object's mesh defines the 3D objects shape through the combination of vertices, edges and faces. In the next step textures and materials are applied to the object to increase its realistic appearance when rendered. While materials describe overall optical characteristics of an object, a texture is a single component of a material that breaks up the uniform appearance of the material. Lastly the lighting is responsible for a pleasing image and atmosphere by integrating the model into the scene. Additional measurements considered in 3D Modelling are animations. Applications that are adaptable to a wide variety of user inputs and game states elevate the feeling of immersion within the user. Therefore, animations of GameObjects are an additional method to achieve the goal of an interactive application. [32] The meshes for the objects in this project are created in Blender as opposed to the textures and materials which cannot be imported properly from Blender to Unity. All further steps were therefore undertaken in the Unity Game Engine.

5.3.1 Blender

Blender is a modeling and animation software for 3D development. The open source creation suite provides users with features that allow them to work on various aspects of 3D projects. The computer software offers free and full access to all its features. Regarding this project Blender was only used for the basic modelling of objects in order to create a realistic environment and to establish the architecture of the Control Room.

5.3.2 Model Creation

Meshes

Virtual Reality applications are typically more demanding than traditional flat 3D content. Hence certain principals have to be enforced to ensure high quality. Attention to detail in object creation is particularly important because the viewer can see objects in VR from almost any angle as Virtual Reality allows the user to place the camera anywhere where the head can turn in the virtual world. Due to the high performance demands of Virtual Reality applications, it is imperative to use low poly objects. This essentially means keeping the vertex count of the objects meshes as low as possible. The Oculus Rift requires frame updates 90 times per seconds, giving approximately 11 milliseconds to prepare each new frame. As a consequence CPU and GPU workloads should be kept reasonable to maintain constant updates. The amount of geometry in the scene can be reduced without significantly limiting visual pleasantness by enhancing low poly models with more detailed textures. Suitable textures for this purpose foremost include normal maps. [6]

Importing objects from Blender to Unity

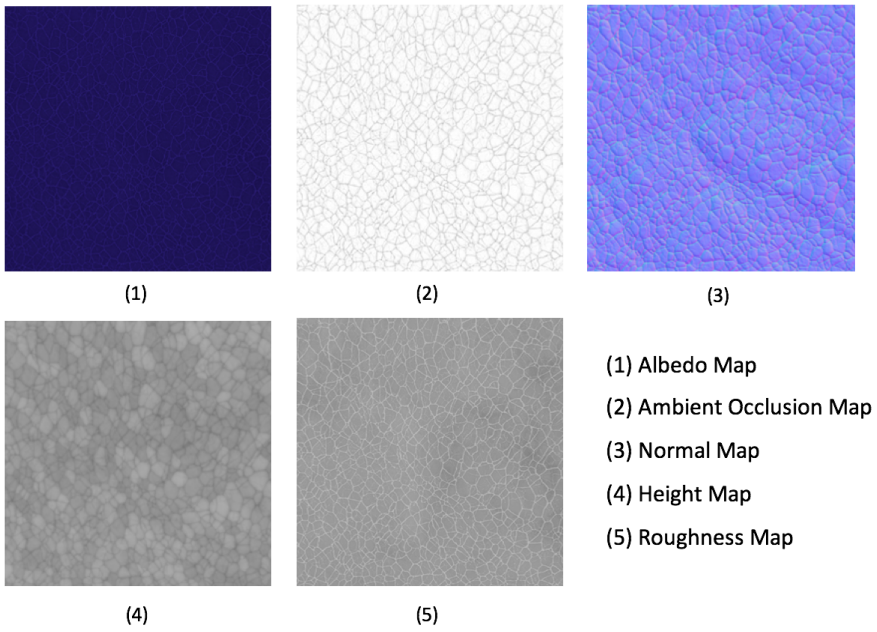
To import models from Blender to Unity, the .blend files have to be imported into the project file structure. The files are natively imported from Blender to Unity by using the Blender FBX exporter. Unity imports the objects meshes with vertices, polygons, triangles, UVs and normals. An object can then be integrated into the scene per Drag and Drop from the Project window into the Hierarchy window or directly into the scene. Clicking on the imported asset in the Project window opens the Inspector tab where the import settings can be managed. Enabling the Generate Colliders checkbox is an easy method to provide the GameObject with realistic behavior. Unity creates a Mesh Collider around the object to make it solid such that the Character Controller and other objects can no longer pass through it. Colliders may also be created manually. Placing an Empty GameObject with a Collider Component around a model creates more primitive but functioning Colliders that increase the performance of the application. If the Blender files of model prefabs are altered during the production, the Unity Editor updates them automatically. [6]

5.3.3 Materials

A general guideline for performant 3D applications is to limit the amount of different materials per asset to reduce CPU overhead. Moreover, GameObjects should be mapped such that the Texel Density, which describes the amount of texture resolution on a mesh, across faces and entire objects remain as consistent as possible. Otherwise visual irritations may occur, resulting in a less immersive user experience. [6]

Textures

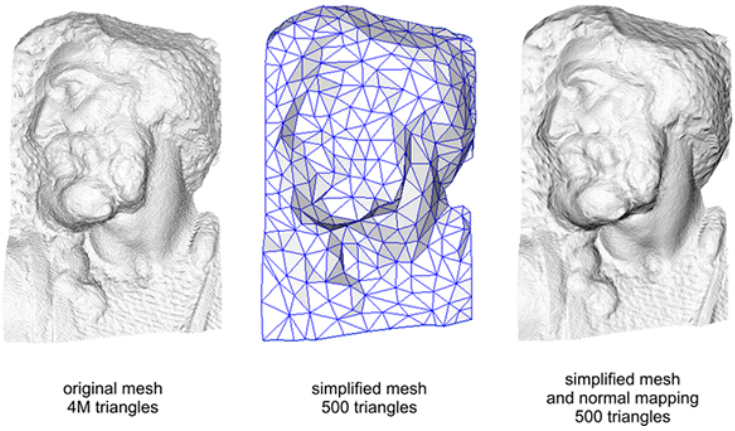
To customize the materials, textures are applied to them. Textures are Bitmap images that deliver fine details for the mesh geometry of an object. Different types of textures exist to regulate specific aspects of a material's surface. The available configuration options for Materials are dependent on their Shader. [6]



These textures were utilized to create a fabric like material to simulate a seat cushion. [33]

Figure 5.4: Overview of different kinds of Texture Maps used in this project.

An Albedo texture is used for the basic coloring of the material. It resembles Diffuse Maps with the difference that all shadows and highlights are removed from the Albedo Map. To provide the missing details Ambient Occlusion (AO) Maps redefine inherently darkened regions to simulate shadows and thereby provide more pronounced detail. Normal Maps affect the normals of the geometry and create the illusion of detail without having to rely on objects with a high poly count. Each pixel of a Normal Map stores RGB values to signify the orientation of the surface normal. Details of the surface of highly detailed models are baked onto Normal Maps that are afterwards applied to low poly models to induce the illusion of high resolution detail. Provided that the camera angle is not to flat Normal Maps simulate the impression of a detailed 3D surface. Similarly to the effect of Normal Maps, Height Maps are used to fake depth or height on a flat surface. However, the height information given by the Map is encoded using black and white values. [34]



The appearance of both meshes is almost equivalent, however the simplified mesh significantly saves on performance. [35]

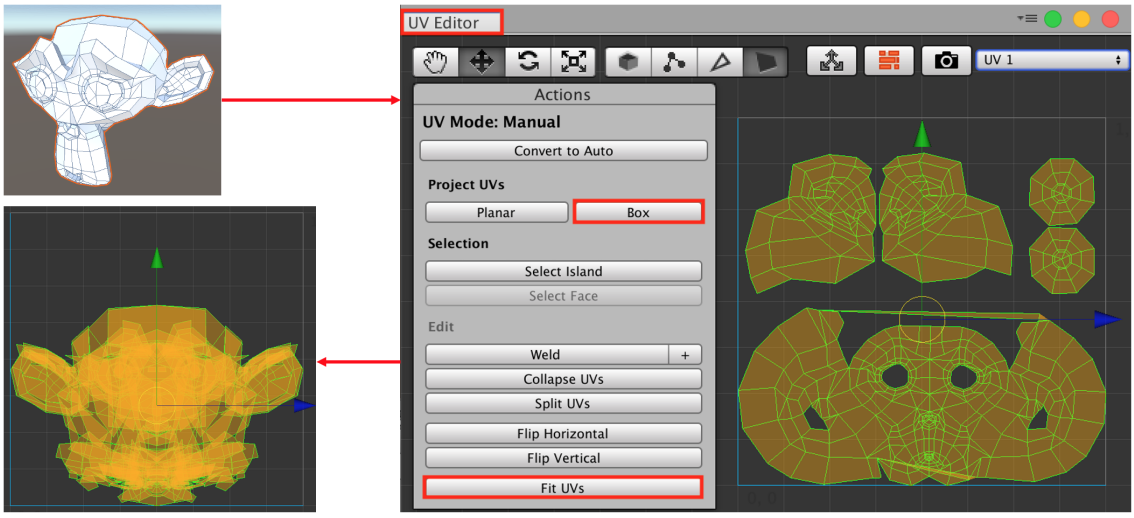
Figure 5.5: Comparison between a high-poly mesh and a low-poly mesh with normal mapping

Shader

Shaders are scripts that take lightning input and material settings, such as textures, into consideration to calculate the color of each pixel when rendered. Depending on the Shader, either Metallic or Specular Workflow can be enabled. For the Standard Shader with the Specular setup, a texture map regulates direct reflections of light sources in the surrounding scene. The intensity and sharpness of the specular reflections are managed by the Smoothness slider. Likewise for materials using the Metallic mode, the map defines the ability of the surface to reflect the imagery around it. Here the specular reflections are not explicitly defined but generated by the Metallic level and Smoothness level. Another option to manage these characteristics of the surface of models are Roughness Maps or their inverted Smoothness Maps as well as Reflection Maps which bear the respective data. [6]

UV-Mapping

UV-Mapping is a method to project two-dimensional image textures on 3 dimensional models. A UV canvas is used to assign each polygon in a model to a part of an image texture. During this process each vertex is assigned a two-dimensional coordinate pair. These coordinates determine which part of the image texture is used for a specific polygon. The two-dimensional coordinates are called UV-Vertices or simply UVs and are comparable to XYZ coordinates in 3D. The process of building a UV map is called UV unwrapping. A mesh must be unwrapped before a texture is applied to it. Otherwise the texture projection has a high chance of being be contorted. This is especially noticeable and inconvenient for curved or complex surfaces. In Unity UV unwrapping is realized by using the ProBuilder package from the Asset Store. Among other tools, it offers Manual UV mode features to edit UVs. The "Box" projection method should be used as it establishes a planar projection for each face. Unlike the "Planar" projection mode that projects the image texture from a single point across all faces. Additionally, the "Fit UVs" feature resizes the selected UVs uniformly for a harmonic appearance across multiple faces or even multiple objects. [6]



The mesh of the complex model on the top left is spread out in the UV plane. The result of applying the Box Projecting mode and fitting the UVs uniformly is shown in the bottom left figure.

Figure 5.6: Process scheme of UV Editing using ProBuilder's UV Editor

5.3.4 Lighting

Realistic lighting leads to a higher level of immersion and a more comfortable VR experience. There are different methods to light a Unity scene. Using the Enlighten method is a profitable solution to deliver real-time Global Illumination and is therefore mostly used for dynamic content. Since the virtual Control Room foremost contains static content, using baked lights is the preferred option. Baking the lights saves on performance and delivers more detailed lighting, as the lighting is calculated beforehand. The process during which Unity calculates all of the lighting values is called "baking". The calculated values are turned into large textures that are later applied to the whole scene. Global Illumination encompasses two types of lighting; direct and indirect lighting. Whereas indirect lighting happens when light bounces off of multiple surfaces before reaching the eye, direct lighting only concerns those light rays that are send out directly from the light source, bounce once and then reach the eye. Global Illumination is simulated by computer algorithms that generally deliver satisfying results by only simulating a few bounces. The lighting in the virtual Control Room is created using Unity's Progressive Lightmapper. It is a path tracing based Lightmapper. Light bouncing around the scene is realistically simulated by gradually calculating the paths of the light rays. [6]

Creating light sources

As the Control Room is an indoor scene, Unity's default Directional Light has to be removed. Instead customized light sources will light the scene. This is achieved by creating emissive materials that are applied to specific objects of the model. The material will emit light onto the rest of the scene. When configuring the material, enabling the emission checkbox in the Inspector tab results in

the creation of an emissive material with the chosen color and intensity. Materials with the emission property enabled will not emit light in real-time. In order for the light to work, Lightmaps are baked before the runtime using the Progressive Lightmapper. The material settings under the emission option need to be changed from Realtime Global Illumination to baked. This method of lighting will only work for static GameObjects. Furthermore, for all imported 3D objects the "Generate Lightmap UVs" setting has to be enabled in the import settings.

Lighting settings

Next, the lighting properties for an indoor scene have to be configured in Unity's Environment Settings. As all light comes from light emitters in the scene, ambient lighting is non-existential and the ambient color is therefore set to black. Moreover the Skybox Material is removed. Realtime Global Illumination is disabled because the Lightmaps are baked. This is a reliable way to get high quality indirect lighting and shadows for low runtime overhead. Using Baked Mode for lighting components allows Unity to pre-calculate the illumination from the lights prior to runtime. The runtime overhead is drastically reduced and the performance improved.

To achieve optimal lightning in the scene, the Lightmapping Settings may be configured according to the figure on the right. The Sample sizes for the Indirect and Direct Samples as well as the number of bounces have no effect on the size of the Lightmap. Instead they only influence the length of the bake time. Unity's post-processing of the Lightmaps sometimes results in faults as Unity takes noisy Lightmaps and smooths them out by applying filters. These filters can be adjusted as shown to reduce lighting glitches. [6]

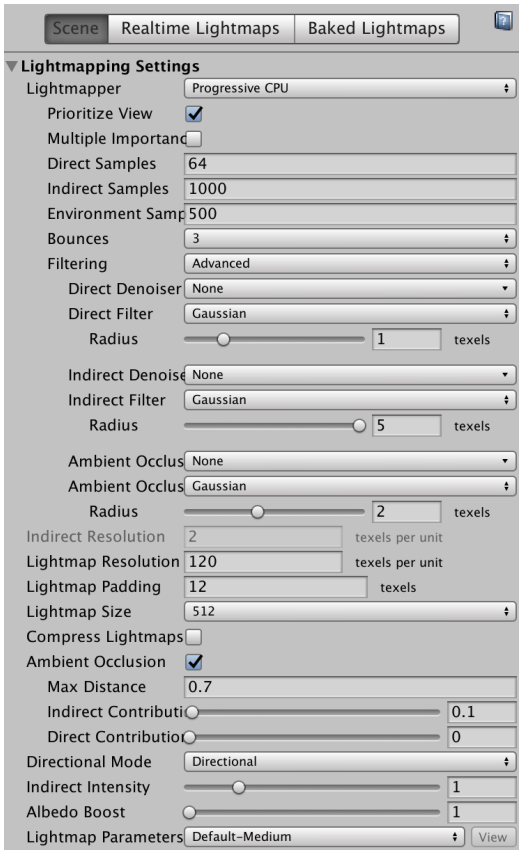


Figure 5.7: Screenshot of the Lightmapping Settings

5.3.5 Animations

Besides the dynamic content on the monitors all other GameObjects except for the doors and the holographic earth are static. The doors serve as a demonstration of the concept of Animations for future development and revision of future Control Room elements. The doors will slide open when the character controller enters a trigger area while standing in front of the doors, afterwards the doors will automatically close again. The holographic earth rotates constantly.

Animation Clips and Animator Controller

Unity enables the developer to create Animation Clips to record defined transformations or other changes in GameObjects. The diverse Animation Clips must be saved in the Asset folder and are managed by the Animator Controller. Animator Controllers act on specified conditions to instruct GameObjects which animation is processed and to realize transitions between states. The Animator Controller uses Animation State Machines and Animation Parameters to establish the work flow and switch from one animation to another. [6]

Configuration of animations

To configure the animation, the GameObject that is supposed to be animated needs an Animator Component. This component requires an assigned Animator Controller asset. The Animator Controller asset must be supplied with at least one Animation Clip. [6]

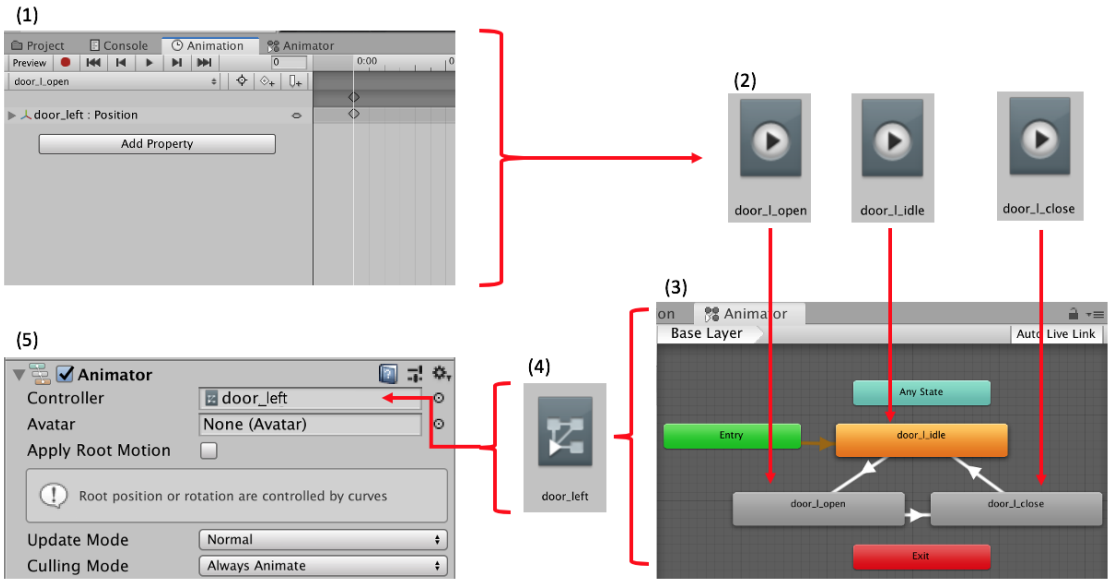


Figure 5.8: Process scheme and required components for the creation of animations in Unity

1. The Animation Window is used to record Animation Clips for GameObjects.

2. The Animation Clips are saved in the Asset folder. The door example requires three animations, a distinct one for the idle state, the opening and the closing of the door.

3. The Animation Clips are referenced in a State Machine to manage the different animations and the transitions between them. This is done in the Animator Window.

4. The information to integrate individual Animation Clips into the project is saved in the Animator Controller.

5. An Animator Component is added to the GameObject that is animated and supplied with the Animator Controller.

5.4 Scripting

Unity GameObjects behave according to the Components that are attached to the object. The configuration of a GameObject is shown in the Inspector tab. All objects posses a Transform Component that is indicating their position, rotation and scale in the scene. Although Unity offers a great variety of scripts and pre-defined Components in its library, custom behavior has to be implemented to provide more flexibility. This is achieved through the use of C#-scripts. They are created in the Unity Editor. The C#-scripts are modified in a text editor, Unity opens Visual Studio per default. All Unity scripts, which are used to control a GameObjects behavior, need to derive from the MonoBehaviour base class. Custom scripts used in this project fulfill the purpose of regulating the telemetry interface, regulating chair and screen visibility and for the animations of the door and the earth hologram.

Any behavior that needs to be repeated constantly during the gameplay is coded in the Update function. This function is called once per frame and is for example used to control the constant rotation of an object.

```
1 public class earth_rotation : MonoBehaviour
2 {
3     public float speed = 0.1f;
4
5     void Update()
6     {
7         transform.Rotate(0, speed, 0);
8     }
9 }
```

The above script is attached to the GameObject that is supposed to rotate. Public variables can be altered and instantiated in the Inspector tab, without the need to open the text editor each time. In the case that a GameObject's behavior is regulated by triggers in the scene, like in the previous example of the sliding door that opens when the user approaches or the chairs that vanish when the user stands in front of the desk, Trigger Events are required to be implemented.

```
1  public class chair_visibility : MonoBehaviour
2  {
3      public GameObject Trigger;
4      public GameObject chair;
5
6      void OnTriggerEnter(Collider other)
7      {
8          chair.gameObject.SetActive(false);
9      }
10     void OnTriggerExit(Collider other)
11     {
12         chair.gameObject.SetActive(true);
13     }
14 }
```

Unity offers build in functions, to detect Trigger Events. The Trigger and the GameObject that is affected by it are declared in the Inspector tab. The script is attached to the Trigger in the scene. A Trigger is created as an Empty GameObject with a Collider Component that has the "Is Trigger" checkbox marked.

For the telemetry interface another script is needed. Once the text is transferred to the Unity Editor via the UDP communication, which is explained in the next section, the text needs to be made dynamic so that it scrolls through the monitors. In the following script a TextMeshProUGUI Component is referenced. It holds the telemetry text. In the while-loop of the start function the y-position of the text is steadily increased to make the text scroll.

```
1  public class scrolltext : MonoBehaviour
2  {
3      public TextMeshProUGUI telemetryText;
4      public float speed = 0.1f;
5
6      private RectTransform telemetryTransform;
7
8      void Awake()
9      {
10         telemetryTransform = telemetryText.GetComponent<RectTransform>();
11     }
12
13     IEnumerator Start()
14     {
15         float height = telemetryText.preferredHeight;
16         Vector3 pos = telemetryTransform.position;
17         float s_pos = 0;
18
19         while (true)
20         {
21             telemetryTransform.position=new Vector3(pos.x, s_pos % height, pos.z);
22             s_pos += speed * Time.deltaTime;
23             yield return null;
24         }
25     }
26 }
```

The Awake function is called once during the lifetime of the script after all GameObjects are initialized. It is used to initialize variables in the script. The Start function is called at the moment in which a script is enabled, right before the Update method is called the first time. The return type IEnumerator for the Start function transforms it to a Unity Coroutine that allows for parallel actions to take place. Coroutines have the ability to halt execution and return control to Unity after which they resume control exactly where they left off in the following frame.

5.5 User Interaction

5.5.1 Man Machine Interface

Human interaction with the computer occurs through User Interfaces (UI) that are also called Man Machine Interfaces (MMI). They allow the user to monitor the system status or even to intervene in a process. The information or feedback can be provided through display panels, lamps, buttons or any other software visualization system that runs on a terminal. Design goals for User Interfaces dictate easy, efficient and comfortable methods to operate and control the machinery. [6]

5.5.2 Unity UI elements

Canvas

Unity utilizes Canvasses to create User Interfaces in applications. All UI elements are required to be inside the Canvas area. Additionally, UI elements need to be child objects of the Canvas GameObject. This relation is automatically established when creating a UI element in Unity. Canvasses feature three different Render Modes. The UIs in the virtual Control Room are designed to be diegetic interfaces, which means they are an integrated part of the virtual world. Therefore only the World Space Render Mode is relevant for this project. The World Space Render Mode provides the possibility for Canvases to be adjusted in size and moved anywhere in the scene. The main User Interfaces in this application are the monitors that display satellite telemetry data and behave in the same manner as all other GameObjects. The monitors in the virtual Control Room simulate simple computer terminals through which the user can monitor the satellite status. For this purpose Unity's Text UI suffice. Separate Text UIs are placed in front of each monitor to display the dynamic text-only content. To interact with the monitors and buttons in the scene through the Oculus Touch Hand Controllers, the Canvas Renderer has to be adapted. The default Canvas Raycaster script attached to the Canvas is required to be replaced with the OVRRaycaster script that is a prefab from the Oculus Integration. Furthermore the UIHelper prefab needs to be added to the scene via the Hierarchy window. This prefab includes a laser pointer that becomes visible when pointed at an element from the Canvas system. [6]

TextMesh Pro

To improve the performance of the application the simple Text UI should be replaced by Components of the type TextMeshProUGUI that can be found under GameObject->UI->TextMeshPro Text. The TextMesh Pro component is also designed to work with the Canvas system but uses a more advanced rendering technique called Signed Distance Fields (SDF). SDF uses a function that takes a position as its input. The function then delivers the distance from that position to the nearest part of a shape as an output. This method allows fonts to be no longer stored as Bitmaps and therefore the render quality is improved for larger sizes. UI Text components blur when they are stretched and resized. This problem is fixed by using TextMesh Pro. Next to the visual improvements TextMesh Pro also offers more control over the text Component via the Inspector and it offers a larger set of functions that can be accessed through scripting. [6]

EventSystem

Per default Unity adds an EventSystem element to the project once any UI is created. Every scene has at most one EventSystem. It handles all input events in the scene. The EventSystem coordinates events in the application based on user input. The input concerning the UI monitors in the virtual Control Room is custom input in the form of scripts that implement a UDP communication. The UDP script is attached to the OVRPlayerController and enables the communication between client and server. [6]

5.5.3 UDP implementation

To realize the virtual Control Room's communication system and to make the telemetry data of the satellite available inside the virtual environment, a UDP communication is established. This requires a UDP client and a UDP server. The server which is a simple console application, forwards the received telemetry data into the Control Room. The client or recipient of the data is integrated in the project's User Interface and makes the telemetry data accessible on the screens in the Mission Control Room.

For the implementation, services of the Microsoft .NET Framework are used. The Framework provides internet services that can be integrated into C# applications and therefore they can also be integrated into Unity assets in the form of C#-scripts. More precisely the implementation is realized through the UdpClient class of the System.Net.Sockets-namespace. This namespace provides developers with a managed implementation of the Windows Sockets interface (Winsock) for a tightly controlled network access. Network services for the implementation of the User Datagram Protocol are encapsulated in the UdpClient class. [36]

With the help of the UdpClient class, UDP Datagrams are send and received via the connectionless User Datagram Protocol. There is no need to create a remote host connection before sending the data as the protocol defines a connectionless communication system. The UdpClient class provides methods for a blocking, synchronous communication mode.

Client

The following code uses a UdpClient to listen for UDP Datagrams on port 11000. The listener receives strings, encodes them in the ASCII format and saves them in the telemetry variable that is afterwards printed on the screens in the Control Room.

```
1  public class client
2  {
3      private UdpClient listener;
4      private Thread thread;
5      private const String IPADDR = "127.0.0.1";
6      public string telemetry;
7
8      public client()
9      {
10         //initialise instance of UDPClient class that listens on given port
11         listener = new UdpClient(new IPEndPoint(IPAddress.Parse(IPADDR), 12000));
12
13         this.thread = new Thread(new ThreadStart(this.Execute));
14         this.thread.Start();
15     }
16
17     public void Execute()
18     {
19         //represent network endpoint as IP address and port number
20         IPEndPoint remoteIpEndPoint = new IPEndPoint(IPAddress.Parse(IPADDR), 11000);
21
22         //continuously receive telemetry data on Endpoint
23         while (true)
24         {
25             try
26             {
27                 Byte[] receiveBytes = this.listener.Receive(ref remoteIpEndPoint);
28                 telemetry = "Data received: " + Encoding.ASCII.GetString(receiveBytes);
29             }
30
31             catch (Exception e)
32             {
33                 Console.WriteLine(e.ToString());
34             }
35         }
36     }
37 }
```

Server

The UDP server that sends the data into the room, is also implemented using the UdpClient class. The message string specified in the console is send continuously. This instance of the UdpClient class is bound to the specified port number.

```
1  public class server
2  {
3      // set the IP-Address to the host device (currently local host)
4      private const String IPADDR = "127.0.0.1";
5
6      private UdpClient udpClient;
7      public String s;
8
9      public server()
10     {
11         //initialize instance of UDPClient class, bind it to specified local port number
12         udpClient = new UdpClient(11000);
13
14         //establish a host, using the specified IP address and port number
15         udpClient.Connect(IPAddress.Parse(IPADDR), 12000);
16
17         //continuously capture new data and send it to host
18         while (true)
19         {
20             s = Console.ReadLine();
21             System.Threading.Thread.Sleep(10);
22             SendData(s);
23         }
24     }
25
26     public void SendData(String data)
27     {
28         Console.WriteLine("Sending...");
29         try
30         {
31             this.udpClient.Send(System.Text.Encoding.ASCII.GetBytes(data), data.Length);
32         }
33         catch (Exception e)
34         {
35             Console.WriteLine(String.Format("Exception {0}", e.Message));
36         }
37     }
38 }
```

Chapter 6

Conclusion

6.1 Results

Evidently, this thesis proves that the implementation of a virtual Control Room for satellite missions is a complex undertaking which requires a great amount of consideration. The paper establishes the advantages regarding a virtual Control Room as opposed to the traditional facilities. Naming only a few; Being able to access the virtual operating room from anywhere in the world, cutting building costs and being able to easily alter and improve the Control Room. The implementation delivers the first version of a functional Mission Control Room for satellite missions with the requirements given in the beginning. The goal to build a text-based telemetry interface was successfully achieved. The communication via the UDP-Protocol proved to be highly suitable for the telemetry data since it supports real-time mission requirements. The Unity Game Engine turns out to be a satisfactory development environment for the purposes specified in this thesis. Unity, used in combination with Blender offers the necessary tools to implement the desired software.

However, for several parts of the implementation it turned out to be a great challenge to keep the application performant while achieving the given requirements. In the end the User Interfaces and their attached C#-scripts had to be simplified to ensure smooth rendering by the Oculus Rift headset. In addition, the Blender models were revised several times to balance the desired design with performing models in the Unity Editor. The Mesh Colliders of the objects in the room had to be created manually to simplify their shapes and thereby reduce CPU and GPU overhead.

The following images display the first version of the implemented virtual Mission Control Room with the Telemetry interface.

Figure 6.1: Virtual Mission Control Room - View to the left

Figure 6.2: Virtual Mission Control Room - View straight ahead

Figure 6.3: Virtual Mission Control Room - View to the right

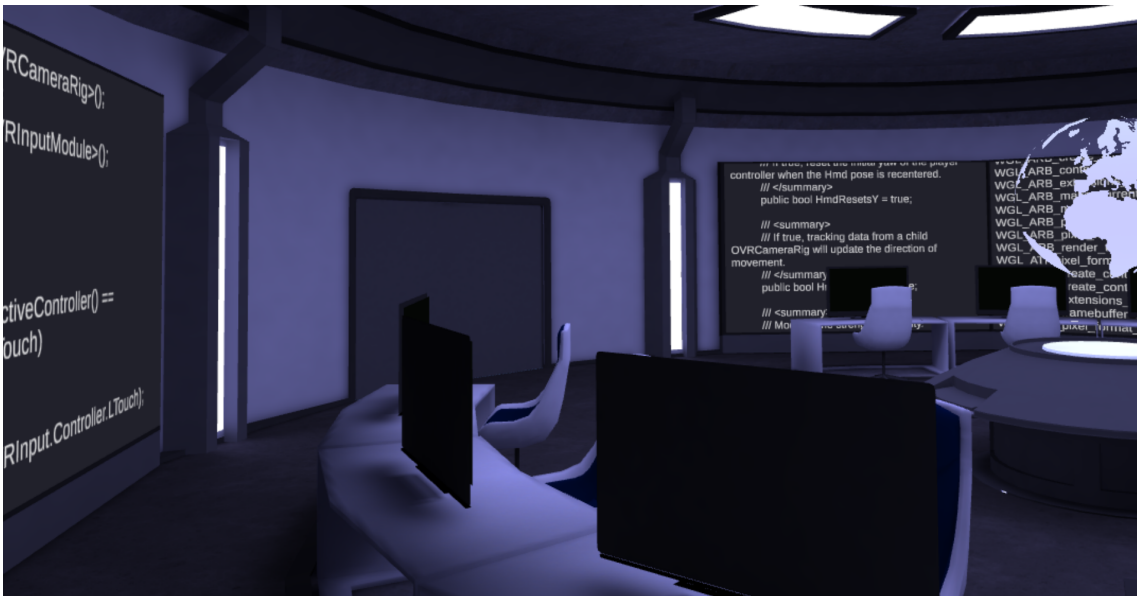


Figure 6.4: Virtual Mission Control Room - View of the entry doors

Figure 6.5: Virtual Mission Control Room - View of the holographic earth model

Figure 6.6: Virtual Mission Control Room - View of individual workplaces

6.2 Outlook

After carefully examining the demands of Mission Control Rooms some elements for further development are found. An adequate method to support the handling of Telecommands needs to complement the Telemetry system. The Oculus Touch Controllers used in this version of the project may not provide enough functionality to produce appropriate user input. Standard computer keyboards may be the preferred choice. Additionally multiplayer support needs to be implemented to enable work in teams in the virtual Mission Control Room. This in turn requires the implementation of voice-loops or any other favoured form of spoken communication between the separate users within the Control Room. At this point the implementation of character representation in the VR environment could be considered for a more wholesome work experience. Another important aspect for future development is the implementation of various visual User Interfaces. For now only text-based interfaces exist. A 3D representation of the satellites orbit is an example for further development. This and other visual representations throughout the room can greatly increase the rooms vividness and thereby promote productivity. Powerful visual tools have the ability to facilitate data representation and data analysis. Therefore graphical user interfaces in the room are needed to complement the existing text interfaces. Considering the implementation in Unity it must be noted that graphs in Unity are handled differently than other User Interfaces like the text-based UIs. A useful tool to create graphs from the telemetry data within Unity during the runtime is Unity's LineRenderer. The LineRenderer is a Unity component that draws straight lines between custom points in the 3D environment. LineRenderer is not a standard User Interface and is therefore not listed under UIs in the Menu. It can be found in the Unity menu under GameObjects>Effects>Line. The data for the graph can simply be inserted as an array of coordinate pairs in the Inspector window. With the help of scripts and the UDP communication structure, these points can be fitted dynamically during the runtime to correspond to the continuously changing telemetry data.

Bibliography

[1] Tara J. Brigham. Reality Check: Basics of Augmented, Virtual, and Mixed Reality. *Medical Reference Services Quarterly, 36:2*, pages 171–178, 2017. doi: 10.1080/02763869.2017.1293987.

[2] Muhanna A.Muhanna. Virtual reality and the cave: Taxonomy, interaction challenges and research directions. *Journal of King Saud University - Computer and Information Sciences*, 27: 344–361, July 2015.

[3] Grigore C. Burdea and Philippe Coiffet. *Virtual Reality Technology.* Wiley publishers, Hoboken, New Jersey, 2003.

[4] Christoph Anthes, Rubén García Hernandez, Markus Wiedemann, and Dieter Kranzlmüller. State of the Art of Virtual Reality Technologies. 03 2016. doi: 10.1109/AERO.2016.7500674.

[5] S. Manjrekar et al. Cave: An emerging immersive technology – a review. *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 131–136, March 2014. doi: 10.1109/UKSim.2014.20.

[6] Unity Technologies. Unity user manual (2019.3). URL `https://docs.unity3d.com/Manual/index.html`. accessed: May 2020.

[7] S. M. LaValle et al. Head tracking for the Oculus Rift. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 187–194, May 2014. ISSN 1050-4729. doi: 10.1109/ICRA.2014.6906608.

[8] Facebook Technologies, LLC. Oculus Rift CV1, . URL `https://www.oculus.com/rift/?locale=de_DEoui-csl-rift-games=star-trek`. accessed: January 2020.

[9] Facebook Technologies, LLC. Oculus Touch Controllers, . URL `https://scontent-dus1-1.xx.fbcdn.net/v/t39.2365-6/24046862_268291453697978_381270238345297920_n.jpg?_nc_cat=110_nc_sid=ad8a9d_nc_ohc=q4PjUIP5nTsAX9Sk-j2_nc_ht=scontent-dus1-1.xxoh=9ee89338805e18e6444ee45eeeb59867oe=5E8F8FAE`. accessed: January 2020.

[10] Facebook Technologies, LLC. Oculus-Sensor, . URL `https://scontent.oculuscdn.com/v/t64.5771-25/12139310_1705050129824606_4730046241839251456_n.jpg?_nc_cat=108_nc_sid=ad8a9d_nc_ohc=Yc3mcp8RSh0AX9ncq4__nc_ht=scontent.oculuscdn.comoh=2b47177f51af42221a1f1ba8da9585b3oe=5ECE5672`. accessed: January 2020.

[11] Oculus VR. Oculus rift: Recommended pc specification. URL https://www.oculus.com/rift/oui-csl-rift-games=mages-tale. accessed: 20 March 2020.

[12] Michael W. Mcgreevy. The virtual environment display system. *Computer Programming and Software, National Aeronautics and Space Administration*, pages 3–9, march 1991. doi: 19910013709.

[13] R. Bowen Loftin and Patrick J. Kenney. Training the hubble space telescope flight team. *IEEE Comput. Graph. Appl.*, 15(5):31–37, September 1995. ISSN 0272-1716. doi: 10.1109/38.403825.

[14] J. Rossmann and Bernd Sommer. The Virtual Testbed: Latest Virtual Reality Technologies for Space Robotic Applications. 01 2008.

[15] Hirofumi Aoki, Ryuzo Ohno, and Takao Yamaguchi. The effect of the configuration and the interior design of a virtual weightless space station on human spatial orientation. *Acta astronautica*, 56:1005–16, 05 2005. doi: 10.1016/j.actaastro.2005.01.028.

[16] Arielle Samuelson. Mars Virtual Reality Software Wins NASA Award, October 2018. URL https://mars.nasa.gov/news/8374/mars-virtual-reality-software-wins-nasa-award/. accessed: 4 March 2020.

[17] LLC. Facebook Technologies. Apollo 11 VR, 2016. URL https://www.oculus.com/experiences/rift/937027946381272/?locale=de_DE. accessed: 4 March 2020.

[18] About virbela, 2019. URL https://www.virbela.com/about. accessed: 2 April 2020.

[19] Hakan Kayal. Skript zur Vorlesung Raumflugbetrieb. *Julius-Maximilians Universität Würzburg*, 2019. Fakultät für Mathematik und Informatik, Lehrstuhl für Informatik VIII.

[20] Deutsches Zentrum für Luft-und Raumfahrt e. V. (DLR). Mission Operations. URL https://www.dlr.de/rb/en/desktopdefault.aspx/tabid-4649/11203_read-25616/. accessed: 7 February 2020.

[21] Deutsches Zentrum für Luft-und Raumfahrt e. V. (DLR). Columbus-Kontrollzentrum. *Raumflugbetrieb und Astronautentraining*, February 2017.

[22] R. A. Hoover. The Houston Mission Control Center. *IEEE Transactions on Aerospace*, AS-3 (2):126–131, June 1965. ISSN 2374-944X. doi: 10.1109/TA.1965.4319792.

[23] Michel Denis. Steuerung und Betrieb von Raumfahrt-Missionen im Orbit und im Deep Space, December 2010. URL https://raumzeit-podcast.de/2010/12/03/rz004-operations/.

[24] European Space Agency. ESOC Mission Control Room. URL http://www.esa.int/var/esa/storage/images/esa_multimedia/images/2019/01/esoc_mission_control_room/19183509-1-eng-GB/ESOC_Mission_Control_Room_pillars.jpg. accessed: March 2020.

[25] NASA. JSC Johnson Space Center. URL https://www.nasa.gov/sites/default/files/styles/full_width_feature/public/jsc2014e077199.jpg. accessed: March 2020.

[26] DLR. German Space Operations Center. URL https://www.dlr.de/content/en/images/2011/1/activity-in-mission-control-during-the-d-2-mission_546.jpeg?__blob=normalv=11__ifc1 024w. accessed: March 2020.

[27] Emily Patterson, Jennifer Watts-Englert, and David Woods. Voice Loops as Coordination Aids in Space Shuttle Mission Control. *Computer supported cooperative work : CSCW : an international journal*, 8:353–71, 02 1999. doi: 10.1023/A:1008722214282.

[28] Gorry Fairhurst. The User Datagram Protocol (UDP), November 2008. URL https://erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html. accessed: 26 March 2020.

[29] Joe Touch et al. Service Name and Transport Protocol Port Number Registry. URL https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt. accessed: 23 March 2020.

[30] A. Malhotra, V. Sharma, P. Gandhi, and N. Purohit. UDP based chat application. 6:V6–374–V6–377, April 2010. doi: 10.1109/ICCET.2010.5486192.

[31] M. Nabiyouni, A. Saktheeswaran, D. A. Bowman, and A. Karanth. Comparing the performance of natural, semi-natural, and non-natural locomotion techniques in virtual reality. *2015 IEEE Virtual Reality (VR)*, pages 243–244, March 2015. ISSN 2375-5334. doi: 10.1109/VR.2015.7223386.

[32] Blender Guru. Blender Beginner Tutorial Series. URL https://www.blenderguru.com/tutorials/blender-beginner-tutorial-series. accessed: January 2020.

[33] Poliigon. Brush Fabric Wrinkles Couch 04. URL https://www.poliigon.com/texture/brush-fabr ic-wrinkles-couch-04. accessed: 17 January 2020.

[34] Rob Garlington. What are the different texture maps for? URL https://help.poliigon.com/en/articles/1712652-what-are-the-different-texture-maps-for. accessed: February 2020.

[35] LearnOpenGL. Normal Mapping. URL https://learnopengl.com/img/advanced-lighting/normal_mapping_comparison.png. accessed: January 2020.

[36] Microsoft. Network Programming in the .NET Framework. URL https://docs.microsoft.com/en-us/dotnet/framework/network-programming/. accessed: March 2020.