UNIVERSITÄT
BERN

University of Bern
Computer Graphics Group

Bachelor Thesis

Creation and modification of 3D meshes in virtual reality

OpenVolumeMeshVR

by Marcel Zauder

supervised by
Prof. Dr. David Bommes

Abstract

Abstract

Contents

Chapter 1

Introduction

Virtual Reality (VR) is one of the most up-and-coming technology sectors in informatics and already today a very important component in many different application areas. However, the search term "virtual reality" gives nearly 1'390'000'000 hits (January 17, 2022). Also in the media VR covers more and more broadcasting time and therefore its acknowledgement is rapidly increasing. Due to these many people nowadays know the term VR, but barely more than superficially and only a few of them have ever ventured into virtual reality. Even in modern companies, as architecture bureaus and the automobile industry, VR is used to create 3D models of prototypes so it can be shown to investors or to highlight problems or hints to make the product better. However, the biggest application of VR is in the gaming industry, in which this type of controlling is very well received. Therefore, it is not surprising that many companies are dealing with this area in order to further develop this technology.

1.1 Methodology

In this bachelor thesis a brief overview about VR is given and how it is used today. Especially highlighted is the usage of it in terms of the creation of 3D meshes. Considering this, already published programs from Google and Microsoft are inspected and their pros and cons are determined by executing a survey about intuitiveness and usability. Also the used Unreal Engine 4 and OpenVolumeMesh are introduced and presented.

1.2 Motivation and Goal

OpenVolumeMesh is a data structure that can represent heterogenous 3-dimensional polytopal cell complexes ([KBK13], Section 2.4). Until the start of this bachelor thesis in order to create a new OpenVolumeMesh mesh it either is converted from an already existing 3D volume mesh or must be generated by coding. Also the viewing of individual meshes was very difficult to achieve due to OpenVolumeMesh and OpenFlipper not having a particularly good renderer.
Therefore the goal of this bachelor thesis is to create an application that is capable of generating and modifying simple OVM meshes and also load and view these within a virtual environment. This application can both be used with a "Virtual Reality Head-Mounted Display" or with the use of mouse and keyboard on a desktop computer. This application will be implemented with the use of the Unreal Engine because it is a C++ based game engine which was adventageous to the integration of OpenVolumeMesh, because its source code is also written in C++.

1.3 Limitations

Because the goal of this bachelor thesis is to develop an application with which it is possible to create simple meshes based on the OpenVolumeMesh data structure, the program will not be able to display meshes with an entity number greater than 5'000. A higher number of entities can be reached if for example the engines settings themselves are overwritten, like culling, render distance or the display entities are altered to more simpler structures. In the end this will only increase the limit to a maximum

of 2'162'688 objects because this is the limit set by the Unreal Engine itself.
Unfortunately due to the Covid-19 pandemic and the associated inaccessibility of the VR-equipment the Virtual Reality implementation is not as refined as possible and does lack in terms of bug safeness and working features. However, during this bachelor thesis the ability of using this program with mouse and keyboard was implemented, which is working as intended.

Chapter 2

Research

2.1 Virtual Reality

2.1.1 Beginnings of VR in Computer Science

The first device related to Virtual Reality was the "Sensorama simulator" invented by Morton Heilig [Hei61] in 1961. This device showed 3D movies while the user was sitting in a vibrating chair, listening to stereo sound and also feel wind as well as smells which were generated by the simulator according to what was shown on the screen. This project was due to the lack of financial backing abandoned by Heilig shortly after its invention, but it is most often regarded as the first approche to the Virtual Reality realm. Simultaneously Heilig was working on a head-mounted display, referred to as "Telesphere Mask". Both of these patents were cited several hundred times underlining the importance of the early work by Heilig [SKB15].
Later in 1968, Ivan Sutherland developed and created an own idea of a head-mounted display which was still supported by arms hanging from the ceiling due to the weight of the heavy device. The goal of this technology was so the user can control the matter within the virtual reality to his liking. Sutherland was comparing this environment to "the wonderland in which Alice walked" [Sut65].
In terms of virtual reality controllers G. Zimmermann developed and patented a glove in 1982 with flexible optical sensorswhich could measure determine the position, especially the bending, of the fingers. This idea was taken up by G. Grimes when he developed the "Data Glove" at the AT&T Labs in 1983. Additionally to the position tracking the "Data Glove" was also able to give haptic feedback through small actuators placed in the glove itself. However, the concept of tactile, immersive feedback within input devices was later scratched by Mattel and Nintendo which where the first distributors of low-cost data-gloves in 1989 [SKB15].
The Virtual Reality headset with motion tracking and display, as they are known today, where prototyped by SEGA in 1993, exclusively made for the SEGA Genesis console. But in spite of that, due to technical difficulties this concept never made it to the store shelves and was a big flop for SEGA. During the late 90s and early 2000s different tech firms and individuals tried to develop a head-mounted VR display, with more or less success, until in 2012 Palmer Luckey launched a Kickstarter for his idea that came up in 2010 to create kit VR headset together with John Carmack. This crowd-sourcing campaign gathered approximately 2.5 million dollars and can be defined as the real start of Virtual Reality headsets - the so called "Oculus Rift". In late 2016 and early 2017 the sales figures of head-mounted devices (HMD) were exploding. The Oculus Rift and the HTC Vive opened the floodgates and the market for HMDs become greater and greater [Soc].

2.1.2 Today

In today's world the Virtual Reality finds use for many different applications [Boa12]:

1. In *Healthcare* Virtual Reality is used for training. Specialists can practice very precise operations without being in an actual emergency. Furthermore it can also be used in therapy to place patients that are suffering from phobias for instance in a controlled environment.
2. Because *Space and Military Organisations* are most often operating in very dangerous environments that are not accessible at all time. Therefore VR is used to prepare missions or even to

help soldiers to help with trauma and PTSD likewise to the healthcare use.

3. In the *Architecture and Automitive Industries* VR is used to view prototypes or to test new models or buildings in simulated situations. Especially for the automotive industry VR becomes more and more important as the demand for smart cars is increasing.
4. The strongest presence VR has, is in the *Entertainment Industry*. There are many different HMD models on the market right now (HMD Devices). These devices are also finding their way into cinemas and even theme parks like it is used in Europa Park in Germany at the Eurosat Coastiality coaster.

There are more examples but all of them have in common that they are used to create a virtual environment in which the user can either be trained with heavy machinery to ensure safety when it is later used or view a product, like an actual object or even a preview for the next vacation [Tri].
Another use case for VR is also in the game development industry in which the virtual environment is used to create 3D assets that are later placed in the game environment, like buildings, animals, etc. Some examples for these kinds of software are later introduced in Chapter 2.2.

2.1.3 Future

The HMDs will be improved to have better displays (already today there are 8k displays built in some models) and use much more powerful processors. Also the need for cables should be neglected such that the user can move more freely. Also the visual interfaces are improved such that the effects, like motion sickness, that are sometimes produced in users are minimised.
Already topday, but more so in the future, the Virtual Reality will be merging with the real world forming the so called Mixed Reality. In those applications physical and digital objects will co-exist and interact in real-time. An example for Mixed Reality devices is Google Glass, which were glasses that showed additional information depending on where the user is looking. Those kinds of devices will be come up more and more and even Apple is rumoured to develop a Mixed Reality device as a replacement for the IPhone [Soc].

2.2 Provided Software

In the following section a collection of 3D graphic software is presented and examined. In each of those applications it is possible to create 3D structures some rather on an architectural level some on an artistic level like drawing or modelling. Also the implementation of equal tools or the approach to the same task sometimes differ on a highly basis.
Ideas and suggestions, which are drawn from these, then form the foundation of the program developed during the bachelor thesis.

2.2.1 Google Blocks [GB4]

Google Blocks is, compared to the other applications listed here, not a comprehensive but a very easy-to-use program. There are 6 main features at the moment: placement of ready-made 3D objects, drawing freehand lines, coloring, modifying, moving, and deleting objects and structures. Within these are also subfunctions, such as grouping, copying, and scaling up or down, available. The resulting scnenes can then be exported as .obj file or as an animated gif. There is also an own marketplace where creations of other users can be viewed and imported as well as own structures can be uploaded and published.

2.2.2 Google Tilt Brush [GTB]

Google Tilt Brush offers the possibility to draw and paint in VR. Unlike the other softwares, here the artistic aspect and not the creation of volumetric meshes or 3D structures is at the point of

interest. Therefore a work of art can be created with many different types of "dynamic brushes". A big variation provides the virtual palette which includes oil, ink, and water colors besides paints with effects as fire, glow, and smoke. Although it is possible to incorporate shapes into the scenery, this option is of less importance.
For executing showcases to a group of observants in-game cameras can be placed, so creations can be viewed from different perspectives and not only in first-person.

2.2.3 Masterpiece VR [MVR]

In Masterpiece VR 3D sculptures can be modelled by using "virtual clay". It is mainly used to create characters, objects, and asset visuals which then can be exported to Unity or Marmoset for example to be integrated in games. It is also possible to create very detailed objects due to the implementation of "pinch" and "noise" tools which roughen the surface of the clay sculptures. In addition the environment can be manipulated so each creation can be viewed under different light conditions and in various settings.
Furthermore Masterpiece VR has the functionality to add collaborators and spectators to experience VR modelling in multiplayer.

2.2.4 Microsoft Maquette [MSM]

Microsoft Maquette is one of the more feature-heavy applications that was tested within the scope of this bachelor theses. Primarily, it is used to quickly create spatial prototypes which can then be used in other application like in the Game Engine Unity for (environmental) assets. Already created objects can be saved in an "inventory"-like menu tab and easily accessed and multiplied within Maquette itself making it much easier to create repeating patterns. It is also possible to place text fields and generate 3D texts within the virtual space in order to create 3D assets which can then be placed in videogames for example. Because the content which is generated with Microsoft Maquette can be exported in many different file types, like .fbx, .glb, and .gltf, these objects can then be viewed and used in many different applications like Blender or the Unity game engine.
Since November 2021 Microsoft does not actively develop this application anymore and uses the feedback that was gathered during the Beta phase to be applied in future Mixed Reality applications.

2.3 Unreal Engine 4 [UE4]

Unreal Engine 4 (UE4) is the latest version of the game engine developed by Epic Games. The first version of Unreal Engine appeared together with the FPS game Unreal in 1998 established by Tim Sweeney, the founder of said company. Due to be written in C++ and also being source-available since 2014 it can be implemented and imported very easily on many different platforms. By already supporting a huge amount of these platforms, it is a highly used engine for programming and setting up a multitude of various games.
The framework consists of a graphic engine and its related script language UnrealScript among many other things. Also UnrealEd, a level editor, is provided with which the layout of maps and levels are changeable on the fly. With its possibility of blueprint visual scripting even complex prototypes can be intergrated in the virtual world without needing a high ackowledgement of coding. This makes Unreal Engine even suitable for the average programmer. By visualizing the different behaviours and dependencies the debugging is also quite easy.
Because nowadays multiplayer experience is frequently used, adds even a scalable server/client architecture to get the ability to implement this component. Additionally, Unreal Engine has gathered a huge community behing itself which means that many different tools and feature integrations can already be downloaded and used via GitHub or the Marketplace.
Through close collaboration between Epic Games and world's leading hardware and software manufac-

turers, UE4 is also geared towards the use of Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) and therefore continues to evolve and improve in these areas.

2.4 OpenVolumeMesh [OVM]

OpenVolumeMesh (OVM) is a generic data-structure based on OpenMesh in order to handle arbitrary polyhedral meshes developed by M. Kremer, D. Bommes, and L. Kobbelt in 2013. It uses the idea of half-edges and accordingly creates half-faces of different orientations (Figure 2.1). The biggest advantage of OVM, however, is that the data structure is equivalent to an indexed array, allowing access to run in constant time complexity.

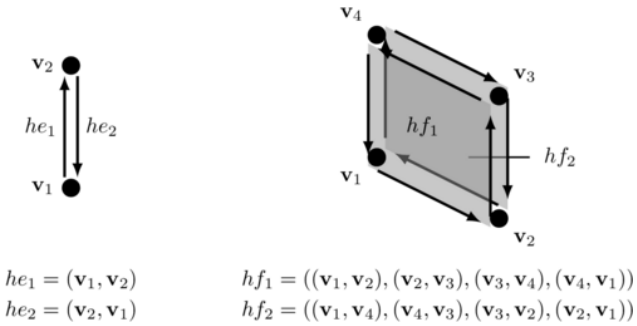


Figure 2.1: Illustration of Halfedge and Halfface

An edge within a mesh is split up into two half-edges from which each contains a start and an end vertex. Several of these half-edges arranged in a loop make up a half-face. The frontside of a half-face is defined as the side from which the incidents of all half-edges is directed in a counter-clockwise orientation. Hence, also each face is made up from two half-faces.

2.5 Conclusion

Chapter 3

Intuition Study

3.1 Concept and Execution

To decide which specifications the new program should include and how they should be implemented, a test was performed in which two of the given programs, Google Blocks and Microsoft Maquette, were compared against each other and their intuitiveness was determined. Testworlds for Google Blocks as well as for Microsoft Maquette were created conducive to identify advantages and disadvantages of the individual features and to achieve the best possible implementation of the respective functionality. Within these testworlds the user was faced with different tasks that should be accomplished, i.e. recreate the object the author has placed. The set of tasks consisted of the following:

- Moving through the virtual environment
- generating any object
- delete objects
- placing pre-made objects (spheres, cubes, etc.)
- place objects with the snapping and align feature
- color the objects
- copy and paste, move, scale and group objects
- modify (extruding, subdividing, etc) predefined objects

The tasks between these testing areas were kept as similar as possible so the measurement of the discrepancy of those two programs and their components is representative. In order to rule out that a program benefits by being tested secondarily, and thus previously learned could distort the test results, the process is always varied, so that half of the probands will start with Google Blocks and the other half with Microsoft Maquette.
While the test persons are completing the tasks, notes are taken by the author wherein the intuitiveness of each service is checked according to how fast the functionality is discovered and understood as well as occuring problems are highlighted. At the end of this process an evaluation sheet, which contains questions about how intuitive and simple the different tasks were and which of the programs they preferred, is filled in (Appendix A).

3.2 Tables

In the following tables the results of the tests are listed. The scale for the movement part has been chosen differently (1-6), as it focuses on a more specific view to be able to study this functionality very closely and to determine what is perceived as a rather better type of movement. Concerning the other parts the point of interest was set on which program implemented each particular mode better. For illustrational purposes, the value of 1 has been assigned to the selected option.
In the end the average value for each component was computed, so the usability and intuitiveness can be easily evaluated.

General															**average ∅**
Impression	What application was perceived as the better														
Google Blocks	1	0	1	1	1	1	0	1	0	1	1	0	0	0	0.57
Microsoft Maquette	1	1	0	0	0	0	1	0	1	0	0	1	1	1	0.50
Simplicity															
Google Blocks	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0.93
Microsoft Maquette	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0.13

Table 3.1: General Evaluation

Movement															**average ∅**
Intuitiveness															
Google Blocks	3	2	6	6	3	5	4	4	5	5	5	4	6	4	4.43
Microsoft Maquette	3	2	5	4	2	4	4	2	4	1	3	5	5	3	3.36
Impression	How well was the form of movement perceived														
Google Blocks	4	1	6	5	6	5	3	5	5	5	5	4	5	4	4.50
Microsoft Maquette	5	1	5	4	4	4	3	2	5	2	4	5	6	5	3.93

Table 3.2: Movement Evaluation

Object Interaction															**average ∅**
Placing															
Google Blocks	1	0	1	1	1	1	0	1	0	1	1	1	1	1	0.79
Microsoft Maquette	1	1	0	1	0	0	1	0	1	0	0	0	0	0	0.36
Moving	only Google Blocks implemented an explicit mode														
Google Blocks	1	0	1	1	1	1	1	1	1	1	1	0	1	1	0.86
Microsoft Maquette	1	1	0	0	0	1	0	0	0	0	0	1	0	0	0.29
Snapping															
Google Blocks	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0.71
Microsoft Maquette	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0.43
Deleting															
Google Blocks	1	0	1	1	0	1	1	1	1	1	1	0	0	0	0.64
Microsoft Maquette	1	1	0	1	1	0	1	0	1	0	1	1	1	1	0.71
Coloring															
Google Blocks	1	0	0	1	1	1	0	0	0	1	1	0	0	0	0.43
Microsoft Maquette	0	1	1	1	1	0	1	1	1	0	0	1	1	1	0.71
Modifying	only Google Blocks implemented an explicit mode														
Google Blocks	1	1	1	1	1	1	1	0	0	1	1	0	1	1	0.79
Microsoft Maquette	1	0	0	1	0	0	0	1	1	0	0	1	0	1	0.43
Grouping	only Microsoft Maquette implemented an explicit mode														
Google Blocks	1	1	0	0	0	0	1	1	1	0	0	1	0	0	0.43
Microsoft Maquette	1	0	1	1	1	1	0	0	0	1	1	0	1	1	0.64
Copying	only Microsoft Maquette implemented an explicit mode														
Google Blocks	0	1	0	1	0	1	1	1	0	0	0	1	0	0	0.43
Microsoft Maquette	1	0	1	1	1	0	1	0	1	1	1	0	1	1	0.71

Table 3.3: Object-Interaction Evaluation

Interface															average ∅
Google Blocks	0	0	1	1	1	1	0	1	0	1	1	0	1	0	0.57
Microsoft Maquette	1	1	0	0	0	0	1	0	1	0	1	1	1	1	0.57

Table 3.4: Interface Evaluation

3.3 Results and Evaluation

3.3.1 Impression and Layout

In general neither program is significantly better or worse, nevertheless, a few discrepancies can be recognized between some functionalities. The main difference is within the simplicity, because Google Blocks has much fewer editing tools than Microsoft Maquette and is therefore considerably more surveyable and incomplex (referring Table 3.1).
Concerning the most interesting point, changing the in-game position, the survey shows, that Google Blocks is slightly easier to understand and to work with. However, moving through the world using the "swimming" concept of Maquette is not considered the best way to implement this essential part (referring Table 3.2). Besides moving through the entire world using that concept often lead to motion sickness, because the brain assumed moving through space without actually moving the body. A huge problem for both designs is that the "Grab" buttons, which are located on each side of both controllers, are difficult to use and not very handy. Therefore, these should only be used sporadically.
In terms of the interface and selection menu none of these programs is especially better or worse than the other one (referring Table 3.4). Nevertheless, it has been observed that the graphical display of the menu, which is located on the side of one controller, has often restricted and reduced the working area. To prevent this, it would be appropriate to implement a hiding option or place it so that it does not interfere with the user. A good example is Google Tilt Brush, where the selection menu is located above the controller.

3.3.2 Object Interaction `(referring Table 3.3)`

A general overview of the object interaction statistics suggests by implementing an explicit mode the associated tasks were perceived as much more pleasant and easier to use. This is also reflected in the fact that when one program had implemented a corresponding mode, whereas the other did not, that program was most likely to be considered more intuitive in that particular area. Therefore, this leads to the conclusion that for each interaction a demarcating mode should be implemented in order to ensure a high degree of user-friendliness.
Placing given objects and shapes is quite similar in both tested programs, whereas generating free-hand lines is much more easier in Microsoft Maquette because a more meaningful signifier is used. But in the study Google Blocks received a better rating. A reason for this is a much smaller, and therefore clearer, collection of given objects. Also switching between the different objects is easier in Google Blocks by swiping through using the touchpad. In Microsoft Maquette on the other hand every shape must be selected individually from the selection menu which is more inconvenient and time consuming.
For moving objects Google Blocks implemented a specific "Grabbing"-mode so it was easier for the probands to figure out how to manipulate the position of structures. On the other hand Microsoft Maquette uses the "Grab"-buttons which intuitively makes sense because users are actually grabbing the thing. However, since the "Grab"-buttons are difficult to handle, the useability of this feature was considered rather bad. Therefore Google Blocks performed much better in that section and a significantly larger number of subjects considered it more pleasant.
The applications include two different methods for implementing rotational snapping and alignment.

In Google Blocks the left-handed trigger must be kept pressed, whereas in contrast Microsoft Maquette uses an additional button on the right-handed touchpad to toggle this mode on and off. The study found that the former was more comfortable to use, as the user did not need to check whether the mode is toggled on or off.
There are two different variants for deleting objects in both softwares: On the one hand an independent mode is implemented, on the other hand, the object can be thrown away so it disappears. Especially the "Disposal"-option was considered very intuitive and useful. Since this feature is much more easily to apply and more reliable in Microsoft Maquette than Google Blocks, it was slightly better rated.
Coloring in Google Blocks holds the problem in that the color palette is located on the back of the selection menu and is therefore not directly visible to the user. Consequently, a lot time was needed to find it, which ultimately had a negative impact on the user's opinion of this part. In Maquette the color selection was much more extensive, due to the implementation of an RGB palette. Also an eyedropper was added, whereby a color selection of already colored objects was possible. However, this involved the problem that the extraction line, which had to be directed onto the object to select the color, was not recognized as such, which made the handling of this option slightly difficult. Nonetheless, Microsoft Maquette did better in this area due to the greater color choice and the more convenient operation.
Exclusively Google Blocks realized an explicit tool for modifying, as subdividing, extrude, and extract, objects and their assets, while in Microsoft Maquette only the length, width and depth of a structure are changeable. Also, handling the feature in Maquette, which requires a synchronized movement of both controllers, was not without problems. Accordingly, the test subjects perceived Google Blocks as a much more pleasent and better option for this type of interaction.
For grouping and copying of several structures and objects both software have implemented an associated tool. In Google Blocks, however, this tool is hidden inside the "Grab" feature, which means it is not directly recognized. In contrast, in Microsoft Maquette, there are even several fields, within the copy function, into which object groups can be copied in order to quickly duplicate them. These details ultimately lead to Maquette performing better in the survey for both options, and more test subjects found it more easier to use.

3.3.3 Conclusion

The survey generally shows that each function is much more easier to use if there is a seperate, special mode for it. In addition, each of these modes should be clearly described and the functionality easy to recognize and quickly understood. Also, within a mode not many other specifications should be present, as this leads to a higher degree of complexity of the program.
From the testers' reports and the author's notes, it further emerges that the "Grab"-buttons are very difficult to use, and thus an "over-use" of them is considered negative. On the other hand, the touchpad is acknowledged as a very good control option, so a good implementation of this can lead to a high level of usability. What is also noticeable is that many testers favor haptic feedback as very useful and should therefore be taken into account.
Another problem was that the orientation of the interface menu often serverely restricted the workspace. Thus, an implementation as found in Google Tilt Brush with this menu above the secondary controller should be considered. Also a toggle mode to hide and show the menu can counteract this problem.
Ultimately, this study has produced some ideas for implementing various specifications and has been able to inform about the differences in the usability of the modes.

Drawn Conclussions for OpenVolumeMeshVR Implementation

The most important thing that could be drawn from this intuition study was that for each task or feature the application should provide a separate mode to accomplish it. These different modes should also not be too extensive and not provide any additional features that might not be directly accessible but need a certain keyboard or controller shortcut in order to be used.

Additionally a toggleable menu will be implemented such that the working space is not restricted or reduced by letting the user decide when the it is visible and when it should be hidden. Furthermore this also gives the advantage that the buttons can be made much larger to ease the selection of the different modes. On the other hand when in the mouse-keyboard control mode a Heads-Up Display has been integrated which shows at all time which modes can be used, with which number-key they can be accessed with, and in which mode the user currently is.

In terms of movement it was decided that the user can only pass through the virtual environment by actually walking. In order to be able to see the whole object it was determined that it is a better approach to grab the whole mesh and move it by itself. For the mouse-keyboard control this part does not matter and the usual "WASD"-movement was implemented.

Chapter 4

Documentation OpenVolumeMeshVR

OpenVolumeMeshVR is a sketch tool in which simple meshes can be constructed and viewed. These so created meshes can then be saved as OpenVolumeMesh files.

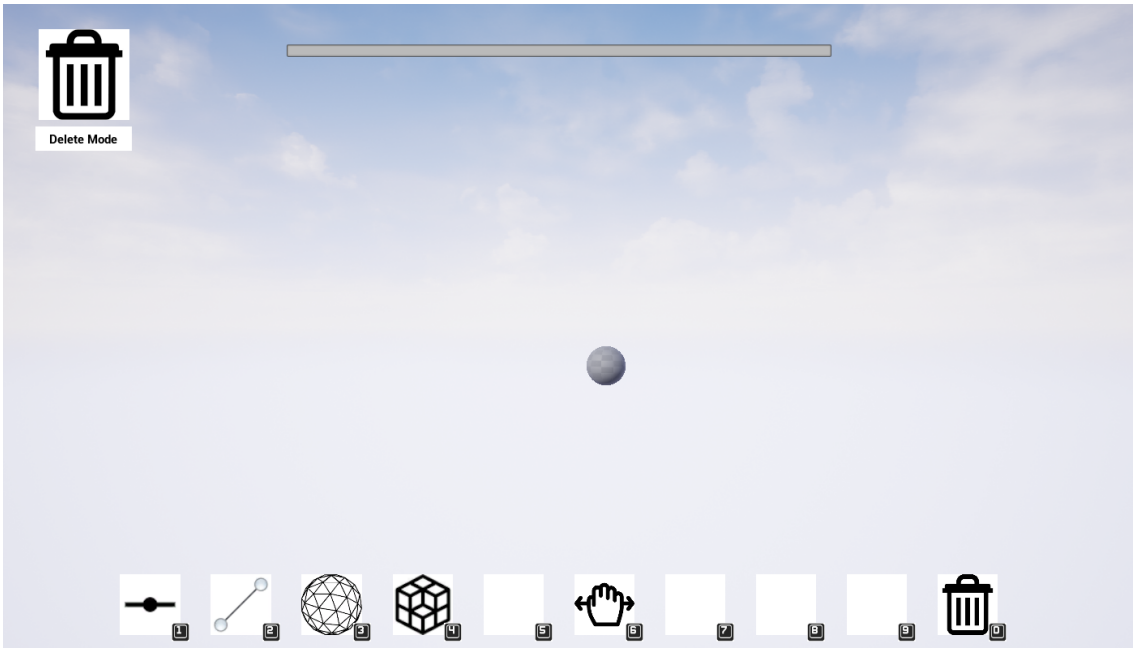


Figure 4.1: View of OpenVolumeMeshVR using Mouse-Keyboard (MK) Control

The ball which can be in the middle of Figure 4.1 will be referred to as the PlacementChaperone. This is the OpenVolumeMeshVR version of the cursor with which the selection, placing and deletion of the different entities can be performed. The distance of it from the player can be altered by using the mouse wheel. When OpenVolumeMeshVR in VR mode the PlacementChaperone will be paret of the right-hand controller. The player can be moved around by using the WASD-keys and Space/Alt to move up and down when in "MK-Control" or by actually moving around when in "VR-Control". To confirm a placement of an entity or likewise either use the left mouse button or the B-Trigger on the right VR controller. The menu can be entered by either using the Escape or "M" key on the keyboard or the B-trigger of the left VR controller. In order to generate and modify OVM meshes different modes have been implemented. On the bottom the already implemented features are indicated and the associated keys with which the mode can be chosen are displayed.

4.1 Spawning and Placing Entities

When placing a vertex a representation of it is spawned in the virtual world and the vertices' position is handed over to the OVM data struct.
For placing an edge there are several different ways to acoomplish that task. If no pregenerated vertex is selected a new vertex will be spawned and a preview of the edge can be seen connecting the newly generated vertex with the PlacementChaperone. When a vertex is selected no new vertex will be created. The same is true if the end vertex of the edge is defined, so that either a preexisting can be selected or a new one is spawned. The edge is the spawned correctly connecting the start vertex with the end vertex.
When creating a face the same procedure is done as for the placement of an edge, but already existing edges can be included in the face by selecting both start vertex and end vertex. If no edge is between two vertices, regardless of whether they were newly spawned in ot not, a new one will be generated. In order to finish the face either the F-button must be pressed or the vertex selected first must be selected a second time, hence spawning a face. Not until the face is confirmed will it be part of the data structure. By pressing the right mouse button the selection is discarded and all newly spawned vertices and edges are destroyed.

4.2 Finding and Generating Cells

For the purpose of creating cells within the mesh a cell finding algorithm has been implemented (for further details refer to section 5.7). When selecting a face while being in the "CellMode" the algorithm will search for a valid cell. If one is found all the faces that are part of the cell are highlighted. By pressing the F-button (button to confirm the selection) the cell is placed, the representation is a 50% downscaled version of the actual cell, and also added to the OVM data structure.

4.3 Moving different Entities

In OVMVR it is also possible to modify the positions of vertices, edges, faces and even cells by moving them around. With the PlacementChaperone the component which should be moved can be selected and the position of the selected component is following the position of the PlacementChaperone until the new position is confirmed or the mode is cancelled, in which case the previous position of the component is restored. Based on the new position of the component adjacent, connected entities may also change there position and appearance. These directly affected entities as well as the selected component will be highlighted during the process and a preview is generated every frame.

4.4 Delete Entities

When deleting an entity of any kind, all other entities that are dependent on the existent of it, like an edge needs two vertices, will be deleted as well. The dependency graph would look as follows:
Vertex $\rightarrow$ Edge $\rightarrow$ Face $\rightarrow$ Cell

4.5 Save and Load OVM Files

It is possible in OVMVR to save the generated mesh as an OVM file in order to use it in OpenFlipper for example or using the BlenderInterrup tool creating a Blender file so that it can be viewed in the Blender application. Already predefined OVM files can be loaded into OVMVR so they can be viewed and modified. The current limitation is 5'000 entities because otherwise the loading can take a very long time or the application can crash if too many vertices and edges must be displayed. Additionally when

loading an OVM file the user is able to increase or decrease the bounding box by inserting the desired diagonal bounding box length into the input box next to the load button.

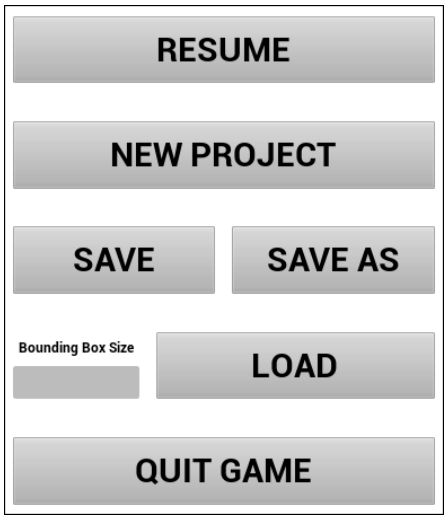


Figure 4.2: Pause Menu

If this input is not a number, lesser than or equal to zero, or empty the bounding box size will be set to a predefined value. This increase of the bounding box will also affect the later saved object, because the positions of the vertices themselves are altered.

Chapter 5

Implementation

5.1 Project Structure

OpenVolumeMeshVR is a C++ Unreal Engine project which means that additionally to the Unreal Engine's own blueprint approach to programming an application native C++ files are integrated into it. As can be seen in Figure 5.1 from the native C++ files an Unreal Engine Blueprint can be created. These Blueprints are then used to create the In-Game Instances which are part of the apllication's environment. The advantage of using this approach is that the developer can implement the much more code-intensive functionalities within the C++ file and then use the Blueprint Class to add further functionalities, like input handling or displaying the HUD and pause menu using UE's Visual Code. Furthermore, when wanting to spawn many instances of a certain class, like it is with the component representations (see Section 5.3), an Actor Blueprint based on the associated C++ class can be created and then can be easily spawned into the world.

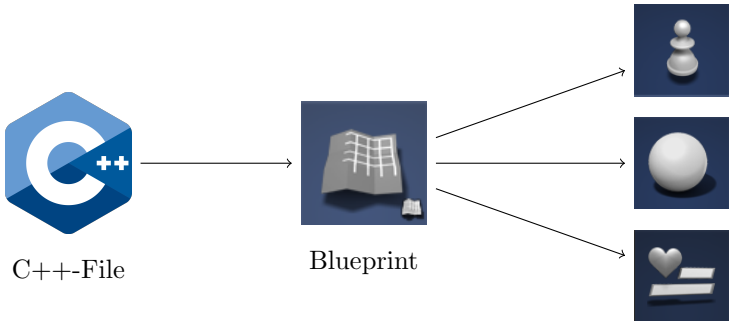


Figure 5.1: Blueprints in Unreal Engine 4

In order to make the OpenVolumeMesh source code available to the project an additional module had to be integrated. Modules in Unreal Engine are the building blocks from which an application or a game can be build. In Figure 5.2 one can see that OpenVolumeMeshVR in itself is already a module. The OpenVolumeMeshModule in which the integration of the OVM source code and also the CellFinding algorithm (Section 5.7) is implemented is independent from OVMVR and therefore can easily be outsourced to other projects if needed.

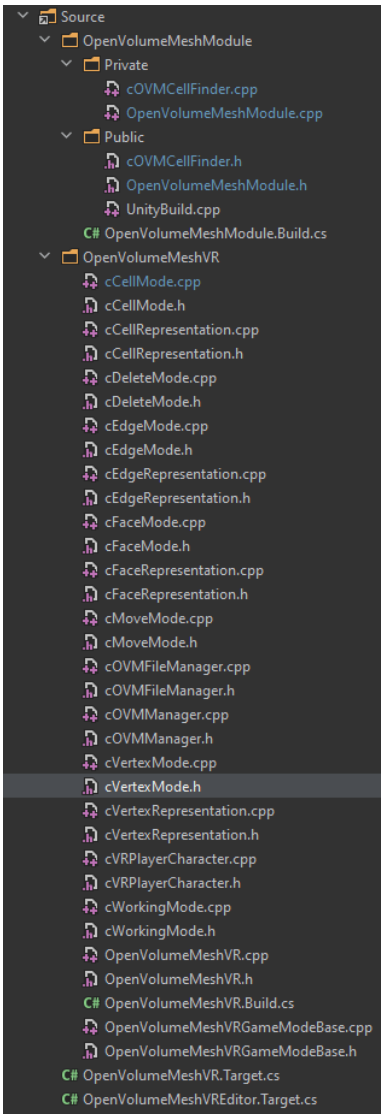


Figure 5.2: Project Structure

5.2 Functionality for both VR and Mouse-Keyboard Control

OpenVolumeMeshVR is able to support both Mouse-Keyboard Control and VR Control. The Mouse-Keyboard Control methods are defined within the AcVRPlayerCharacter class. The VR Control, however, is handled by the engine itself, by binding the position of the head-mounted device with the position of the camera. To make the controllers in VR visible, meshes provided by the Unreal Engine are loaded and displayed if a motion source, also known as the head-mounted device, is tracking. Otherwise these meshes will not be displayed and only the PlacementChaperone, replacement for the cursor, is shown. This behaviour is controlled within the VRCharacter Blueprint, a blueprint created from the cVRPlayerCharacter C++ File.

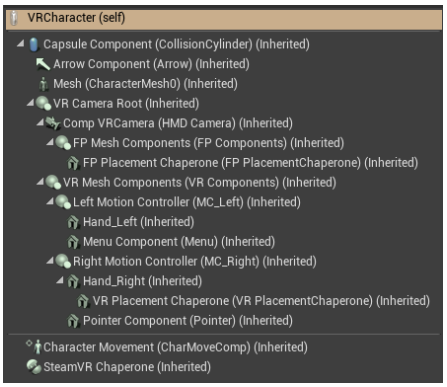


Figure 5.3: Contents of VRCharacter

As seen in Figure 5.3 It contains a component for VR components and one for Mouse-Keyboard Components. When the VR Character is initialised all of these components will be created. With the help of Unreal Engine's Visual Code, the state of the motion tracking can be queried and therefore the appropriate components can be shown (Figure 5.4).

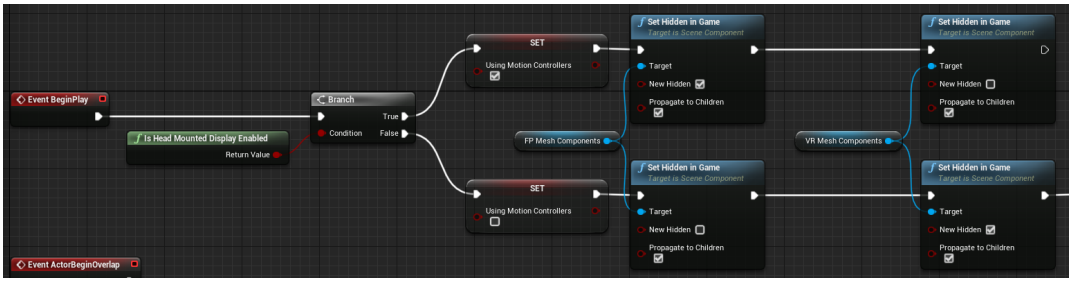


Figure 5.4: Visual Code for showing the appropriate components

5.3 Representation of Components

The Vertex and Edge Components, the visualization of both these entities, are predefined Assets, Actor Blueprints from C++ Files, within the Unreal Engine. They can be found in UE's content browser in the directory Content/Blueprint/BaseGame/Assets. This approach had the advantage that the functionality of each component, like handling the material or enabling the Tick function in order to move the asset, is integrated by using C++ code whereas the shape can be set in the Blueprint Editor, which is much easier than coding it from anew. Both the vertex and the edge representation are referenced in the VRCharacterBlueprint in which so called UProperties were added which are class variables that can be

set in the Blueprint Editor. This can be seen in Figure 5.5 in that the UProperties are defined within the code but actually are set within the Blueprint (red part) therefore not needing any ObjectFinder and can guarantee that the application always spawns the correct component representation. If these should want to be changed either the assets themselves can be altered or new references can be set in the VRCharacter blueprint.

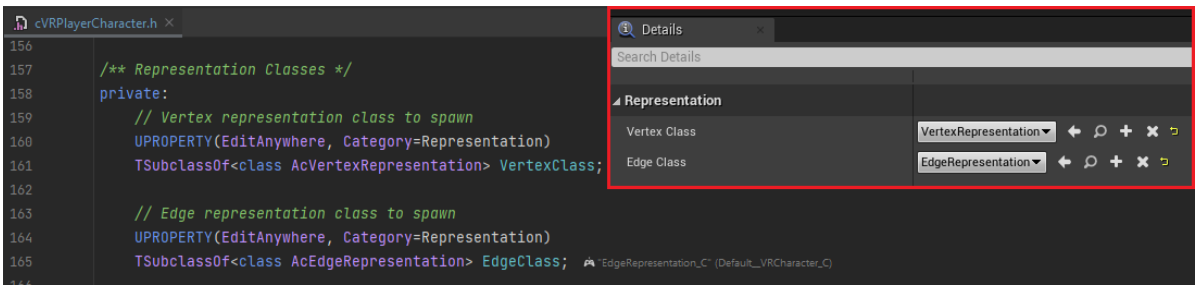


Figure 5.5: The creation and setting of UProperties

Because Faces and Cells are created differently - with the use of Unreal Engine's Custom Mesh (Section 5.4) - there is no blueprint for these assets.
Additionally different materials were defined such that the highlighting of objects, as well as the indication that the cell finding algorithm has terminated, can be performed. These materials are loaded in the respective constructor of each representation class:

```
static ConstructorHelpers::FObjectFinder<UMaterial>LoadedDefaultMaterial(TEXT("path"));
```

This line of code will search for the named material and save it in the LoadedDefaultMaterial struct. With LoadedDefaultMaterial.Succeeded it can be queried whether the loading was successful and with LoadedDefaultMaterial.Object the material instance can be accessed. A similar approach as seen before using UProperties could have been used for the materials as well, but because Faces and Cells do not have any Blueprint associated with them and therefore for these cases the second approach would inevitable, hence for the sake of consistency this approach was chosen for all components. In consequence when one wants to change the materials, one has to change the materials themselves or need to adjust the path within the code if new materials are imported and are chosen to be used.

5.4 Creation of CustomMesh to represent Faces and Cells

Unreal Engine has the option to generate a custom mesh during runtime which will be used as the representation for faces and cells in OpenVolumeMeshVR. A custom mesh is a geometric entity which contains many mesh sections. A mesh section can be understood as a single face of which one can define its vertices and the triangles it is made from. In order to create this kind of entity, first a UProceduralMeshComponent must be instantiated. This class is predefined by the Unreal Engine developers which allows the user to create custom triangle mesh geometry. When creating a face or a cell within OpenVolumeMeshVR an array of vertices, or faces respectively, will be handed over to the function which is responsible for generating the mesh entity. Because a cell is just scaled down version of multiple faces, which are therefore corresponding to the individual mesh sections, only the approach for generating a "face mesh" will be clarified. In preperation to create a mesh section first all vertex position relative to the center of the actual face are added to an array. Then a triangulation is performed in order to create several triangles which will make up the whole face. The current method used in OpenVolumeMeshVR is to create another vertex in the barycenter of the face which does then combine with two adjacent vertices in order to create a triangle. These triangles are saved also in an array simlar to how OpenVolumeMesh does, by referring the index of the vertex in the position array mentioned before. By calling the CreateMeshSection method and handing over the position and triangles array a mesh section is generated by the Unreal Engine.

5.5 Integration and Use of OpenVolumeMesh

In order to be able to run and build the programm on many different platforms also the OpenVolumeMesh framework must be built previously in order to make the library accessible to the algorithms within the application. In Unreal Engine native C++ libraries can be implemented by the use of modules. A module is a building block in unreal engine which contains its own Build file. Within this module the required files of the OVM source are included in a UnityBuild file, hence building the OpenVolumeMesh library during the build of the actual application.
This approach has the advantage, unlike by directly linking the OVM library to the project, that the application itself can build on any platform without any need of first building OpenVolumeMesh and searching for the correct library because this does depend on the platform it is used. Also when distributing the project's source code via GitLab the project can be opened within the Unreal Editor and instantly be compiled and used.
If a new entity is spawned into the virtual world a corresponding EntityHandle is added to the PolyhedralMesh data structure. Additionally a OVM-Property is added to this Handle in order to create a direct difference between it and the actual representation component that was spawned.

5.6 Algorithm for Moving Components

5.7 Algorithm for Finding Cells

To let the algorithm find a cell the user has to select a "halfface". Because in OVMVR halffaces are not a single component that is shown, but only the face on its own is present, the user is placing the PlacementChaperone on the preferred side of a face, so the direction to which the cell should be created is known to the algorithm, simulating the behaviour if a certain halfface would have been selected. Therefore when there isn't a cell on the preferred side of the face but on the other one, the algorithm will terminate by finding the other cell. But because it would have fully searched for the one that was preferred this behaviour can be disregarded, because the user still has the control over whether the cell should be created or not. The algorithm will either terminate by highlighting a found cell containing the selected face in red or won't mark anything at all and indicates that it has not found anything.
To visualize the following steps, the following mesh is considered (most of the highlighting is not shown in the editor at runtime; only the result is marked in red):

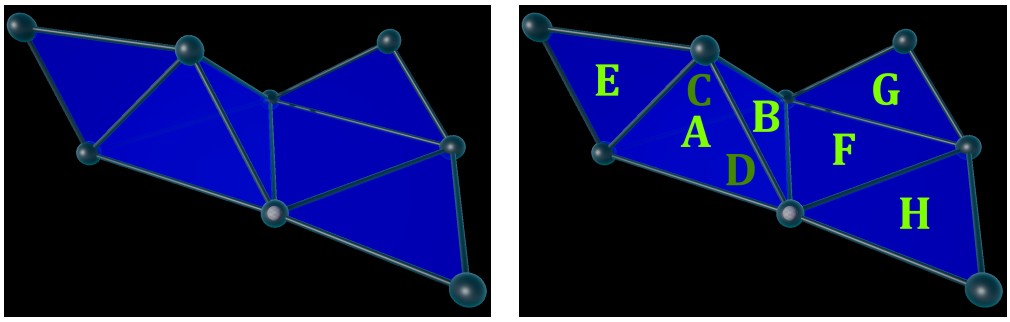


Figure 5.6: Initial Mesh

The user wants to create a cell by selecting face **A**, which is then passed as a parameter to the *CalculateCell function*:

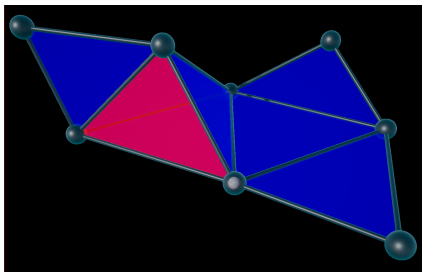


Figure 5.7: Selected Face

First this selected face is added to the *CheckedArray* to keep track of the already processed faces to save time and computational power. Then this face is checked whether it is "fully enclosed" in the following manner:

```
for each edge boundary of currentFace :
        if edge is boundary of at least two faces :
                add currentFace to FaceArray
                check faces adjacent to currentFace ...
        otherwise :
                return false
```

Because in this example the selected face is "fully enclosed" it is added to the *FaceArray* - which in the end will hold all the faces of the valid cell (none if no cell was found). Then all the adjacent faces of the selected one are checked recursively one after another whether they will lead to a valid cell or not. The adjacent faces are the following:

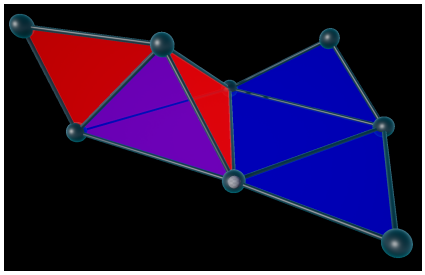
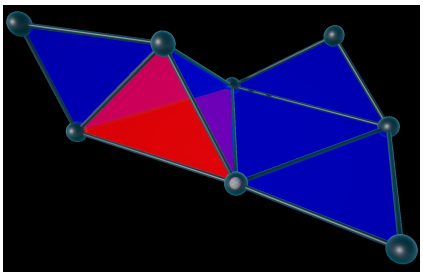


Figure 5.8: Adjacent faces of the selected face

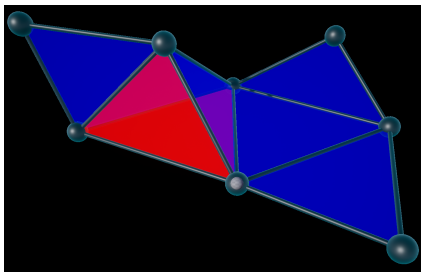
These faces then are sorted by prority:

1. face which is adjacent to the most faces which are already contained within the *FaceArray* (synonym with the future cell)
2. sharpest angle between the face and the previous checked face (also considering which side of the initially selected face was preferred)

Therefore the first face that is checked is one that is contained within the terahedron, because the angles to the initial face are sharper than the angle to the face to the left (**E**); for this example the bottom face of the tetrahedron (**D**) is the first one in the array after sorting. This face is then passed as a parameter to the *CalculateCell function* and is added again to both *FaceArray* and *CheckedArray*, because it is also "fully enclosed".



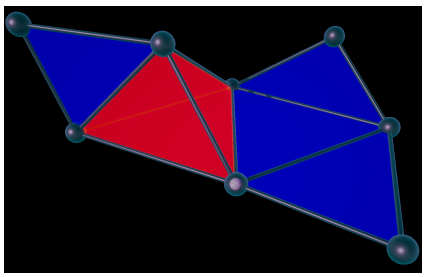
(a) Content of FaceArray



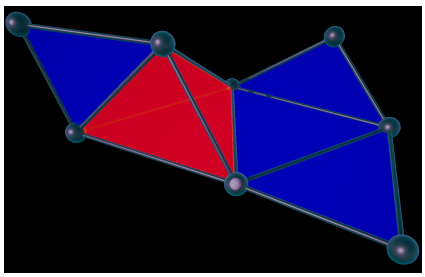
(b) Content of CheckedArray

Figure 5.9: Contents of both arrays

Again the adjacent faces are fetched and sorted. Because the front and bottom face are contained in the *FaceArray* the left and right side of the tetrahedron (**B** and **C**) have a higher priority to be checked. After one was checked and added to both arrays, the last face of the tetrahedron has a higher prority than all the other faces therefore after three steps the *FaceArray* and *CheckedArray* are containing the following faces:



(a) Content of FaceArray

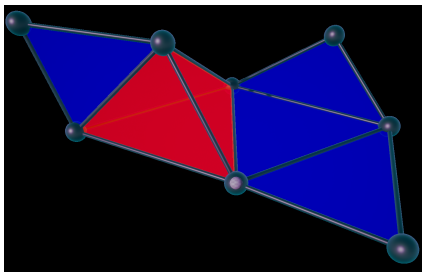


(b) Content of CheckedArray

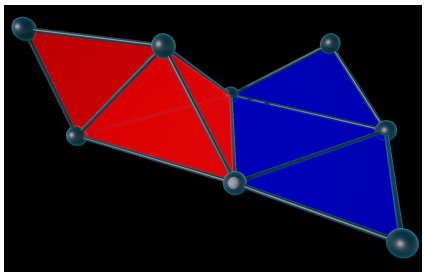
Figure 5.10: Contents of both arrays

The recursion algorithm is not over so if the last face that was checked was face **C** its adjacent faces will need to be checked as well. All the faces of the tetrahedron have the same priority but because they are already part of the *FaceArray* they are skipped. The face on the far left (face **E**) is therefore the only candidate available where the cell could continue further, but because it is not "fully enclosed" it is removed from the *FaceArray* and it will count as checked. Because all of the edges of face **C** have one additional face that can lead to a valid cell the face will not be removed from the *FaceArray* and the algorithm will return true for this particular face:

```
for each edge of face X:
        if all adjacent faces return false :
                return false
return true
```



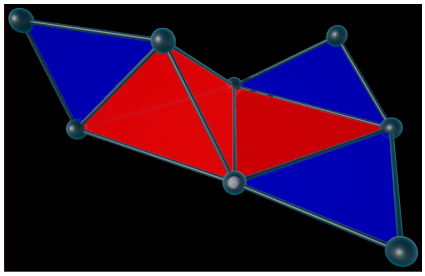
(a) Content of FaceArray



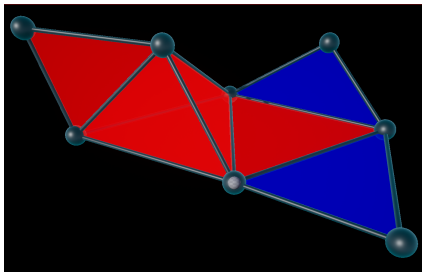
(b) Content of CheckedArray

Figure 5.11: Contents of both arrays

Next the other checks for the right side of the tetrahedron are made. Again the bottom and front part of the tetrahedron are skipped because they are still in the *FaceArray*, and the algorithm returned true for the left side of the tetrahedron. The last thing to check is whether the face adjacent to the right (face **F**) could be a continuation of the cell. Because it is "fully enclosed" it will be added to the *FaceArray*:



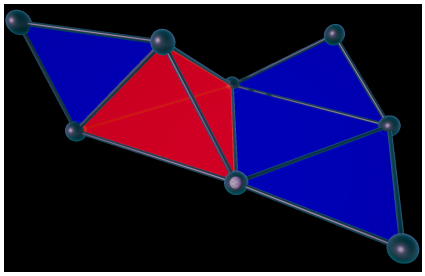
(a) Content of FaceArray



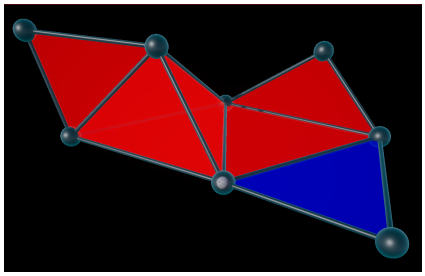
(b) Content of CheckedArray

Figure 5.12: Contents of both arrays

Its adjacent faces are then checked. The faces that are part of the tetrahedron are again skipped. Because faces **G** and **H** are not "fully enclosed" their checks will return false. Because the face does not fulfill the criteria that each of its edges have exactly 2 faces that fulfill the "fully enclosed" criteria it is removed from the *FaceArray*. Because one side already returns with a false the other side does not need to be checked. Therefore the check for the enclosed face on the right will return false, but because the right side of the tetrahedron (face **B**) fulfills the criteria that there are exactly two "fully enclosed" faces on each of its edges, it will not be removed and the algorithm returns true for it. The same is happening for the bottom and the front face of the tetrahedron. The algorithm terminates with the four faces of the tetrahedron selected as a valid target to create a cell.



(a) Content of FaceArray



(b) Content of CheckedArray

Figure 5.13: Contents of both arrays after the termination of the algorithm

5.8 Implementing additional Modes

A mode represents the implemented class of a given feature. As concluded in Section 3.3.3 for each feature that should be of use a different mode should be implemented, i.e. there are four different modes for placing the four different entities vertex, edge, face, and cell. A mode like the edge or the face mode is considered to have a continuous mode, because after the first vertex being placed the behaviour of the code must be changed in order to create an edge or even a whole face.
When implementing a mode it must be of type **AcWorkingMode**. AcWorkingMode is an abstract Actor, which behaves like an interface, because Unreal Engine does not give a developer the option to use an actual interface for this usage. The code within the header class of the new file (suppose the new mode is called NewMode) will look as the following:

```
// cNewMode.h file

#include "cWorkingMode.h"

class OPENVOLUMEMESHVR_API AcNewMode : public AcWorkingMode
{
    GENERATED_BODY()

    // Declaration of variables and methods
}
```

The following 4 methods are necessary for each mode and must therefore be overriden (more information can be get from the documentation of the code):

```
// What happens if the left-mouse button/right B-trigger is clicked
virtual void AcceptSelection(class AcVRPlayerCharacter * cActor, ...) override;

// If the WorkingMode contains a continuous mode this method will handle the
// "Undo" function if the mode is terminated or switched while being actively used
// (i.e. deleting all newly spawned entities)
virtual void ForceUndo(class AcVRPlayerCharacter* cActor) override;

// Returns true if the mode is in the continuous working mode (which will also
// indicated at the top of the HUD) and false otherwise
virtual bool IsInContinuousWorkingMode() override;

// The name of the mode that should be displayed in the HUD
virtual FString ToString() override;
```

In order to also be accessible during the game the *VRPlayerCharacter* must include this new mode and the new mode must be instantiated when the corresponding number key is pressed. In the cVRPlayerCharacter.cpp there are already predefined methods called ChangeModeX, where X is the number corresponding to the button pressed. The code within this method must be changed accordingly:

```
void AcVRPlayerCharacter::ChangeModeX()
{
    ResetMode();
    CurrentMode = NewObject<AcNewMode>();
}
```

Furthermore in the blueprint editor of Unreal Engine for HUD display which is shown during the game a fitting image should be placed on the corresponding canvas to ensure usability and information giving for the actual user.

Chapter 6

Conclusion

6.1 Comparison: Set Goal - Actual Outcome

Chapter 7

Future Work

OpenVolumeMeshVR is a very simple application for generating and modifying a mesh. During the development of it a few different ideas popped up which can be implemented in a future version of it.

Snapping Mode Extention
Add the possibility to snap certain entities to other ones, like a vertex in the middle of an edge or face, or implement a degree snapping for generating edges or faces. Additionally length snapping could also be an option.

Coloring of Entities
By coloring single vertices, edges, faces or even cells certain attributes of the mesh can be highlighted. A color gradient along edges and faces can also be implemented if the adjacent vertices are differing in color.

Quality Metric of OVM
OpenVolumeMesh provides many different quality metrics for example for cells which can be implemented by for example highlighting the particular part of the mesh.

Placing cameras for different perspectives
In Unreal Engine it is possible to spawn cameras which will open in a new window in order to be able to view the mesh from different perspectives at the same time. This will greatly enhance the possibilities for showcases of a mesh.

Multiplayer
When in the Unreal Engine Editor itself it is possible to use OpenVolumeMeshVR in a multiplayer environment. This possibilty is not given if the application is build and also within the editor it is not working reasonably good. It may be a great experience to work with others and generating a mesh together.

Increase the Limit of Entities
As of now the soft limit cap of entities of OpenVolumeMeshVR is approximately 5'000. By using culling or change the render distance within the UnrealEngine or by altering the representations, for example via a level of detail (LOD) approach, this limit could be greatly increased.

Implement Undo and Redo function
While a user is creating a mesh errors can happen. Therefore it might be a good idea to have an undo and redo function within the application to quickly correct these errors.

New (better) Triangulation Algorithm for the Generation of Mesh Sections
For the creation of mesh sections one must first define the triangles from which it is made of. Currently an additional vertex is placed in the middle of the face and all the other vertices next to each other are connected in sequence with it in order to create the triangles. For better results this triangulation algorithm can be changed and adjusted.

Literature

[Boa12] Boas, Yuri Antonio Gonçalves V.: Overview of Virtual Reality Technologies. (2012)

[GB4] *Google Blocks Documentation and Features.* `https://vr.google.com/blocks/`. – Accessed: 15.03.2019

[GTB] *Google Tilt Brush Documentation and Features.* `https://vr.google.com/blocks/`. – Accessed: 31.03.2019

[Hei61] Heilig, Morton L.: *Sensorama simulator patent - US3050870.* 1961

[KBK13] Kremer, M. ; Bommes, D. ; Kobbelt, L.: OpenVolumeMesh - A Versatile Index-Based Data Structure for 3D Polytopal Complexes. (2013)

[MSM] *Microsoft Maquette Documentation and Features.* `https://www.maquette.ms/`. – Accessed: 15.03.2019

[MVR] *Masterpiece VR Documentation and Features.* `https://www.masterpiecevr.com/`. – Accessed: 05.04.2019

[OVM] *OpenVolumeMesh Documentation.* `https://www.openvolumemesh.org/`. – Accessed: 24.04.2019

[SKB15] Seiler, Uwe T. ; Koch, Volker ; Both, Petra von: Immersive Virtual Simulation of Spaces. (2015)

[Soc] Society, Virtual R.: *History Of Virtual Reality.* `https://www.vrs.org.uk/virtual-reality/history.html`

[Sut65] Sutherland, Ivan: The Ultimate Display. (1965)

[Tri] Trio: *Virtual Reality Applications: 10 Industries Using Virtual Reality.* `https://trio.dev/blog/virtual-reality-applications`

[UE4] *Unreal Engine Documentation.* `https://www.unrealengine.com/en-US/`. – Accessed: 03.05.2019

Appendix A

Intuition Study: Evaluation Sheet

Fragebogen (1 = sehr schlecht, 6 = sehr gut)

1. Bewegung
   a. Wie intuitiv waren die Bewegungsformen für Sie?
      i. Google Blocks ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6
      ii. Microsoft Maquette ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6

   b. Wie gut gefiel Ihnen diese Bewegungsform?
      i. Google Blocks ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6
      ii. Microsoft Maquette ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6

2. Objektinteraktion
   a. Welches Interface (Ansicht der Controller, Auswahlpalette, etc.) gefiel Ihnen besser?
      ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
   b. Welche Art der Interaktion war für Sie einfacher zu verstehen und zu benutzen?
      i. Platzieren von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
      ii. Rotationssnapping von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
      iii. Löschen von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
      iv. Färben von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
      v. Modifizieren und Verändern von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
      vi. Kopieren von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
      vii. Gruppieren von Objekten
         ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
   c. Welches Programm gefiel Ihnen im Allgemeinen besser?
      ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich
   d. Welches Programm war einfacher zu handhaben?
      ☐ Google Blocks ☐ Microsoft Maquette ☐ beide gleich

3. Allgemeines
   a. Haben Sie schon einmal ein VR-System verwendet?
      ☐ Ja ☐ Nein
   b. Haben Sie schon einmal ein gesten-/bewegungsgesteuertes System verwendet (Nintendo Wii, Playstation Move, XBOX Kinect, VR etc.)?
      ☐ Ja ☐ Nein
   c. Haben Sie schon einmal 3D-Objekte (Meshes) am Computer erstellt?
      ☐ Ja ☐ Nein