Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Creating sketches of virtual worlds for interactive applications using VR

Bc. Ondřej Perný

Supervisor: Ing. Ondřej Slabý
Field of study: Open Informatics
Subfield: Computer Graphics
January 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Perný Ondřej**	Personal ID number:	**470171**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Graphics and Interaction**		
Study program:	**Open Informatics**		
Specialisation:	**Computer Graphics**		

II. Master’s thesis details

Master’s thesis title in English:

Creating sketches of virtual worlds for interactive applications using VR

Master’s thesis title in Czech:

Tvorba náčrtů virtuálních světů ve VR pro použití ve tvorbě interaktivních aplikací

Guidelines:

Research existing applications dealing with the creation of simplified 3D models and current methods of user interface design for the purpose of 3D modelling in VR, especially with the aim of creating virtual worlds for interactive computer applications. Based on this research, identify key properties, advantages and disadvantages of designing virtual worlds using virtual reality as opposed to traditional 2D interfaces. Design and implement a software tool mainly in C++ using Unreal Engine and its Blueprint visual scripting system, which will make use of such key properties. The tool should allow its user to easily sketch a world or its part and provide the ability to naturally navigate the resulting model. The tool should also be able to work with already existing models and export models in a format which is compatible with popular game engines, such as Unreal Engine or Unity. Consider the possibility of integrating the tool directly into the Unreal Engine editor in such a way that makes it possible to extend the resulting integration to editors of other game engines. Test the resulting application in terms of user friendliness and the degree to which the process of creation of virtual worlds changes with real users.

Bibliography / sources:

Beever, L., Pop, S. W. & John, N, W. (2020). LevelEd VR: A virtual reality level editor and workflow for virtual reality level design. 2020 IEEE Conference on Games (24-27th August). 10.1109/CoG47356.2020.9231769
Conesa-Pastor, J.; Contero, M. EVM: An Educational Virtual Reality Modeling Tool; Evaluation Study with Freshman Engineering Students. Appl. Sci. 2022, 12, 390. https://doi.org/10.3390/app12010390

Name and workplace of master’s thesis supervisor:

Ing. Ondřej Slabý Department of Computer Graphics and Interaction FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **01.02.2022** Deadline for master's thesis submission: ___________

Assignment valid until: **30.09.2023**

Ing. Ondřej Slabý Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to express my gratitude to my supervisor Ing. Ondřej Slabý for his continuous support, guidance, and patience during this work. Furthermore, I would like to thank all the people close to me for providing their support and encouragement.

Declaration

I declare that I have worked on this thesis independently and that I have listed all used sources in the bibliography.

In Prague 8. January 2023

Abstract

This thesis aims to research the potential of creating level blockout/sketching virtual worlds for interactive applications using virtual reality. It identifies key properties, features, advantages, and disadvantages of the virtual reality interface as compared to traditional 2D programs. The core of this work is the presented design of an ideal application for this purpose, based on research and tests of similar existing applications. The design is then used to develop a prototype application, called BlockoutVR, using Unreal Engine 5 to test the viability of this process and compare it to standard 2D interfaces. The possibility of integrating the application into game engines through plugins is discussed.

Keywords: virtual reality, virtual world sketching, blockout, Unreal Engine 5

Supervisor: Ing. Ondřej Slabý

Abstrakt

Tato práce se zabývá možnostmi tvorby náčrtů virtuálních světů pro interaktivní aplikace pomocí virtuální reality. Identifikuje klíčové vlastnosti, funkce, výhody a nevýhody rozhraní virtuální reality ve srovnání s standardním 2D rozhraním. Jádrem této práce je navržení ideální aplikace pro tento účel, založené na výzkumu a testování podobných existujících aplikací. Tento návrh je poté použit k vytvoření prototypu aplikace nazvané BlockoutVR. Prototyp byl vyvinut pomocí Unreal Engine 5 a slouží k otestování použitelnosti tohoto návrhu a jeho srovnání s desktopovými programy. Probírá se také možnost integrace aplikace do herních enginů pomocí pluginů.

Klíčová slova: virtualní realita, náčrt virtuálního světa, blockout, Unreal Engine 5

Překlad názvu: Tvorba náčrtů virtuálních světů ve VR pro použití ve tvorbě interaktivních aplikací

Contents

Figures

Tables

Chapter 1
Introduction

1.1 Motivation

Virtual reality (VR) is becoming an important field in the entertainment and professional industries because it provides a better interface for presenting 3D space than regular 2D screens. However, VR is not usually used in the creation of VR application content. This work aims to understand why this is the case and whether VR can effectively contribute as a tool for creating virtual spaces, world sketches, and level design.

The created application will focus on level layout/blockout rather than complete level design, a more general problem. It is not a generic world-building application, as that would imply a full set of tools for creating complete worlds and levels. I suspect that the main reason VR is underutilized in the creative process is that accessible VR programs are too general in what they do, failing to provide a productive workflow for specific tasks. For that reason, the proposed application focuses strongly on only one aspect: level layout, also known as level blockout, which is defined as a rough draft level built with simple-shaped objects (primitives) but without any details or polished art assets [50]. The application proposed in this work will be referred to as BlockoutVR. The result can be exported as a 3D model for further work in any model processing software (e.g., Blender, Maya) or directly to a game engine (e.g., Unreal, Unity).

1.2 Content

This thesis has three main parts. The first chapter is an analysis of all relevant sources of information on world sketching and blockout in VR or related tasks, including academic research, guidelines provided by VR distributors, and similar existing applications. It covers essential topics such as general VR properties, comfort concerns in VR, comparison with 2D interfaces, VR user interfaces, and the features and technical solutions for problems related to world sketching.

The second part of the thesis is the design of an ideal application for blockout in VR using the knowledge obtained in the analysis chapter and

Figure 1.1: Common modern HMD with standard pair of controllers, Oculus Quest 2. [35]

personal experience from working on a similar application in the past. This concept is designed as an abstract template for implementation, defining key features, locomotion, settings, tools, controller scheme, VR control interface, user interface, and other features such as importing and exporting models. Important design decisions were based on specific findings during the analysis and are often referred to directly in the text. Essential features such as control schemes are described in detail, while some other settings are defined, but allow for some flexibility in implementation.

The last part of the thesis is about the implementation of a prototype application BlockoutVR, based on the design from the previous chapter. Essential features from the design were selected and implemented, allowing for testing and comparison with non-VR programs. The testing methodology is described at the end of the work in the testing chapter.

Chapter 2
Analysis

The analysis chapter focuses on previous research, similar applications, related work, and guidelines for VR or other relevant extended reality (XR) contexts. As mentioned in the previous chapter, the popularity of VR is relatively recent, which means that many aspects of it remain unexplored by the academic community. However, developers of virtual applications have empirically solved many problems, and their articles, presentations, and applications are valuable sources of information, particularly regarding the virtual user interface (UI), design, and architecture of VR software.

2.1 2D versus 3D, advantages and disadvantages

Virtual reality (VR) adds an extra dimension to regular 2D programs, which has several benefits for 3D modeling. One of the main advantages of VR is that it allows us to perceive depth and shapes more accurately because we don't have to project 3D space onto a 2D screen, thereby preserving all spatial information. This means we can work directly with 3D objects in their correct space, scale, and shape, without losing any information due to projection. In addition, VR allows us to use a 3D cursor for input, so we can move in all three axes without changing the view or axis guidelines.

The downsides are more limited interface and some input options compared to regular 2D programs. Head-mounted display (HDM) occludes our full view, therefore full attention is in VR, which can be mentally taxing and hinder multitasking (with anything outside VR). Additionally, using VR controllers or hands for input requires a lot of physical movement, which can lead to fatigue, especially compared to using a mouse and keyboard or a tablet. According to a study [30], this can be a problem in VR, but it does not affect 2D workflow as significantly.

2.2 General VR guidelines

While VR applications offer new possibilities compared to regular desktop programs, they also come with their own set of challenges and limitations. One well-known effect called virtual reality sickness is a common phenomenon associated with using VR. With symptoms similar to motion sickness [29], including general discomfort, headache, stomach awareness, nausea, vomiting, pallor, sweating, fatigue, drowsiness, disorientation, and apathy [26], virtual reality sickness can affect people to varying degrees based on factors such as age, postural stability, flicker fusion frequency threshold, ethnicity, gender [26]. Experience with VR systems can build up resistance. When creating a VR app, it is important to keep these issues in mind and try to prevent them whenever possible.

Since this work is focused on people with limited or no VR experience, it is important to include these VR-specific problems, even if some of them are non-technical in nature. These issues will also play important roles in the Design chapter.

2.2.1 Preventing discomfort

Virtual reality provider Meta Quest (previously known as Oculus), have its own guidelines for designing VR apps with important parts devoted to preventing discomfort. Their *Meta Quest Developer Center* [36] is a valuable resource for basic guidelines to avoid potential pitfalls and unnecessary discomfort for users.

Each user should set their own session duration respecting both mental drain (mismatch between virtual and real inputs is straining) and physical burden (VR often requires standing and moving body). An Application should incorporate ways to encourage users to take necessary breaks without disrupting their workflow [36].

The typical cause of discomfort is a sensory mismatch and discontinuities with real-world experience [34]. Modern VR hardware does a great job minimizing this problem with minuscule video lag, responsive input, and smart techniques such as rendering focus based on eye movement (Eye Tracked Foveated Rendering [42]). Then there are factors dependent on a software developer. *Meta Quest Developer Center* [36] describes those comfort risks and usability issues.

Comfort Risks

- **Vection** - Illusory perception of self-motion, e.g. when the VR character moves, but the user stands still.
- **Vestibular Sense** - Can be described as a sense of balance, and discomfort can be induced by rotating or tilting.
- **Visual-Vestibular Mismatches and Comfort** - Dissent between vision and vestibular system (e.g. caused by seasickness).
- **Proprioception** - Awareness of the movement and position of the body. The VR character should mimic real-life positions.

- **Disorientation** - Losing track of the position in the the virtual environment. May occur due to movement factors like teleporting or snap turns.

Usability Issues

- **Space Limitations** - The size of the area covered by these systems ranges from small stationary spaces to location-based virtual reality (LBVR) systems, which can be of arbitrary size.
- **Fatigue** - Working in VR can be physically demanding, for example, VR games may require users to hold their hands in stretched-out positions for long periods of time or make excessive swings during combat. To minimize strain, interface and interactive elements should be within easy reach and not require complicated body positions, if possible.
- **Accessibility** - Certain physical needs may create barriers for some users, such as the need for prolonged standing sessions.

All the mentioned issues have solutions that will be addressed in a subsequent chapter in order to mitigate or solve them.

2.2.2 Vision

Vergence is a process of moving both eyes to track single binocular vision. Just like in real life, if an object is approaching closer to us, the eyes must conform to the vergence demand to maintain proper binocular vision. If an object is too close to the eyes, this can become very straining. In VR, nothing should be rendered less than half a meter from the headset camera, ideally over a meter. Unlike in real life, there is generally nothing stopping the user from putting their head directly in an object in the VR scene, so it is important to have a mechanism to accommodate this situation if it occurs.

Improper depth representation of objects can break VR immersion. To prevent conflicting depth signals (such as a visible controller even if it is behind another object), it is important to respect the depth of objects in the view during rendering. This is one of the reasons why it is generally best to avoid any sort of head-up display (HUD) and occluding the view [38].

In addition to stereopsis (the perception of depth based on the disparity between the viewpoints of each eye), there are other monocular (for one eye only) depth cues that can be used to improve depth perception, as described in [38].

Monocular depth cues

- **Motion parallax** - Close objects move faster.
- **Curvilinear perspective** - Lines converging in distance.
- **Relative scale** - Objects are smaller further in distance.

- **Occlusion** - Closer objects occlude distant ones.
- **Aerial perspective** - Distant objects faint (due to the atmosphere refraction).
- **Texture gradients** - Texture patterns diminishing in distance.
- **Lighting** - Shadows and highlights.

2.2.3 Locomotion

The intensity of locomotion should be aligned with the user's VR experience. Avoid fast acceleration for less experienced users, limit the direction of movement, and give the user control over movement rather than forcing them to move in a certain way to avoid simulator sickness [28].

2.2.4 User Input

To avoid sensory disruption, controllers (and other tracking devices) should match the relative position between real life and VR. In addition, controllers should use familiar button mapping, and applications should be accessible to both left and right-handed users [28].

2.2.5 Rendering

To ensure a comfortable VR experience, use a font that is easily readable in UI and scene elements, and choose a text size that is appropriate for comfortable reading. Consider using a signed distance field to keep the text size consistent. Avoid creating the flickering, high-contrast, or flashing elements, as these can be disruptive to the user. Use geometric models or parallax mapping instead of normal mapping, as it doesn't account for binocular disparity or motion parallax, making flatness more noticeable.

HMDs may suffer from lens distortion, so it is important to use proper distortion correction through SDK post-processing for the target device. Responsiveness is critical for VR immersion, so aim to keep latency consistently under 20ms [31] for a compelling VR experience. Finally, for a smooth feeling, aim for the highest frames-per-second that the HMD allows, if possible.

2.2.6 User Orientation and Positional Tracking

The user should be allowed to set their own origin point. Positional tracking should not be disabled or modified. Use room-scale (a defined physical space where the user can safely move) to stay within the boundaries of the free space, in order to prevent breaking immersion and keep the user safe from physical obstacles. The virtual camera allows the user to move into unusual positions that would be impossible in real life. A head-object intersection can cause discomfort, so it should be prevented by not allowing the user to get too close to solid objects [32].

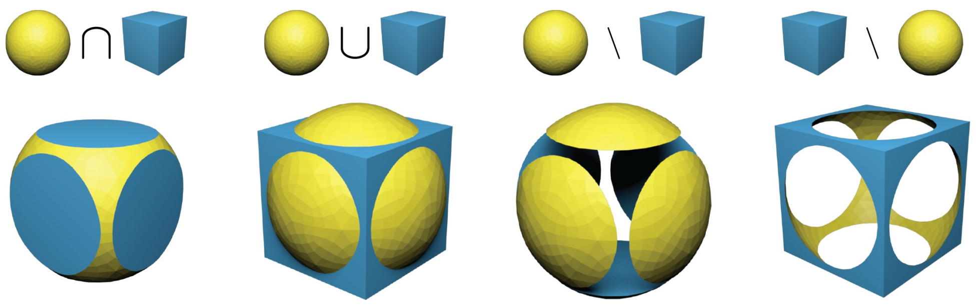


Figure 2.1: Sample of basic 3D boolean operations. [43]

2.2.7 Controls and interaction

Use haptic feedback for additional clarity. Add ray casts to simplify navigation and selection, they can as well assist users with impaired vision. Visually represent controller inputs with button highlights. Optionally, offer the option to personalize controller configurations. The complexity of a controller scheme should be minimized. Likewise, amount of buttons necessary for each action should be minimized [33].

2.3 3D modeling in VR

This thesis includes research on existing applications for creating simplified 3D models. In the context of 3D model creation, this work discusses two different approaches: a direct geometry modeling tool for basic objects and a world composition tool for creating virtual spaces.

2.3.1 Geometric modeling

Geometric modeling refers to the creation and modification of 3D models. VR in geometric modeling allows users to interact with and manipulate objects directly in 3D space, providing a more intuitive and immersive experience compared to traditional modeling techniques. The goal of this work is a world sketching application, so not many modeling tools are necessary. There is not expected to be a full modeling toolset, but only a few functions to support level creation by modifying existing objects in a scene. Useful types for this use include solid modeling with Boolean operations and wireframe modeling for simple changes to an object's geometry, both in the context of object modification rather than constructing new objects.

Boolean operations

Boolean operations for 3D models (see Figure 2.1), such as union and intersection, can be used to combine multiple objects into a single one. Or to remove unwanted parts of an object to create the desired shape, by Boolean subtraction. For example, a level designer can use a union to combine multiple

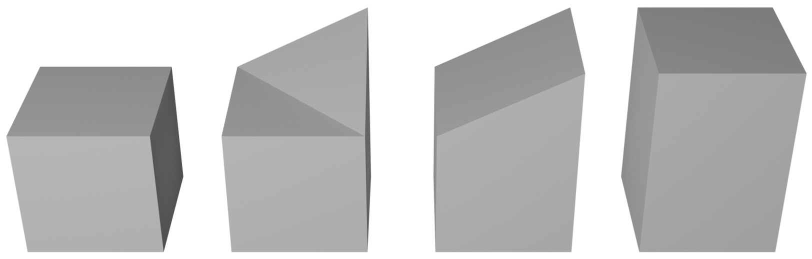


Figure 2.2: The cube on the left is the default cube, with the other cubes showing the effects of moving a vertex, edge, and face, respectively.

walls to create a room, or use an intersection to make a hole for a window in a wall. Other operators can also be implemented. These operations are the extension of the standard boolean operations to 3D space. The result is similar to logic operations but applied to models rather than logical meaning.

Level blockout typically consists mainly of primitives. Therefore, it makes sense to use Boolean operators, as there are many simple solid objects in the scene.

Boundary modifications

This operation is meant for simple geometry shifts (see Figure 2.2). Moving the face, edge, or vertex of a model in a scene. This means the topology of the model vertices stays the same, but the shape can change. One or multiple faces/edges/vertices can be selected, and by dragging them, changing their relative position to the rest of the model. The designer can quickly achieve expected shapes without necessarily combining multiple objects.

Complex VR modeling

One of the papers researching the modeling in VR is "EVM: An Educational Virtual Reality Modeling Tool; Evaluation Study with Freshman Engineering Students" by Conesa-Pastor and Contero [8]. It focuses on single objects rather than scenes or levels but provides useful insight into the vertex geometry tools in VR. Presents EVM (Educational Virtual Modeling) VR tool for modeling engineering-related objects. It describes a robust toolset for creating complete models (mostly engineering parts) from scratch. It works on the basis of curves and triangulated faces between them, specifically: "a curve is given as a set of consecutive lines, whereas a surface is defined by a contour formed by lines and curves and a set of mesh objects responsible for filling and shading its interior" [8].

One of the most interesting parts of the paper for this thesis is the user study. The engineering students used this application to rate their VR experience, compared to similar work in a standard 2D environment (SketchUp). In terms of the time needed to create a given model, both EVM and SketchUp performed similarly. However, from an experience perspective, the paper describes that participants in their comparison between EVM as a VR tool and SketchUp seemed more enthusiastic and motivated to work in VR as opposed to the 2D experience.

Possibly, this could be due to participants' inexperience with VR. However, it could also indicate a positive trend of preferring VR due to its immersive and overall better experience over traditional 2D interfaces. It should be noted that the paper mentions some problems with VR modeling tools, such as precision issues and missing functionalities for technical objects. However, since this work does not aim to provide a full modeling tool, these issues can be overlooked.

2.3.2 Designing virtual worlds

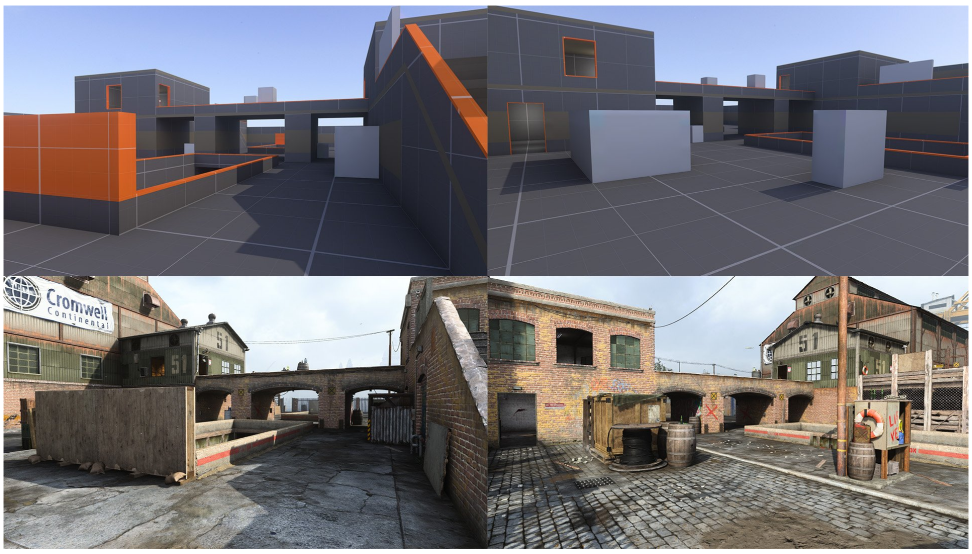


Figure 2.3: Blockout from Modern Warfare(2019) level design by Brian Baker [4]. (blockout of the level on the top images and corresponding finalized level below)

The main focus of this thesis and the follow-up BlockoutVR application is the design of virtual worlds through the process of composing simple objects (primitives) in space and defining their functions. These objects are usually basic 3D shapes without any visual details and serve as placeholders for the final, detailed objects during the level design and blockout phase. Examples of the blockouts and final scenes are depicted in Figures 2.3 and 2.4, where even simple primitives can clearly convey the overall appearance and function of the objects. Simple colors can also add distinct characteristics and specific functions to the models, such as grey for static objects and red for destructibles. This allows level designers to quickly communicate their intentions for the level and the interactivity of specific elements.

The research conducted by Beever, Pop, and John in "LevelEd VR: A virtual reality level editor and workflow for virtual reality level design" [5]

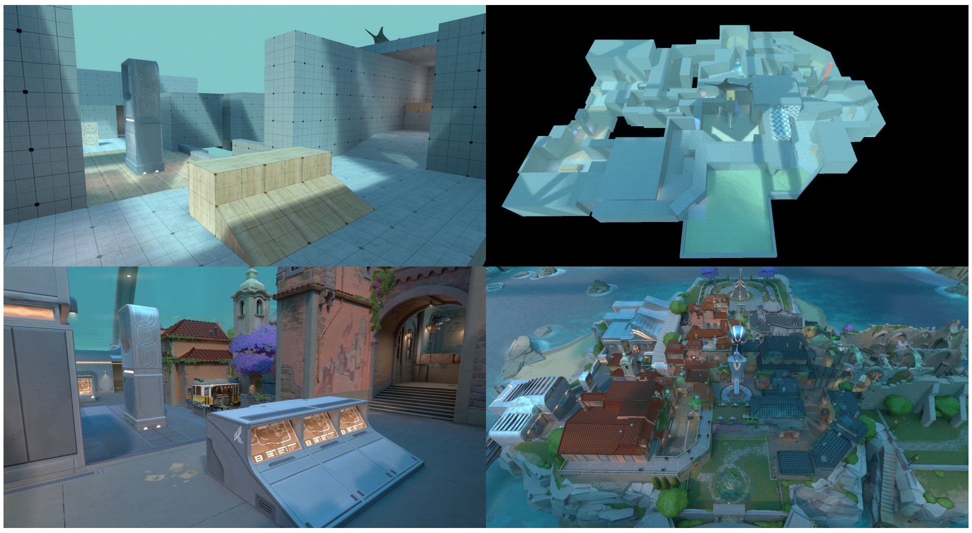
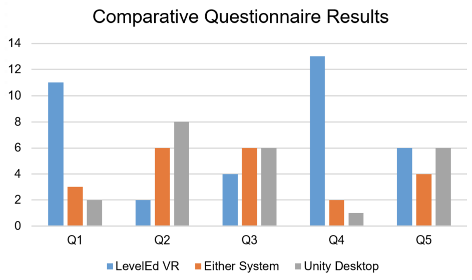


Figure 2.4: Blockout from Valorant level design by Pearl Hogbash [19]. (blockout of the level on the top images and corresponding finalized level below)



	Questions
Q1	When creating a level blockout for a virtual reality game I would prefer to use the following system
Q2	When creating a level blockout for any type of game I would prefer to use the following system
Q3	When adding scripted game play to a virtual reality game level I would prefer to use the following system
Q4	I found creating virtual reality game levels with a good sense of scale easier in the following system
Q5	I found my workflow was quicker/more efficient for developing virtual reality game levels in the following system

Figure 2.5: Questionnaire results for LevelEd vs Unity from [5].

covers similar topics, but with a focus on the overall level design process rather than just the blockout phase. Their study compared their VR application, LevelEd VR, to the Unity game engine through experiment testing.

Test subjects generally liked LevelEd for modeling space for use in VR, as they were able to perceive the scale of objects and the scene easily. However, when considering a universal application (not just VR), most participants voted for Unity, with the rest expressing no preference, as shown in Figure 2.5 along with other survey results. This is an interesting result, as it confirms that modeling VR programs can be effective in creating virtual space for VR use. The results are unclear on whether modeling in VR applications can bring significant advantages over 2D applications for use in 2D, such as desktop third-person view games.

Opinions on scripting the level logic inside LevelEd are fairly balanced. Compared to Unity, LevelEd VR's visual scripting may be more limiting, but it offers an interesting alternative. An older paper, "MakeVR: A 3D World-Building Interface" [21], describes another world-building tool that is simpler than LevelEd VR but provides a useful comparison in terms of toolset and user feedback. Another similar work, "Genesys: A Virtual Reality Scene

Builder" [9], proposes a scene builder with similar goals as the previously mentioned applications. There are a few more papers from the last ten years (2012–2022) that focus on world or scene-building, but they have very similar findings as the papers mentioned above, therefore will not be discussed here.

The previously mentioned "LevelEd VR" paper [5] has a slightly broader range of focus, including simple geometry modeling and in-app simple visual scripting. However, among other papers exploring similar topics (world design), this one is most closely related to this work with its strong focus on VR workflow, and therefore the knowledge gained there is central to this thesis.

While individual works differ in some ways, they usually agree on a certain set of core tools, including spawning, copying, moving, rotating, scaling, and deleting objects. User interfaces are usually relatively neglected, as they are seemingly considered a marginal necessity that doesn't receive enough care (in contrast to commercial VR programs). Another negative trait commonly found is impractical or counterintuitive control, as a lot of work was dedicated to the core functions but not as much to the workflow of their use.

An essential part of a world-building tool is a set of primitives, which are simple 3D models used to build the scene, such as cubes, spheres, or cones. However, there is no standardized set of primitives or consensus on which shapes to use. VR applications don't usually use custom imported primitives but rather a limited set of pre-prepared objects or rely on their own modeling system.

Previous research has solidified the general VR opinions about its advantages and disadvantages. While they set solid core design ideas, the final products (which are usually done as proof of concepts) suffer from various shortcomings. The design part of this thesis builds on the core ideas from these papers while trying to address the mentioned flaws.

2.4 User interface in VR

The "Visual Design Methods for Virtual Reality" [2] provides an excellent and rather complete overview of the virtual reality interface, backed by experience in the applications shown throughout the paper. It discusses where interface elements should be located, what form they should have, and what interactions they provide to the user. It also covers how specific choices, such as the color or location of buttons, can affect the user and their decision-making, but most importantly, how to provide the user with the most intuitive and pleasant experience.

Aside from this paper, the best sources for interactive VR interfaces are already existing applications. It doesn't even have to be a necessary VR application with the same goal, as general VR interface rules are shared for all VR uses. While the academic sphere has yet to explore this topic more, many software developers are already solving related problems and sharing their solutions. The website "The UX of VR" [14] gathers and provides all kinds of resources for everything related to the VR user experience (UX),

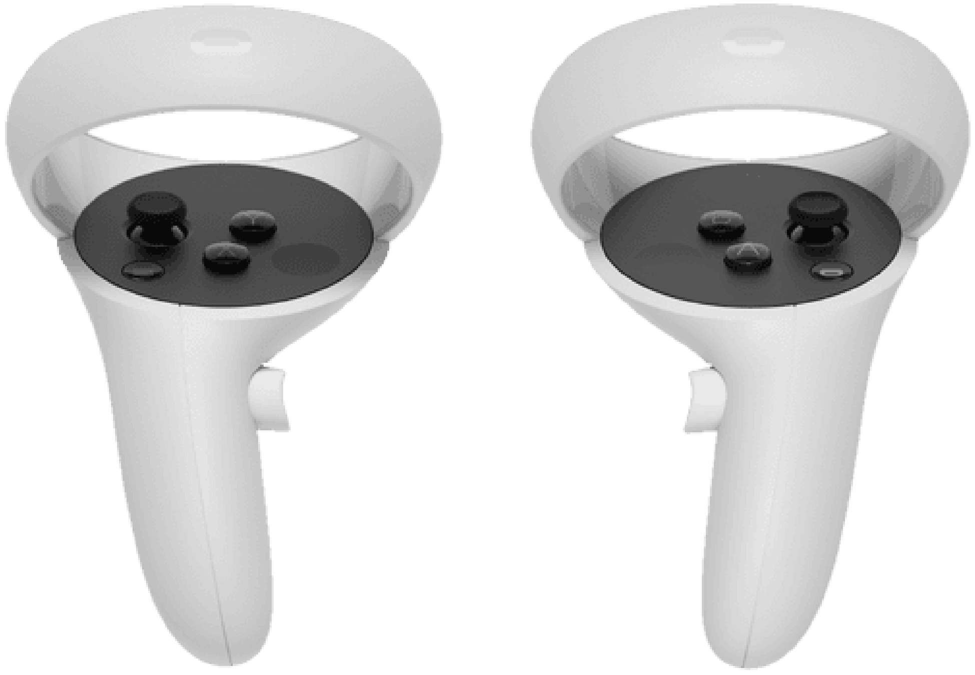


Figure 2.6: Meta Quest 2 controllers. [49]

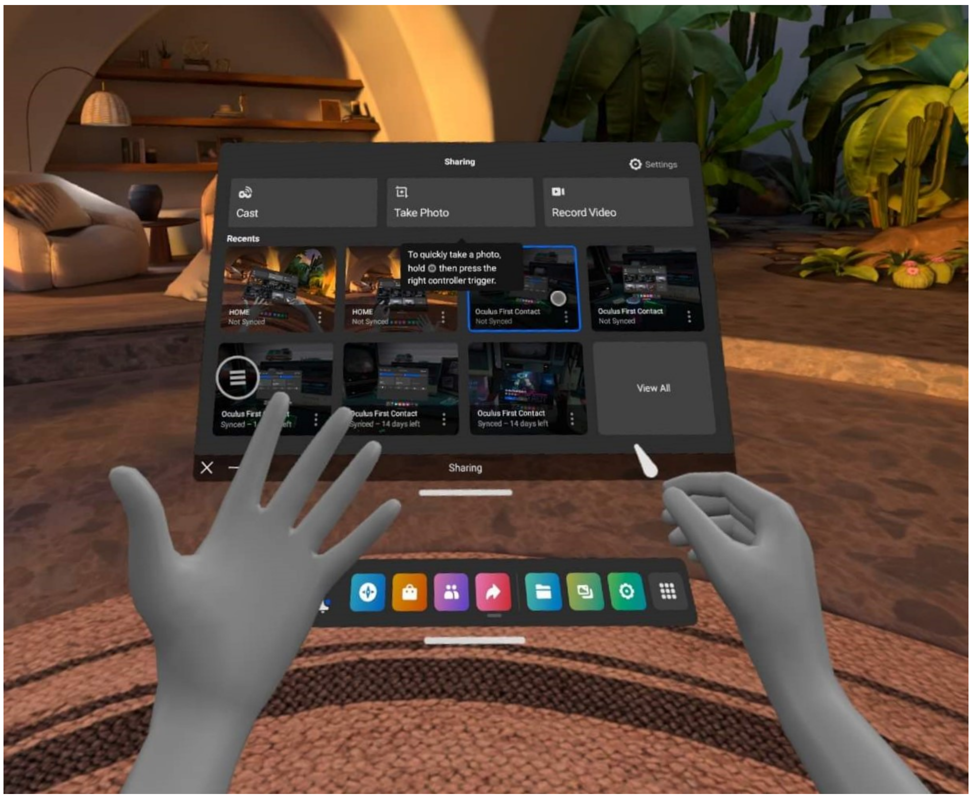
with a focus on the UI. In addition, tested applications provide insight into great user interfaces.

2.4.1 Input hardware

Standard 2D applications usually use a keyboard and mouse. In contrast, VR usually uses controllers (some VR sets, such as Quest 2 and Quest Pro, support hand tracking as well) with a somewhat limited amount of buttons. A regular full-size keyboard has 104 keys, which allows for many shortcuts, especially in combination with ctrl/alt/shift keys, which modeling software utilizes extensively.

The main difference and advantage of VR in modeling is the cursor position control. The mouse can only manage two axes, which is limiting when working in 3D space. VR controllers, on the other hand, track position in 3D space, allowing us to control the position on three axes.

There is no standard button layout for controller buttons, but all relevant VR hardware providers (except for HTC) follow a very similar set of inputs. A full input set includes five buttons (A, B, Grip, Trigger, Thumbstick button), thumbstick axes, and a Menu button on each controller. The trigger and grip buttons provide linear input with a range of zero to one, depending on the depth of the press. Figure 2.6 shows this layout on Quest 2 controllers. Some controllers also support capacitive touch for most buttons (the button is not directly pressed, but the user is physically in contact with it). However, as this is not universal for all controllers, the proposed application avoids this option to maintain compatibility with other devices.



To quickly take a photo, hold ⊙ then press the right controller trigger.

Figure 2.7: Quest 2 has in-built hand tracking, using its camera as input to track the hands. Gestures such as pinching, as shown in the image with the right hand, act as buttons.

Another alternative is hand tracking and hand control. Multiple sources are actively developing general tracking controllers, but there is no mainstream solution. Notably, Meta's Quest 2 and Quest Pro offer in-built hand-tracking based on computer vision via their camera input 2.7. However, since this is only available on some headsets, and even within them, usage is quite limited (e.g., tracking doesn't work well with any occlusion between the hand palms and camera). Another limitation is the lack of buttons/grips/triggers and thumbstick options. While hand tracking may support a few gestures to compensate for that, it is still lacking in comparison. Due to these reasons, this input type seems unfit for most current applications.

2.4.2 VR interface position

Researchers define the **content zone** as the distribution of 3D spacial into specific zones based on their function with the center [2]. Zones are defined by the distance from an HDM, vertical viewing angle, horizontal viewing angle, and the reaching distance of hands. Its goal is to define an ideal position for content and UI elements. A Figure 2.8 shows the horizontal distribution. All content that users interact with at a given time should be within the comfortable content zone. The curiosity zone is for content

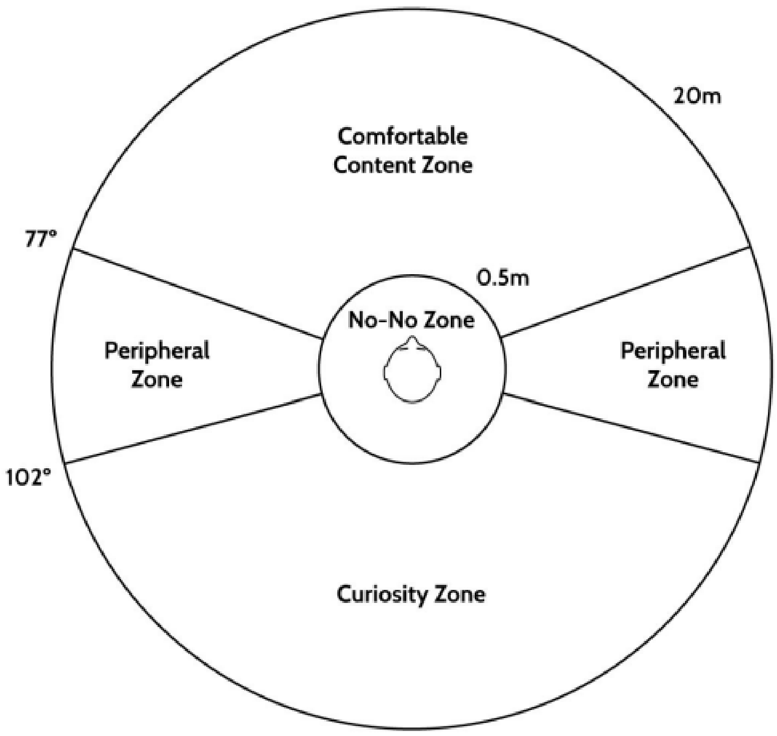


Figure 2.8: Horizontal distribution of the content zone. [2]

that is not necessary at the moment but can be easily accessed. The no-no zone is a space near the user where nothing should be placed, as it can be uncomfortable for the user. The vertical viewing angle from the HMD should be below 0 degrees, if possible. Considering this and the reaching distance, Figure 2.9 shows the process of defining the ideal UI zone.

2.4.3 VR specific problems

Creating a user interface (UI) in VR presents additional challenges compared to traditional 2D UI. Menus and text cannot be placed directly over the projection plane because there are two different rendered images, one for each eye. A simple solution would be to place the interface in the 3D space directly in front of the head-mounted display (HMD) with a constant offset from it. However, this approach can cause unpleasant experiences for users because in VR, people do not expect objects to be anchored to their head movement. Many sources agree that this is a poor approach [14].

A common approach is to use rectangular windows, which may be curved, in the virtual space in front of the user. This can be seen in the general Meta Quest 2 menu 2.10. These windows are generally movable and resizable, and may even follow the user's field of view (FOV) horizontally in non-continuous steps.

Many applications use this approach for their menu implementations, and it seems to be becoming a standard for menu interfaces. However, even this

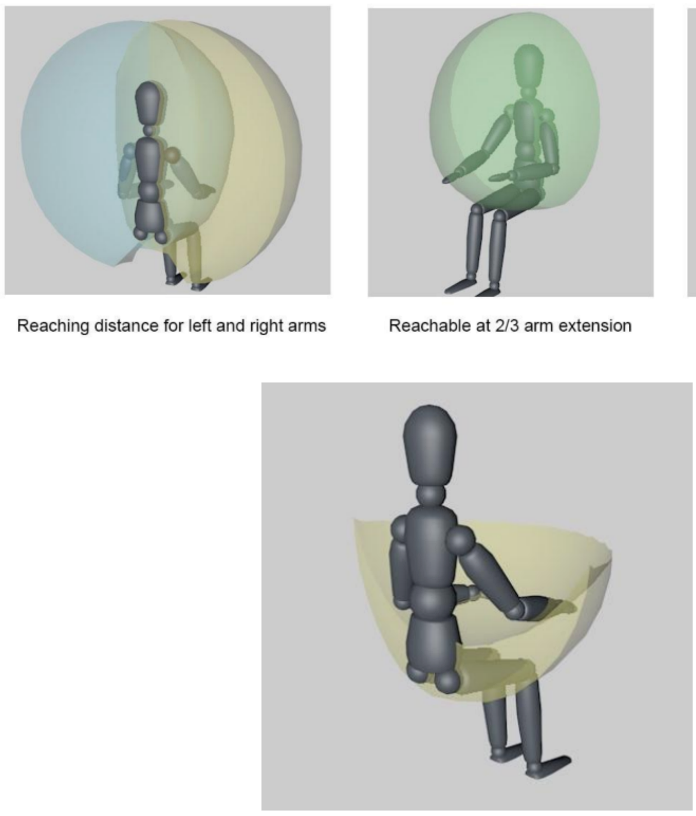

Reaching distance for left and right arms
Reachable at 2/3 arm extension
Intersection with no-no zone

Figure 2.9: Finding the ideal UI zone position, depicted by the yellow zone on the bottom image. [2]

approach has its drawbacks. The overlay menu in the application is drawn on the screen, even if there are objects in the scene that are closer to the headset, resulting in an unrealistic depth rendering. To prevent nausea, it is important to maintain a physically realistic view, if possible, which means ensuring the correct depth of elements and avoiding clipping of UI elements within the objects in the scene. This can be achieved by embedding the UI directly into the virtual environment as much as possible. Despite these issues, this approach is still widely used in cases such as menu overlays, as there is currently no better solution. Unfortunately, this effect cannot be properly shown in a monoscopic 2D image, so an example is not included.

2.4.4 Common VR interface elements

VR Interface is one of the most differentiating factors from the 2D interfaces, as most of them would not work for VR needs. Therefore a couple of popular VR interface elements emerged.

An interface is one of the most differentiating factors between VR and 2D programs since most of the 2D interfaces are not suitable for VR. As a result, a number of popular VR interface elements have emerged.

Floating windows in front of the user are a popular main menu solution,

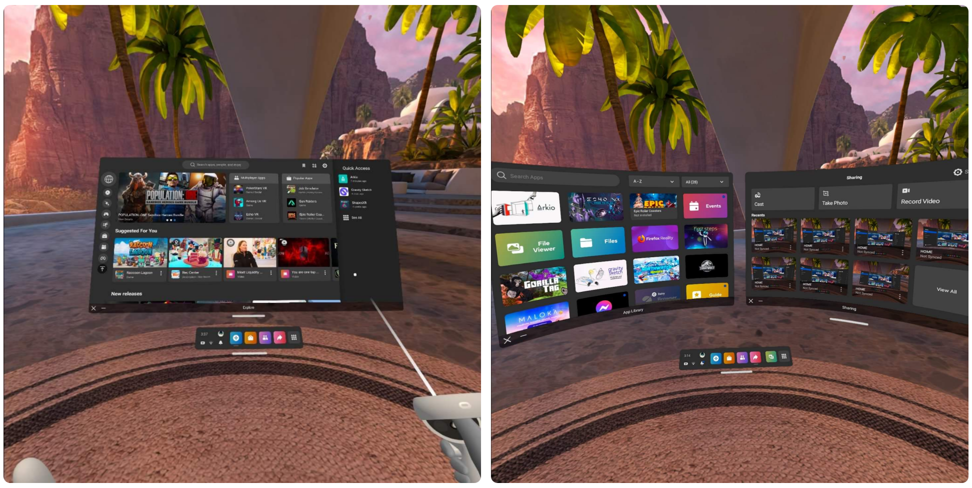


Figure 2.10: The Meta Quest 2 main menu interface can be used either as a standard flat smaller window, as shown on the left, or as multiple curved windows as shown on the right.

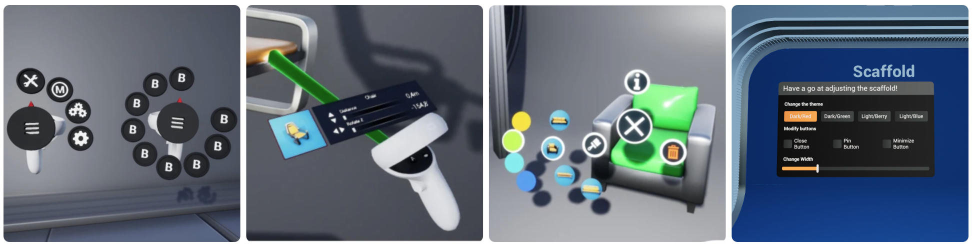


Figure 2.11: Examples of common VR interface elements. From left, radial menu, controller attached menu, object attached menu, floating window. (images from Advanced Framework - VR [20])

as they provide an intuitive extension from 2D interfaces. The windows essentially represent monitors in virtual form. A more integrated solution, and probably the most common one, is controller-attached menus, which use the other hand for interaction. Menus can also often be seen attached to specific places or objects in the VR environment. Other trends include radial menus for fast selection and menus with 3D elements since VR allows for the proper display of all three dimensions.

The interaction with menus and other UI elements is usually either by touch (moving the controller directly to the virtual button) or by using a laser pointer from the controller and a designated button for clicking. Touching is more intuitive but generally requires larger buttons and UI elements to be comfortably usable, which may not be practical in all situations. The laser pointer is more similar to desktop input, as it represents a cursor but in 3D space.

2.5 Existing applications

In this section, multiple 3D modeling VR tools are tested and compared, ranging from sketching to geometry modeling to world composition. These VR tools are counterparts to standard desktop applications such as game engines like Unreal Engine and Unity for scene composition, as well as modeling programs like Blender or Maya. It's worth noting that there are no clear boundaries between modeling and scene composition. For example, Blender, which is mainly a surface modeling tool, can also be used to create entire scenes. On the other hand, Unreal Engine 5 now includes geometry modeling tools directly within the editor. These trends can also be observed in VR tools.

A selection of significant existing VR modeling tools will be compared and evaluated based on the following factors. If a particular factor is highly valued for a given application, it will be used as inspiration for the design of the proposed application. Additionally, some applications offer unique features not covered by the specified factors. If these attributes are relevant to the context of this work, they will also be discussed.

2.5.1 Evaluated factors

1. **Locomotion** - Locomotion - Any type of movement mechanics.

2. **User interface** - Evaluated based on ease of use, intuitive design, clarity, and access to tools.

    - **Comfort and Effectiveness** - The comfort level (both mental and physical) of using the tool and the user's ability to be effective (e.g. ease of switching between tools or the number of steps required to perform a specific action).
    - **Application control and adjustability** - The availability and usefulness of settings provided for the user.

3. **Toolset** - The quality and quantity of available tools for creating block-outs, their ease of use, and their effectiveness in creating and manipulating a 3D scene.

    - **Object transformations tools** - Any tools related to manipulating primitives.
    - **Import/Export capabilities** - Does the application support import and export, and are there any limitations?

4. **Additional features** - Unique or noteworthy features of a given application.

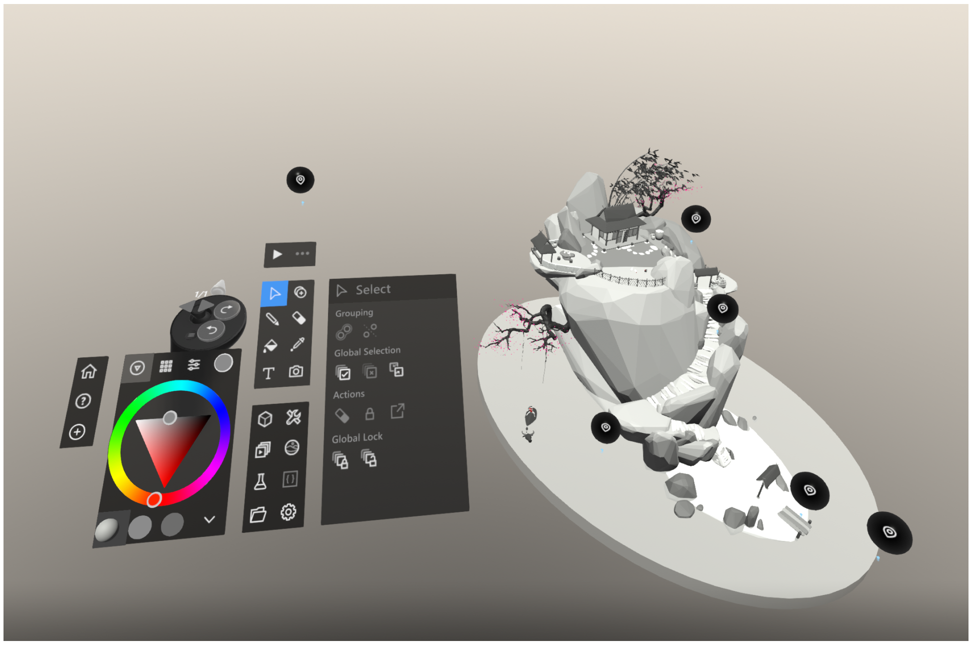


Figure 2.12: Example Maquette scene, small miniature on the right, controller with the attached menu on the left.

2.5.2 Microsoft Maquette

According to its own description, "Microsoft Maquette is a general purpose mock-up tool for spatial prototyping within virtual reality." [39]. As of 2022, Maquette appears to be the most complete VR model editor available. While its primary focus is more general than the goal of this work, it offers an extensive toolset for modeling virtual space and includes many clever interaction elements, some of which inspired certain design decisions in this work. In particular, object manipulation and transformation features in Maquette are very refined.

Maquette offers two types of locomotion: the standard teleport to the position of the laser aim, and a motion system that Microsoft calls "swimming". In short, swimming refers to the action of anchoring controllers in place and changing the relative position, rotation, and scale (derived from controller movement) between the user and a scene (see section 3.4.1). This type of movement will be referred to as swimming further on in this work. While swimming works well for working on miniature models directly in front of the user or small-scale scenes (such as a virtual room), it can be tiring for navigating large scenes or open worlds. Both modes of locomotion feel natural in VR without causing discomfort, which can be a problem with other types of locomotion. However, for the world sketching application, fast and simple navigation through the entire scene is deemed necessary.

This application offers many features, so the user interface has to accommodate a slightly more complex menu. There is a controller-attached menu that is interacted with using the other controller. The menu consists of a

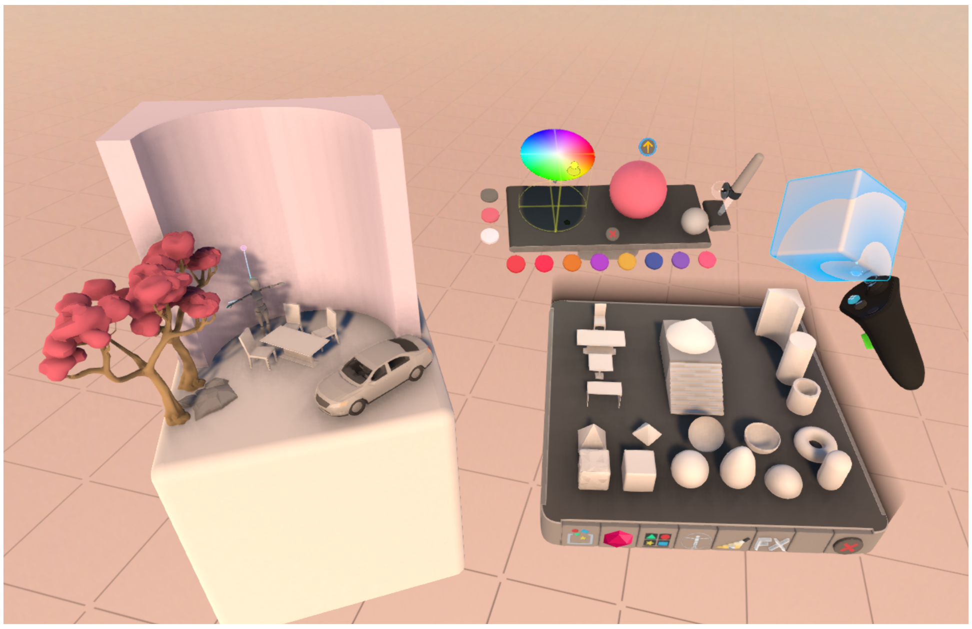


Figure 2.13: Example Tvori scene, small miniature on the left, model and coloring tools on the right.

set of windows, some of which switch based on the selected context. From this menu, users can access all settings, tools, primitives, and everything else. When it comes to object manipulation, Maquette probably has the widest toolset, providing all the basic tools for manipulation, duplication, deletion, multiple selections, coloring, grouping, and many others. On top of that, there are additional settings for all kinds of object snapping and general scene positioning. These tools provided inspiration for any sort of object manipulation tools for the design of BlockoutVR.

In summary, Maquette is an amazing tool with the best features for world-building among the tested VR applications. There are a couple of additional features provided, such as object locking to prevent accidental changes on specific objects and parametric primitives for a wide range of easily accessible shapes. However, the main problem with Maquette is that development has been canceled and it seems unlikely to be continued anytime soon. As a result, it may eventually become obsolete and unusable.

2.5.3 Tvori VR

For locomotion, Tvori [48] uses a combination of swimming and dragging. Dragging is a simplified form of swimming where one controller is anchored to the scene and the user is shifted relative to this controller.

The user interface is similar to that of Maquette and other discussed tools, with the main difference being that when menus are opened, they are not attached to the controller as in most cases, but rather to a position in space where they were spawned. They can be moved around arbitrarily, and multiple menus can be spawned at the same time, which provides fast access to settings

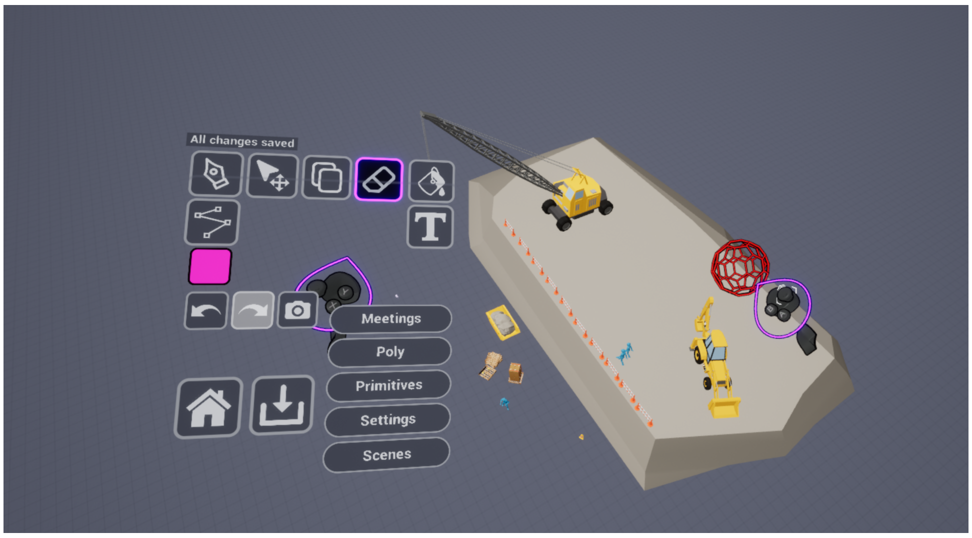


Figure 2.14: Showcase Sketchbox scene called "Construction training". Controller with tools on the left.

for multiple tools, but on the other hand, it can complicate movement and work in other areas as the menus have to be moved again. The control of Tvori is a bit more complicated and seems less user-friendly compared to other tested applications, as it provides complex additional animation tools.

The toolset of Tvori is similar to that of Maquette in terms of general object manipulation tools, but it has more limited modeling capabilities. Tvori focuses on 3D animation prototyping (which Maquette cannot do) and spatial UX/UI prototyping for AR and VR applications.

Tvori provides many unique additional features, but there is one particularly useful one: a transformation gizmo for selected objects. This gizmo is similar to the ones commonly found in desktop applications for moving objects along specific axes, rotating them, and scaling them. VR applications typically do not include this feature as users can grab the object directly, but the gizmo allows for fine changes without making unintended changes in rotation or position, which can often happen when an object is grabbed directly.

2.5.4 Sketchbox

Sketchbox [45] is a multipurpose sketching tool. Locomotion is done using a combination of teleport and swimming, similar to Maquette. Its user interface, while slightly primitive compared to others, has all features and settings together on one big menu, with one additional pop-up window. The menu is attached to one controller, and it can be switched to the other controller for left-handed users. However, the menu is big and seemingly designed for the left controller, so when attached to the right one, some buttons are harder to access and the menu is generally a bit cumbersome to use. During the design of BlockoutVR, care should be taken to prevent similar issues.

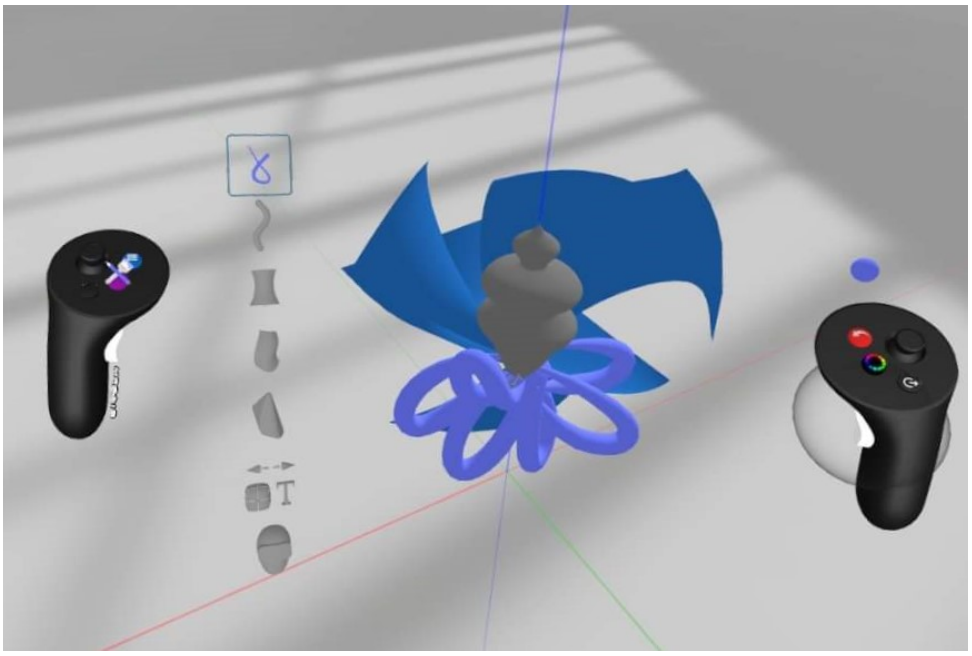


Figure 2.15: Example Gravity Sketch scene.

The strong point of Sketchbox's toolset, compared to other programs, is its ability to perform precise modeling with features such as measuring distances, space snapping, and placing guides. It is used in professional spheres for any sort of idea sketching and supports import and export in common formats. It focuses on line sketching and has a variety of tools for free drawing, straight lines, and other shapes.

In terms of additional features, Sketchbox is strongly focused on collaboration and is designed for a comfortable workflow for multiple people in one scene. It also has a range of tools for line drawing and some useful guiding tools for precise drawing in space, which are unique to this application. One interesting feature is the measuring tool, which allows users to place measuring lines directly in the scene and displays the distance between their ends.

2.5.5 Gravity Sketch

Gravity Sketch [17] is an amazing sketching tool with a great UI, providing easily accessible and usable tools for modeling shapes. It is also the only one of the main tested programs that can run natively on a standalone VR headset (Quest2), while other tools require a PC.

In Gravity Sketch, the user is statically set around the environment origin and the locomotion is very limited. The default room-scale is present, as with most applications. The only other option is to grab the model and move it around the user to the desired position. There is no other locomotion available, such as teleporting or shifting position. In the context of this application, this seems reasonable, as it focuses on product design and sketching and extensive

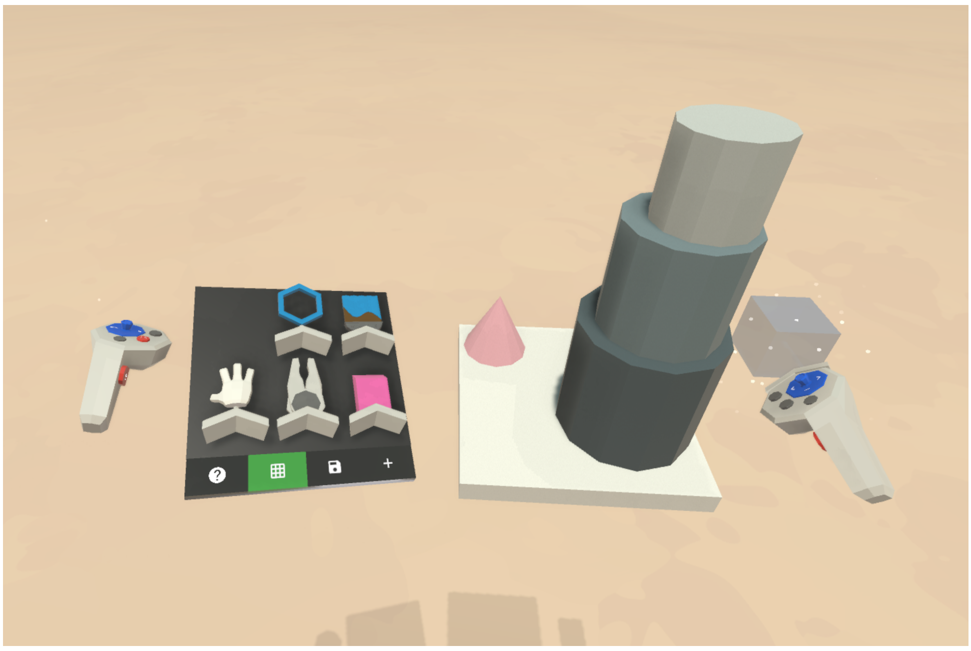


Figure 2.16: Example Blocks scene.

scenes are not expected to be built. The user interface is great and extensive in options, while still being intuitive. There are three menus for different tasks. Each menu is designed for its purpose. The first one has just a couple of buttons on the side of the controller for quick saving, help, or exiting the application. Then there is a settings menu and a tools menu.

The toolset in Gravity Sketch is the most unique among the other tools, but it is probably the least useful for a level blockout. The tools are mostly for creating different kinds of lines and surfaces, but the geometry is not connected among the elements. This clearly targets sketching and the creation of single models (or small scenes at most), and therefore it is unsuitable for inspiring the design of a world composition application. It seems clearly focused and useful for product design, for example, in the automotive industry.

As an application focusing on surface modeling, it provides additional features such as layers, 2D side views, and the ability to model around axes, as well as other standard desktop object modeling tools. The modeling tools use the VR controller smartly, allowing the creation of surfaces between controllers and the ability to sculpt the shape directly.

Although Gravity Sketch offers one of the most interesting tools, its focus is far from blockout, and few learnings from this application can be applied to a world sketching application.

2.5.6 Blocks by Google

Blocks [15] seems like a perfect application for introducing VR modeling to inexperienced users because of its simplicity and intuitive tools. It also has a great interactive UI, which is the best among the tested apps. However,

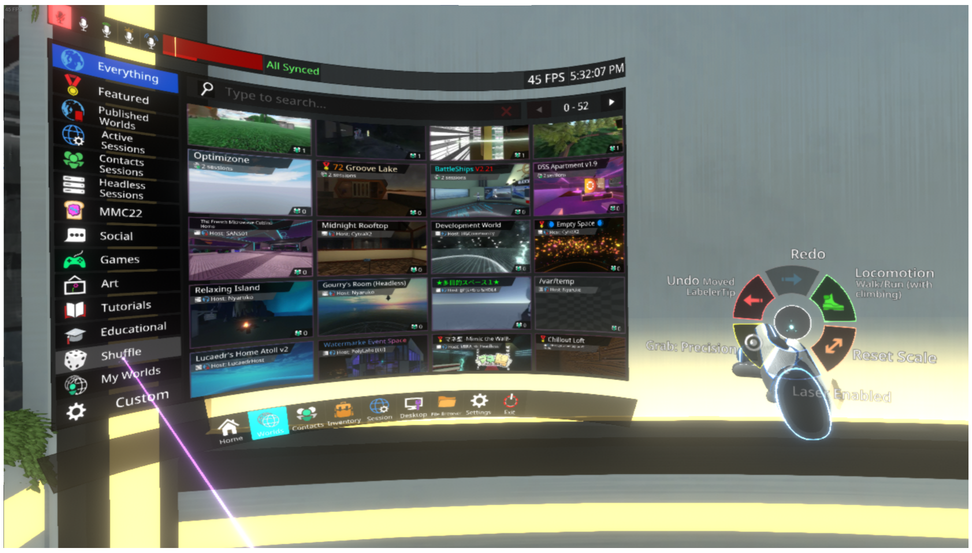


Figure 2.17: Example Neos scene.

due to its plainness, it is missing some core features for blockout, such as the ability to import your own models.

The locomotion in Blocks is simple swimming. The user interface is well-designed. Although Blocks has an advantage in simplicity because it doesn't provide as many features as the other applications, the UI is so well-arranged that it can be easily understood without any previous learning experience. It's worth noting that Blocks also provides a tutorial upon opening the application, making it extremely user-friendly.

The toolset in Blocks is very limited, with only six basic tools for common transformations, saves, and a couple of provided primitives, without the ability to import your own models. As additional features, Blocks provides a simple but powerful Modify tool for geometry modifications, such as extrusion, subdivision, and moving specific faces, edges, and vertices. Another interesting idea is the color palette on the back side of the tool menu, which would otherwise be unused. It is a little bit cumbersome to access as it requires twisting the hand, but it can be reached quickly without using the menu itself. Both of these features would be interesting in a world sketching application.

2.5.7 Neos VR

Unlike previous programs, Neos [40] is not a modeling tool, but a metaverse. A key feature of a metaverse is the ability to create and shape worlds, so Neos offers a wide range of tools for world modeling, even if it is just a subset of what it has to offer.

Neos offers a wide variety of locomotion types that can be swapped, including HDM-based ground movement, climbing, flying, and teleportation. The user interface is significantly different from the previous applications. There are multiple interfaces, including a radial menu on each controller for quick access to actions such as switching locomotion types, deleting or cloning objects, and other basic settings and interactions. There is also a dashboard in the form of a floating window in front of the user, which has multiple tabs and serves as the main menu, settings, inventory, and interface for basically anything else that is accessible.

Neos' toolset provides many tools for modifying objects in various ways, which are stored in the inventory as equipable tools. Objects are manipulated using a combination of laser and grip. This seems very useful, and the design chapter of this work is heavily inspired by Neos' object manipulation.

As a metaverse, Neos has many additional features compared to previous applications, but most of them are not relevant to a level blockout. However, Neos can provide design inspiration for locomotion, UI, toolset, and object interaction, if we can recognize useful features for this work.

2.5.8 Other VR tools

The sections above provide a description of existing applications that may offer relevant design ideas for creating the world sketching application. I have also tested and explored other accessible modeling VR tools, such as Arkio [3], ShapesXR [44], Adobe Medium [1], and Tilt Brush [16]. However, these tools either have a very specific focus or are not unique enough compared to previous programs.

2.5.9 Missing features (for world sketching) from existing applications

The proposed application differs from most of the tested apps in that it is a specific solution for blockout and level layout sketches, rather than a general solution. It is designed for a fast and intuitive workflow and a creation of large-scale, low-detail scenes, rather than the smaller but more detailed scenes or models that are more common in other applications. None of the existing solutions has this primary focus.

Most of the tested applications are designed to allow users to manipulate objects in a scene using controllers that directly grab the objects. This is useful for working on smaller models or miniatures, but it can be difficult to use in cases where the scene is vast, such as when creating a level blockout. In these cases, the user has to be scaled much larger than the scene objects to be able to reach everything with their hands. Otherwise, most objects will be out of reach. This work presents a design that combines direct grab with a laser pointer to allow users to work with both close and distant objects easily.

There is a significant similarity among the tested VR applications in terms of their aim for simple control and intuitiveness. Even Tvori VR, which seemed the most complicated and required a few tutorials on the menu and user interface, didn't take more than half an hour to understand. While simplicity in the workflow is undoubtedly pleasant for beginners, it may hinder more effective workflow for experienced users. As shown in the figures for each application, there is always a menu attached to one of the controllers where you can swap tools. However, for the simple action of spawning, moving, scaling, and copying an object, the user needs to re-select the tool at least three times. This means moving the arm from right to left and back to the model three times, which can be slow and tiring for continuous work. There are nearly no professional content creation desktop programs that can be used intuitively. Whether it's Blender, Maya, Unreal, Unity, or others, the learning curve may vary, but all require significant knowledge and the use of shortcuts for fast context switching. The small number of buttons on the controllers is not suitable for standard keyboard shortcuts. Therefore, this work proposes a design focused on fast context switching, which may not be as user-friendly but is more comfortable and productive for experienced users.

Chapter 3
Design

This design aims to define the core functionality of an application for building 3D worlds in VR. The core features include the ability to place, move, and manipulate objects in 3D space, import and export 3D models, and provide a variety of primitives (simple shape objects) and tools for modifying objects. The application should also allow the user to save and load their work. The user interface should be easy to use and meet all VR-specific requirements and recommendations recognized in the previous chapter. The design presents an abstract overview of the necessary features and quality-of-life elements without considering specific implementation details.

The chapter Analysis provides a lot of insight into the design and features of similar VR applications, particularly the existing applications (see section 2.5). Designing the application involved selecting the best elements from previous research and applications, as well as my own experience creating a VR modeling application and assembling these selected elements into a single functioning unit that provides the best experience for blockout/sketching worlds, being intuitive and effective. The design chapter serves as a template for the following implementation, but it is not expected that all the described features will be implemented, simply due to scope and time limitations.

3.1 Application definition

The previous chapters provided a strong indication of the expectations for the designed application as the goal of this work, but let's summarize the key properties that it is trying to achieve. This includes defining expected control, features, accessibility, and other factors based on the previous research and existing applications.

Definition of expected application: A VR application for general HMDs with two controllers as input devices. It is intended for creating sketches of virtual worlds (level blockout/world composition) and allows users to navigate the final scene naturally. It provides a toolset for manipulating and transforming objects (model transformations, spawn, delete, clone, etc.) and simple object geometry modifications. It offers a combination of multiple locomotion types for comfortable and effective navigation through the scene, suitable for both new and experienced VR users. It has a dynamic control scheme that can be used by both right-handed and left-handed people. The user interface is friendly and user-friendly. It has settings for object-related tools and auxiliary tools to control the application (save, load, and other settings). The tool is able to import existing models and export models (and the entire scene) in widely used formats that are compatible with popular game engines. Objects in the scene are arranged in a hierarchical structure. By default, new objects are unrelated, but users can parent them to each other.

3.1.1 Functional requirements

1. Intuitive user interface
2. Comfortable locomotion
3. Save/load scene
4. Importing models
5. Exporting scene
6. Undo/Redo object operations
7. Object manipulation tools
8. Geometry tools
9. Parenting objects
10. Group selection
11. Multiple modes
12. Snapping

3.2 User interface and experience

The user interface (UI) and user experience (UX) are integral components of virtual reality (VR). VR technology allows users to interact with digital environments in a natural and intuitive way, providing a sense of immersion and presence. The UI and UX in VR refer to the design of the interface and the overall experience of using VR technology.

VR UI and UX design focuses on creating intuitive, user-centered interfaces that enable seamless interaction with virtual environments. This involves considering factors such as the visual design of the interface, the layout of controls and buttons, and the use of haptic feedback to provide a sense of touch. A convincing design can enhance the user's sense of immersion and enable them to fully engage with virtual environments, leading to an enjoyable and intuitive VR experience. Good VR UX should also take into account the user's psychological and physiological responses to the virtual space in order

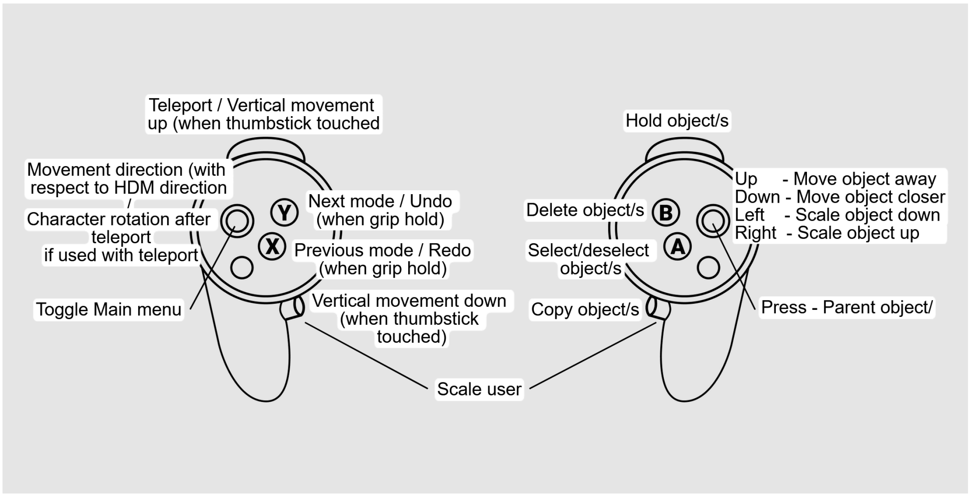

Teleport / Vertical movement up (when thumbstick touched
Movement direction (with respect to HDM direction
/
Character rotation after teleport
if used with teleport
Toggle Main menu
Y
X
Next mode / Undo (when grip hold)
Previous mode / Redo (when grip hold)
Vertical movement down (when thumbstick touched)
Scale user
Hold object/s
Delete object/s
B
Select/deselect object/s
A
Copy object/s
Up - Move object away
Down - Move object closer
Left - Scale object down
Right - Scale object up
Press - Parent object/

Figure 3.1: The design idea for the controller scheme (when the Object mode is selected). The underlying controller sketch was made by [41].

to prevent discomfort. This includes minimizing motion sickness, a common side effect of VR use, and providing clear visual and audio cues to guide the user's movements and actions.

3.2.1 Controllers

Each controller has a different role and different functions. The first one, referred to as the Menu controller, has all the supporting functions such as locomotion, holding the Main menu, the Primitives menu, and context switching for the other controller, which is called the Tool controller. The Tool controller has all the tools of the currently selected mode (which is explained in more detail in section 3.3) and provides interaction with scene objects.

By default, the left controller is the Menu controller and the right controller is the Tool controller, but they can be switched at will through the menu settings. For additional clarity, each controller should have a description explaining the current function of each button. This description can be enabled or disabled through the Main menu.

There is one interaction for both controllers that allows the user to change their scale. By holding both grip buttons and moving the controllers away from each other, the user will scale up. Similarly, moving the controllers closer together will scale the user down.

Menu controller

The Menu controller has mostly supporting functions for general navigation, control, and settings. Due to the limited number of inputs, some functions change based on the controller's state. However, the defined behavior for specific buttons is fixed, unlike on the Tool controller. For example, during teleportation, the thumbstick does not control movement, but rather the direction of the teleport.

The primary function of the Menu controller is locomotion. The trigger button is reserved for the teleport function, and an optional teleport direction can be added using the thumbstick. The thumbstick is also used for HMD-based movement. The trigger and grip buttons control vertical movement when the thumbstick is touched. The Y and X buttons (or B and A buttons on other controller) cycle to the next or previous mode, respectively. The Y and X buttons also serve as undo and redo functions when the grip button is pressed. Access to undo button only with a grip press is to prevent accidental undo calls, which is the issue that some tested VR applications suffered.

The Main menu is attached above the Menu controller. The Primitives menu, which is used more frequently and therefore has a preferred position, is attached to the inner side of the Menu controller.

Tool controller

A primary controller for scene interactions, the buttons on this controller have variable functions depending on the currently selected mode. However, there is consistency whenever possible. For example, the trigger is always used for interaction with objects, while other buttons may offer different kinds of interaction or supporting features. Each mode has a different layout and specific controls, which will be explained individually in the modes section 3.3.

The controller has two interaction components. The first is the *grip hold*, which represents the action of holding an object in hand. The visual cue and center of the grip's collider is a small sphere above the controller. The second component, a laser functioning as a pointer, called *laser hold*, originates from the same location as the grip (see Figure 3.2). Both interaction components are treated similarly in terms of interacting with objects, but the *grip hold* has priority over the *laser hold*. Therefore, if the grip interacts with an object, the laser is ignored.

3.2.2 Visual interfaces

User interfaces are a crucial difference between standard and VR modeling, as they must take into account the difference in dimensionality (see section 2.4). This has been thoroughly explained in the analysis.

Main menu

The Main menu is the user interface for settings (general scene and tool-related) and auxiliary tasks (such as save, import, export, etc.). It is not designed for frequent use, but rather as a simple way to access all non-modeling related tasks. It is a simple, curved 2D panel that is attached to the main controller and rotates to face the user. The tool controller has a laser pointer for aiming at the menu, and the trigger button serves as a press in

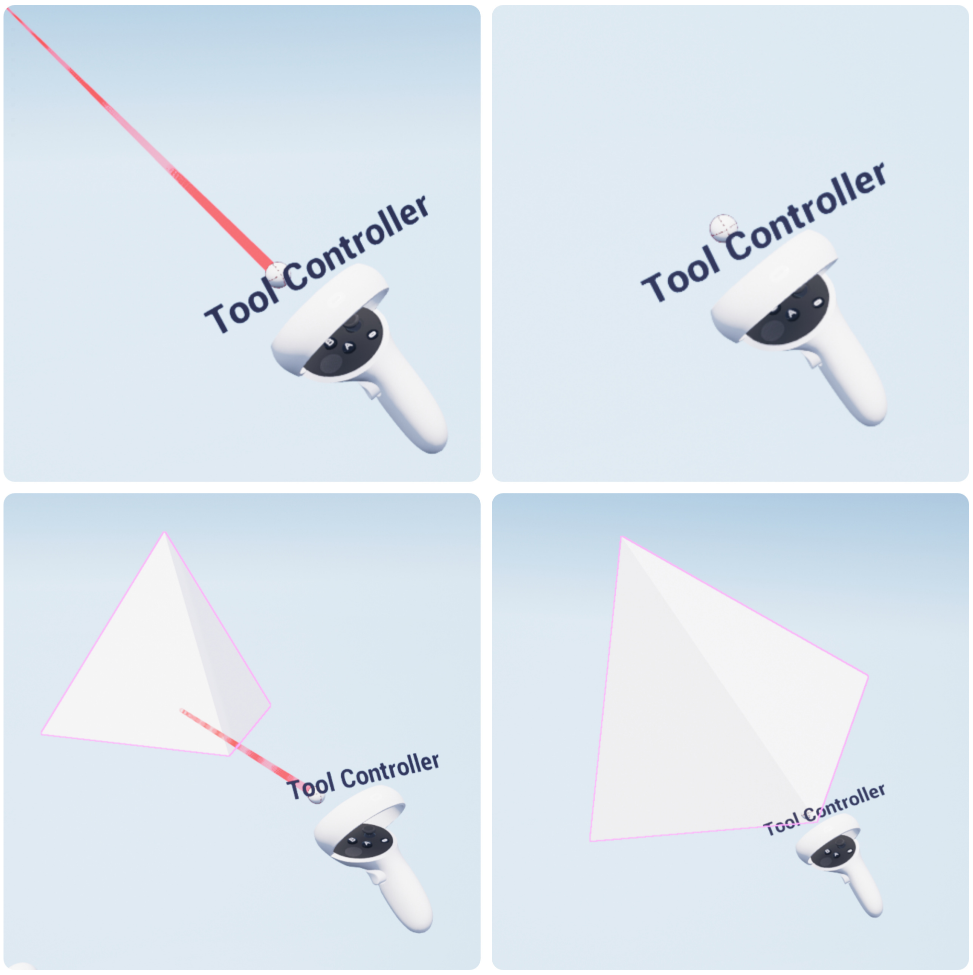


Figure 3.2: The upper two images depict possible interaction components - a laser and a grip sphere. The images below show how these components interact with an object.

the menu. It provides access to all features and settings described in sections 3.4 and 3.5.

The main menu is expected to have multiple tabs (as shown on draft 3.3), each for related things. One tab is reserved for showing all objects in the scene in the form of a scene collection view. This is basically a panel with text names of objects and visual connections and indentations to depict the hierarchy, similar to what typical 2D programs offer.

Primitives menu

A 3D panel with available primitives is attached to the Menu controller and provides a preview of 3D miniatures of all primitives. The primitives can be selected by pointing at a given primitive or by directly grabbing it. The panel is hidden in modes where it is not relevant.

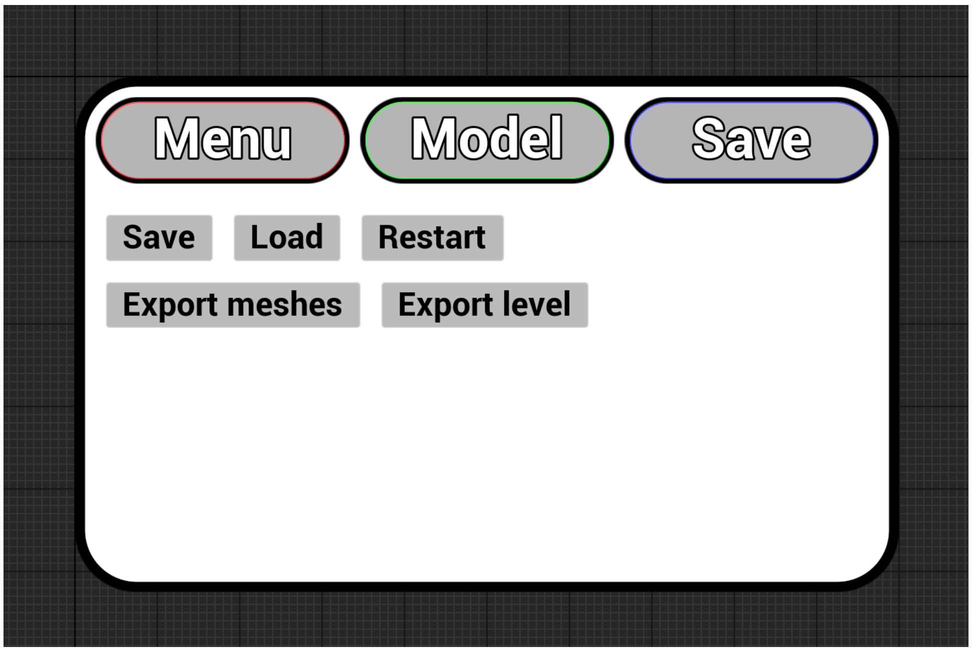


Figure 3.3: Original Main menu draft (used in the prototype), shown with "Save" tab selected.

3.3 Modes

To design a control, it is necessary to define the button layout to work with. In this case, the most common VR controller input, and the one considered in this work, is depicted previously in the section on controller input (see 2.4.1).

It was mentioned that the amount of input buttons is rather limited, so not all features can be mapped directly on the buttons. A common solution in the tested applications was to switch tools in the menu, rather than assigning specific functions to buttons directly. This makes sense as the issue of insufficient buttons is moved to the menu, where enough space can be dedicated to it. However, this can result in an ineffective workflow, since only one tool can be selected at a time. Even relatively simple operations may require unnecessarily many tool swaps.

To address this issue, two techniques are combined. The first is defining modes for context switching. This involves defining sets of logically related tools and implementing each set of tools together on a single layout. This way, instead of having many different layouts (one for each tool), only a few different layout modes are necessary to contain all of the tools. The modes can then be switched by the user. The second part is fast switching the context (the mode layouts) not through the usual menu attached to the other controller, which requires moving the whole arm, but instead via X/Y (A/B) on the Menu controller directly. This does not require the user to move their hand or look at the controllers. While moving the arm might not seem like a big issue, repeating this action many times over during prolonged work can

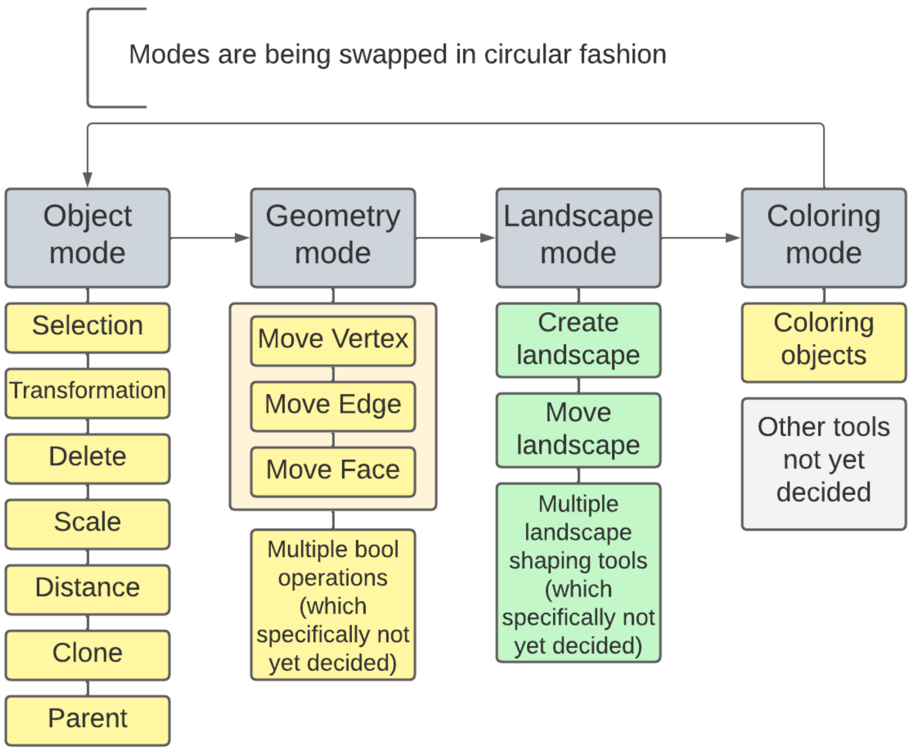

Modes are being swapped in circular fashion
Object mode
Geometry mode
Landscape mode
Coloring mode
Selection
Transformation
Delete
Scale
Distance
Clone
Parent
Move Vertex
Move Edge
Move Face
Multiple bool operations (which specifically not yet decided)
Create landscape
Move landscape
Multiple landscape shaping tools (which specifically not yet decided)
Coloring objects
Other tools not yet decided

Figure 3.4: Draft of modes tools and circular swapping. Yellow nodes represent tools affecting the objects, while green affects a landscape.

cause fatigue, a common problem in VR. It is essential to make it visually evident which mode is currently selected to prevent any confusion during mode switching. A simple and effective solution would be to color the Tool controller mesh in light colors, each color representing a different mode.

The following subsections explain each mode individually. Note that the specific mode only affects the Tool controller (while the Menu controller stays the same during any mode). Therefore, buttons such as the trigger always refer to the Tool controller's trigger in this section.

3.3.1 Object mode

This is the primary mode, including the essential transformation and interaction tools for creating a scene with primitives. It is expected to be used most of the time. Interaction with the objects is done using a combination of the laser pointer *laser hold* and grip *grip hold*. This allows the user to comfortably work with both close and distant objects. The tools aim to provide the same interaction with both *laser hold* and *grip hold* if possible, but this is not feasible in all cases. The following subsections describe the provided tools, their functions, how they use the different object states for interaction, and if their interaction differs between *laser hold* and *grip hold*.

The objects in the scene may have multiple states: *normal*, *focused*, *se-*

lected, and *held*. *Normal* is the standard state of the object when nothing is happening. An object is *focused* when it is being pointed at with the laser pointer or within the range of the grip hold. Objects are *selected* when they are marked for group selection (more in section 3.3.1). *Held* is an object that is being directly interacted with through the trigger button (more in section 3.3.1). In the *held* state, there can be multiple objects if there were previously multiple objects in the *selected* state. These states are not exclusive; for example, an object can be both *focused* and *selected*.

During this mode, the Primitives menu (see section 3.2.2) appears, attached to the Menu controller, to provide primitives as the building blocks for the scene.

Selection tool

The ability to select and manipulate multiple objects as a single entity. It can help to save time and increase productivity, especially when working with complex models. The capacity to move parts of the scene is essential for the level design process. While having multiple primitives in a parent-child hierarchy can support this to some degree, it is often necessary to move objects that are logically connected but not part of the same hierarchy. For this purpose, a tool is available to select multiple objects, and other tools can then be applied to all of the selected objects.

This tool transfer objects to the *selected* state. When the A button is held down, every *focused* object is set to *selected*. This enables the user to hold the button and point at all of the objects that they want to select. A single press of the A button on a *selected* object will deselect it. A fast consecutive double press of the A button will release the selection and set the objects to the *normal* state. The *selected* objects are highlighted for visual clarity.

Transformation tool

By pressing the **trigger**, *focused* and *selected* objects become *held* and their position and rotation can be changed.

The *laser hold* can move *selected* objects to any position that the laser is aiming at, up to the distance of the initial hold. In addition, by rolling the Tool controller to the left and right, the selected objects can be rotated around their local z-axis. With the *grip hold*, *selected* objects follow the controller's position and rotate with the controller in any of the three axes.

Delete tool

Pressing the B button deletes the *focused* and *selected* objects, or the *held* objects if any are in an active hold.

Scale tool

Objects that are *held* can be scaled up by pushing the thumbstick to the right, and scaled down by pushing it to the left. This works the same for both the *laser hold* and the *grip hold*.

Distance tool

Pushing the thumbstick up and down will increase and decrease the distance of the *held* objects, respectively. If the objects are *held* via the *laser hold*, then the range of the laser is extended and shortened accordingly. If the range decreases to zero, the *held* interaction type is switched to the *grip hold*. Similarly, increasing the distance from the *grip hold* will result in a transition to the *laser hold*.

Clone tool

The clone tool works the same for both *holds*. There are two ways to use the copy tool. The first option is to have objects in an active *held* state. By pressing the grip, copies of *held* objects are spawned at the current position, while the original objects remain in the *held* state.

Another way to use the clone tool is to hold down the grip (no objects in *held* state) and then press the trigger with the *focused* or *selected* objects. Instead of the standard interaction where the *focused* and *selected* objects become *held* (as explained in 3.3.1), they are copied and the copies become *held*, while the original objects remain in the scene untouched and are released from the *selected* to the *normal* state.

Parenting objects

Blockout typically relies on primitive objects to create scenes. The ability to parent these primitives allows the user to create more complicated objects, as the parent object and all of its children can be treated as a single entity.

By pressing the thumbstick, the *focused* object is assigned as the parent of the *selected* objects. If the *selected* objects had a different parent previously, they are re-parented to the new one. The entire children hierarchy remains persistent even when the parent is changed or a new one is assigned.For the same interaction where the *focused* object is already the parent of the *held* objects, the parent-child relationship is removed, and the objects become individual again.

3.3.2 Other modes

There are four other defined modes. They are described and their features are outlined, but unlike the primary object mode, there are no specific instructions on how to implement them. These modes will not be implemented in the prototype application, but they are important enough for the blockout process to be included in the design chapter.

Geometry mode

From the geometry tools that were researched, there were two that seemed particularly useful for level blockout in VR. First are Boolean operations, which are described in section 2.3.1 along with information on how they work and how they can be useful. The other useful method is the direct modification of geometry by shifting faces, edges, and vertices, which is described in section 2.3.1.

There would also be other tools for modifying the object in various ways, such as scaling along individual axes.

Coloring mode

Colors are important in level blockout because they allow the user to assign meaning to objects. They can be used to depict important places in a scene, indicate the expected color or texture of the final material, or express intentions about the interaction between the user and scene objects. Even before any logic is implemented, the use of colors can make it clear to the user how objects will react to them based on their color. The specific meanings of the colors used are up to the user to define.

Landscape mode

When modeling levels, there are two main types: closed levels and open levels. Closed levels are enclosed spaces such as a series of connected rooms or cave dungeons, while open levels are large, accessible environments like a village surrounded by woods, fields, and mountains. There is no clear border between closed and open levels, as there can be partially open levels, but the distinction is based on whether a landscape is used or not. Landscapes are typically large meshes that represent the land, and they are useful for world sketching. During the blockout phase, it is possible to simulate a landscape with primitives instead of using actual landscape tools. However, providing the correct landscape tools is a useful way to distinguish a world sketching tool from common VR modeling tools, which do not typically offer anything related.

The difference between casual primitives and landscapes is that landscapes can be sculpted. This means there are tools available for adding and subtracting mass to create the desired shape of hills, valleys, and other features that the designer envisions.

Testing mode

Making a level blockout is not just about how the level will look, but also how it will feel to play in the game. This mode provides tools for common types of game locomotion and interactions. while editor movements like floating and teleporting are great for browsing the level, the player's experience will be different if they can for example only walk. This mode aims to provide

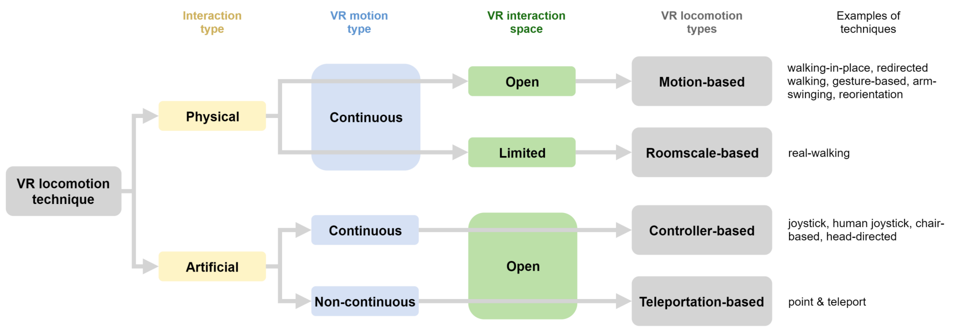


Figure 3.5: The VR locomotion typology defined in [6].

the most common experiences that players might recognize in a custom game, while taking into account the fact that different games have their own unique characteristics.

3.4 Other features

Core features that are not specific to any mode include locomotion, settings options, and general features like import/export, load/save, and undo/redo.

3.4.1 Locomotion

Locomotion is essential for many applications, but it can also present potential pitfalls as most of the comfort risks mentioned in section Preventing Discomfort can be triggered by improper movement techniques. Therefore, there has been a significant amount of research on this topic. Many studies try to identify discomforting effects associated with common locomotion systems, some even attempting to find a new system that eliminates the problems of existing ones [18]. Others explore how different movements can negatively impact distance estimations in virtual reality [23]. The VR locomotion typology breakdown in Figure 3.5 from [6] provides a breakdown of the locomotion types.

There is no single best type of moment. For example, continuous motion can cause motion sickness due to the confounding effect on the vestibulo-ocular reflex. Teleportation can also lead to spatial disorientation [18]. Some types of motion may be more comfortable for users, but may not be as efficient. This is why this design combines multiple approaches, allowing users to choose what they feel comfortable with.

Room-scale Movement

Room-scale in virtual reality (VR) refers to the ability of the user to physically move around within a defined workspace while using a VR headset [27]. Physical motion is accurately translated to the virtual space, making it the most natural and comfortable movement. The only limitations are the

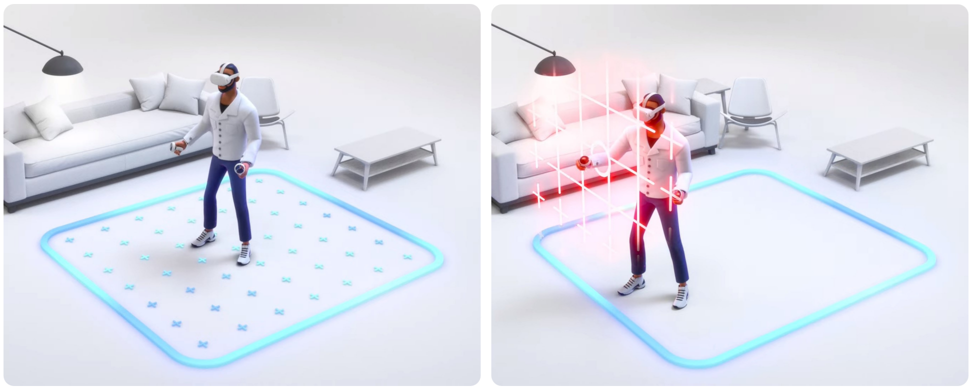


Figure 3.6: Visualization of the room-scale (with its Guardian system) in Meta Quest. On the right image warning of the user when they reach the border. [37]

size of the physical space and the hardware tracking capabilities. Most VR applications that involve locomotion support this type of movement, although the downside is that the room-scale is usually only a few meters in diameter, which is too small for many VR experiences. Nonetheless, it should be used whenever possible.

Most VR headsets have room-scale built in. Figure 3.6 shows a visualization of the Meta/Oculus room-scale mechanism. This feature can be set up for the headset and specific environment once, and it will then work as an overlay for any running application. This feature is typically accompanied by a control mechanism that activates if the user gets too close to the border, alerting the user to stay within the designated area.

Teleport

Teleport is a widely used VR locomotion method [6] that can be found in many VR applications that support movement beyond just room-scale. Its popularity is due to its user-friendliness and comfort, which makes it ideal for non-casual VR users. Teleport implementation often involves an arc from a controller to the ground to determine final position.

Sometimes, additional rotation is offered on teleport, which changes both the position and rotation of the user, providing more variability. Research on teleport locomotion [7] has found that the direction component can negatively impact the user experience. To address this issue, some approaches propose using a two-phase teleport, where the user can use standalone teleportation with a single button, but can also add direction at will using a thumbstick. This seems natural, but many existing applications force the user to choose a direction before teleporting.

The default scene of the blockout application is empty, so teleport is not suitable as a locomotion method in this case, as there are no objects to be teleported at. However, it should still be included for sensitive users.

HDM Orientation Based Movement

This type of movement combines thumbstick input with the current head-mounted display (HMD) rotation. It reads values from the Menu controller's thumbstick, with the X-axis value ranging from 1 to -1 for moving forward and backward, respectively, and the Y-axis for moving left to right in the same range. These values are combined into a single directional vector, which defines the orientation of the expected movement. However, this movement direction vector is then applied to the local coordinates of the HMD (rather than the global coordinates) and shifts the user in the final direction. This may seem confusing at first, but it simply allows the user to move forward in the direction they are looking by pushing the thumbstick forward. This makes this locomotion type feel relatively natural and easy to use.

Along with teleportation, this is the most suitable type of movement for long-distance traversal in a scene. However, unlike teleportation, this type of movement does not require any attachment points in the scene to which the user is moved.

HMD orientation-based movement is fitting for a world-building editor. While this type of movement feels natural and is easy to use, it does violate some of the comfort risks mentioned in the section on preventing discomfort, notably vection and vestibular sense. This can be unpleasant for new and sensitive VR users. For those users, there are other movement types available. For experienced users, however, this type of movement is expected to be the primary locomotion option.

Vertical Movement

Vertical movement can be seen as an extension of the previous orientation-based movement. While it allows for vertical movement, it is quite impractical if the user needs to go directly up or down because turning the head straight up or down would be straining and could potentially cause the head-mounted display (HMD) to fall off the head. Therefore, an additional mechanism for vertical movement is introduced. The Menu controllers' trigger and grip buttons move the user vertically up and down, respectively. The user must be touching the thumbstick to use these buttons, as the standard trigger and grip buttons are reserved for other purposes. The trigger and grip buttons support single-axis input so that the vertical movement speed can be determined by the depth of the press.

Swimming

Swimming refers to the ability to move through a virtual space in a manner similar to real swimming. In VR, the head-mounted display (HMD) typically defines the center of the body, and the controllers are positioned in relation to it. When swimming is active, the controllers anchors to their current position in the scene, and the user's virtual body position is moved instead. This movement is often combined with functions to scale the user based on the distance between the controllers and to rotate the user based on the angle between the controllers, projected on the horizontal plane (as rotating the user upside down would be an unwanted action).

This locomotion can also be described as grabbing the whole scene and moving it around the static user, with locked roll and pitch to allow only horizontal rotation. It refers to the same movement as described in the previous paragraph, just from a different perspective.

This type of movement allows for comfortable navigation in small areas of the scene due to the quick rotation and scaling. While it can be used as the sole movement type in an application, it is unsuitable for traversing large scenes.

3.4.2 Save/Load scene

The standard function for continuous workflow with arbitrary closing and reopening of the application is the save and load functions. The save function saves the current scene to local storage, which can be extended to save on cloud storage for increased accessibility and backup. The load function loads the saved scene from the save file into the application for continuing work.

This mechanism is solely for the user's convenience and is not intended for importing or exporting models. As such, the implementation and format used for the save and load functions are not specified. These functions are accessible in the Main menu.

3.4.3 Import models

Although blockouts mostly rely on preset primitives, it is important for level designers to have the option to import their own 3D models. This design considers that and offers this option. The workflow for using imported objects is different from that of primitives, as they are not expected to be used as often, but rather as an additional option.

The imports can be accessed through the Main menu. In order to be recognized by the application, 3D models must be placed either directly in the designated folder or defined in the location in the configuration file. An additional option would be to allow the user to browse folders directly in the application, but using folder browser in VR is cumbersome and adds considerable implementation complexity without significant benefits.

The application should be able to load 3D models in common formats such as FBX and OBJ, and optionally any other existing formats.

3.4.4 Export scene

A key function in BlockoutVR is the ability to export the created scene/world in a format compatible with game engines, when the blockout is finished. The default export format selected is glTF, managed by the Khronos group. "glTF is a royalty-free specification for efficiently transmitting and loading

3D scenes and models in engines and applications. It minimizes the size of 3D assets and the runtime processing required to unpack and use them." [24].

While glTF is a newer specification, released in 2015, compared to obj or fbx, it is growing in popularity and is already supported by most relevant programs. Therefore, considering its pros and cons, it seems like a good choice for transmitting world sketches due to its efficiency in loading 3D scenes. Additional export formats can be implemented based on user preferences.

3.4.5 Import reference images

It is common practice to use images as references in 3D modeling. Therefore, even VR applications should have this feature. A simple ability to import 2D images in standard formats such as PNG and JPEG and place them in a custom position, rotation and scale in 3D space, with the option to align them to the world axes, would be sufficient. Both import and axis alignment settings should be accessible through the Main menu.

3.4.6 Undo/Redo

A basic tool for consecutively undoing or redoing the last changes made in the scene. Changes in menu settings are not included for undoing. That is for clarity on what changes have occurred, and the menu is expected to be simple, so undoing settings there seems unnecessary.

Both when undoing and redoing, the user should be notified of the specific change that occurred, as some changes can be very minor and not immediately visible. This notification should take the form of a sentence describing the action that was reverted or restored for object A, such as "Cube primitive rotation reverted". This helps the user to understand what has happened and not worry that a major change has occurred outside of their field of view.

The undo stack should be limited only by the available memory, rather than a hard-coded limit on the number of changes, as is still common in some programs.

3.5 States - Settings

There are multiple parameters that can affect manipulations with objects. Each of the following settings can be enabled or disabled, and some may offer additional tweaking options.

3.5.1 Surface snapping

Surface snapping is a common technique in CAD applications that constrains the movement of selected objects to a specific surfaces. This allows the user to properly align objects. Primitives are typically not designed to use collisions, so surface snapping can be used as a method for achieving consistent alignment.

Surface snapping can be implemented using a combination of geometric algorithms and heuristics to determine the closest surface to the selected objects. The selected objects are then positioned so that the bottom of the objects intersects (or is slightly above) the closest found surface.

3.5.2 Grid snapping

Grid snapping is another common snapping technique that relies solely on the 3D space itself - that is, the global coordinate system or a predefined custom local coordinate system. Objects are snapped to the closest grid point in a regular grid of points in 3D space. The density of the points can be determined by their distance from each other on each axis. By using a custom local system instead of the global one, other factors can be modified, such as the rotation of the whole grid. This is useful for aligning objects at specific angles, which would not be possible otherwise.

3.5.3 Angle snapping

Angle snapping is a technique that differs from the previous two because it doesn't affect the location of objects, but rather their rotation. It is particularly useful for creating symmetrical designs and aligning the rotations of objects with each other. When placing objects, angle snapping locks their rotation at specific angles only.

3.5.4 Grid lines

Shows a transparent or dashed 3D grid over the whole scene (or at least near the user). The grid size can be assigned.

3.5.5 Guide lines

Guidelines are visual aids that help position objects. They can generally show distances from other elements or other values if needed. In combination with grid snapping, they can be used for precise modeling with proper alignments and specific lengths.

3.5.6 Environment

To create the image of a complete scene, the application should provide common scenery elements such as a water plane/ocean, sky (sun and clouds), daylight, weather, fog, and mountains around the center of the scene. These elements can easily and effortlessly set the tone of the created world.

Each scenery actor can be individually enabled or disabled through the menu.

3.6 Integration to game engines

Part of this work involves considering integration into game engines for real-time synchronization, specifically the one-way communication from this application to an engine or other software. This option was researched (for the Unreal engine) and seemed possible through an Unreal editor plugin that would update scene information based on the received data through a network communication channel using TCP. For the best compatibility with other engines, a reasonable solution is to develop a simple, uniform communication API for the Blockout application and then create individual plugins for integration with specific engines using this API.

On any change in the scene, this API would send batch data (in JSON or any other easy-to-parse format) with all objects whose state (transformation) was modified, along with the new state (transformation) of each object. The plugin would then only need to read the data and apply it to the corresponding objects or create new ones if they don't exist yet.

Another idea for the API solution was to send operations and a list of references to which objects the operation is applied, which is a common practice to reduce the amount of data sent. This would generally require fewer data to be sent over the network, but it has some significant downsides. The first proposed API could theoretically struggle with operations on many objects at once (as more data has to be sent), but modern networks should have no issues, especially between devices on the local network, which is the expected use case. In addition to this, the first API has two big advantages. Firstly, since the actual state of the object is sent, the engine always has correct and precise information, and there is no risk of accumulated error through multiple operations. The other advantage is that the implementation of the plugin for any software is far simpler since it is only necessary to create an object and set its transformation parameters. The second API would require implementing all operations in every plugin, making its implementation far more complex and increasing the risk of introducing a bug and possibly creating an inconsistency between engines, as each plugin could interpret operations slightly differently.

This integration would certainly be useful and should be considered as a possible expansion, but in the context of this work, the possibility of integration will not be discussed further or implemented in the prototype. The main focus of this work is on VR modeling, and implementing this feature would require a lot of network programming, which does not fit within the scope of this work and would take considerable time away from more important features.

3.7 Additional feature ideas

This section presents ideas that are not part of the current design but could be theoretically useful and potentially considered for similar applications in the fture.

- **World map** - A panel featuring a top-down (ortographic) map of the entire scene, with the option to teleport to any location on the map for fast travel.
- **Drawing tool** - The ability to draw in the virtual space using a controller is a feature that many VR applications offer, and while it may not be necessary for blockout, some users may find it useful for level design.
- **Biome paint tool** - The ability to 'paint' surfaces such as grass, forests, fields, etc. by designating an area on objects or within a volumetric space where surface-specific objects (e.g. tree primitives for a forest surface) would be spawned could be helpful in populating the environment more quickly.
- **Custom camera view** - Placing a virtual camera at a custom position in the scene and providing a real-time preview on the UI panel, to allow the user to see how their build looks from different positions and angles.
- **Chain tool** - Instantiating objects along a predefined curve or line.
- **Figurine model** - A 3D model of a human character that can be placed in a scene for scale and perspective.
- **Spectator** - The ability to detach the camera on the PC screen of the running application and move it independently to watch the user in the application from an arbitrary viewing position.

Chapter 4

Implementation

The implementation chapter describes the process of creating a prototype VR application based on the research and design presented in the previous chapters. This application is referred to as "BlockoutVR" as a simple and concise reference to its purpose. It was primarily developed as for PCVR (application running on a PC connected to a headset) for easier testing, but it was implemented using standard VR techniques and should be possible to build for standalone devices as well.

The prototype will be released as open-source software under the MIT license, allowing the use of only self-made assets or assets with a permissive license that can be used under the selected MIT license. This eliminates the possibility of using any plugins from the Epic Marketplace, as Unreal Engine's EULA does not allow sharing plugins with people who have not signed the EULA, even if the plugin is released for free. In the end, two external plugins were used, both released under the MIT license.

BlockoutVR was implemented using the alpha version of Unreal Engine 5.0 and later migrated to the stable UE 5.0 version upon its release. Since then, a new version 5.1 has been released, but it did not bring any specific features that would have been useful for this work, so the application remains on version 5.0.

Performance was not a primary concern for the prototype application, but common VR performance issues were taken into account during implementation. The application did not experience any noticeable issues such as frame drops or slowdowns, in a scene with thousands of primitives and hundreds of them being interacted with simultaneously (as shown in Figure 4.1). This was tested on the following hardware: Meta Quest 2 (render target: full resolution 3664x1920, 90 fps), CPU Ryzen 5 3600, RAM 16Gb dual-channel DDR4-3600, and GPU AMD Radeon RX 5700 XT.

4.1 Implemented functional requirements

This work includes the development of a prototype for the ideal application outlined in the design chapter, featuring a reduced set of the features presented in the design chapter. The design chapter presented the functional requirements of the ideal application, and Figure 4.1 shows which of these

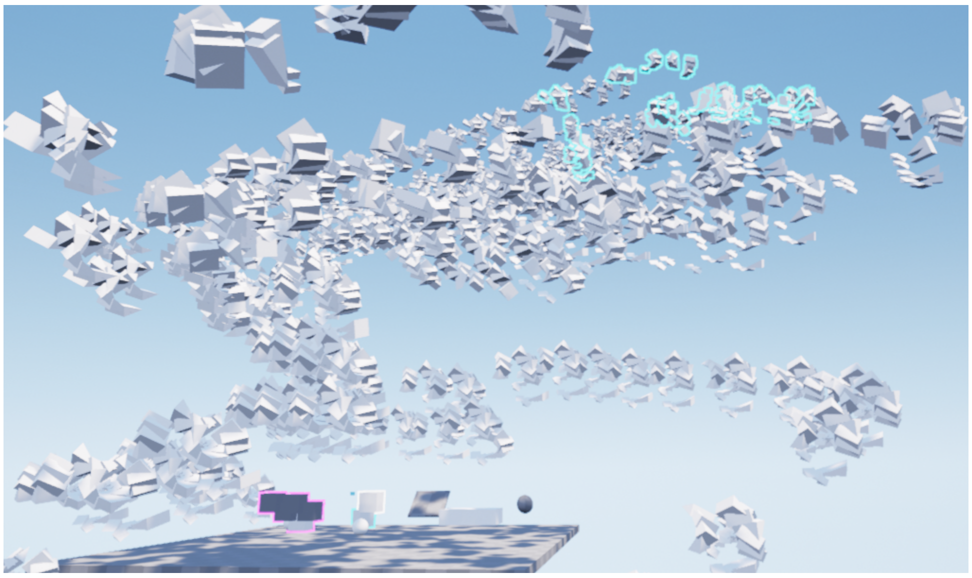


Figure 4.1: This scene in BlockoutVR features thousands of primitives (each ramp-shaped object is a single primitive).

have been implemented in the prototype.

Feature	Implemented
Intuitive user interface	Yes
Comfortable locomotion	Yes
Save/load	Yes
Importing models	Yes
Exporting scene	Yes
Undo/Redo object operations	No
Object manipulation tools	Yes
Geometry tools	No
Parenting objects	No
Group selection	Yes
Multiple modes	Partially
Snapping	No

Table 4.1: Overview of implemented features.

4.2 Unreal Engine

BlockoutVR is developed as a standalone application using Unreal Engine (UE). The original idea was to create the application in the form of a plugin for UE, which would have significant advantages in terms of combining the VR and 2D editor workflow and synchronizing between them. However, after researching this option, concerns arose and the decision was made to create an independent application and use UE only as an engine framework. This was mainly because the application needed to be unbiased towards cooperating programs (game engines or other modeling tools), and the plugin design would have locked it to use in the UE editor only. Extending integration to other engines would not have been possible in a reasonable manner.

Programming in the UE can be done using the general-purpose compiled language C++ or the built-in visual scripting system Blueprint [10]. In UE, the term "Actor" refers to any object that can be placed in a level. Actors contain variables, components, and logic (using Blueprint), similar to what classes do in C++. The term "Widget" refers to a visual UI element used to display things. Other specific terms should be self-explanatory or will be explained individually.

4.2.1 C++

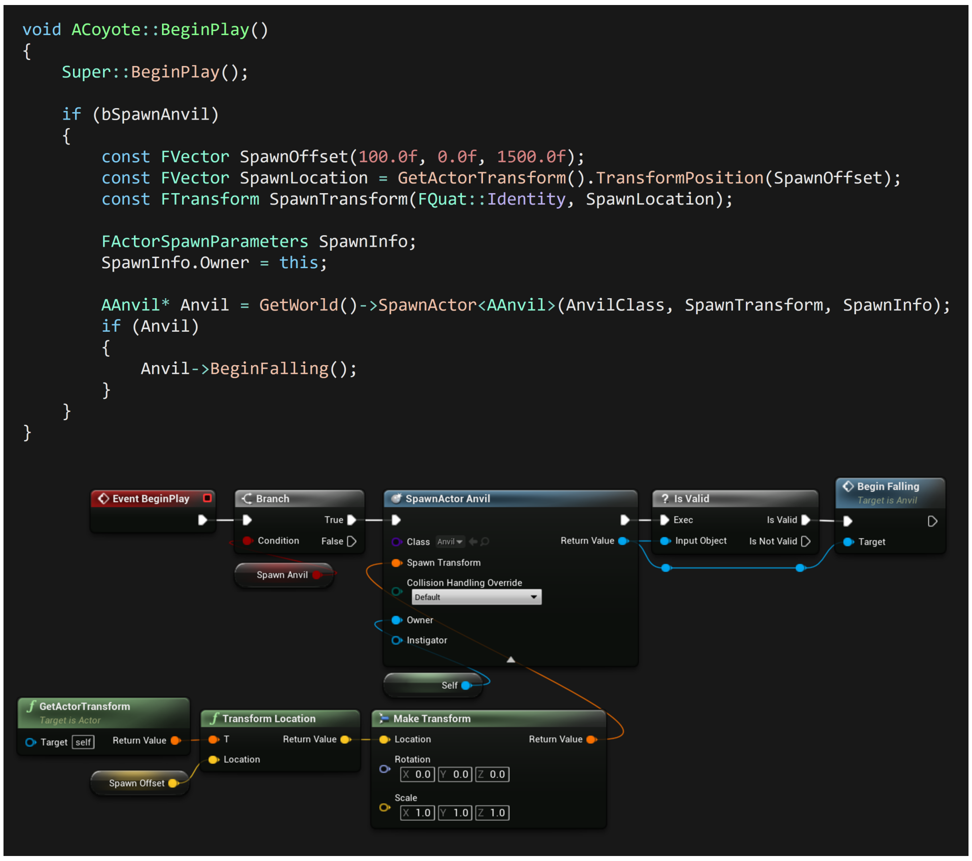
C++ is a standard, fast, compiled language with several advantages over a scripting language. The most apparent factor is performance; compiled languages tend to be faster in general. When comparing C++ with Blueprints, the performance difference depends heavily on the specific use case and cannot be accurately quantified. However, it is worth noting that there is a significant gap. For this reason, C++ is preferable for performance-critical tasks. C++ code is stored in a simple text form and can be merged and diffed with other versions of the code, which is not something that is easily possible with a visual script.

Unreal Engine itself is written in C++, so the language is closely integrated with the UE API, which means that all engine features are directly accessible from C++. UE also allows changes to the engine's code base if the released version is not suitable for a specific use case, but this can only be done using C++. Modifying the engine is a rather rare case, but there are more common actions that Blueprints simply cannot perform. Most notably, integrating any external libraries and tools in the project is often only possible with C++. Even for engine features, not everything is exposed to Blueprints and requires code, usually in fields where performance is critical and C++ is expected to be used. Common uses of C++ in UE include math functions, networking, communication with external programs, and integration of external libraries and tools.

In terms of their capabilities, Blueprints are a subset of C++ but are still very useful in many cases. While everything can be done in C++, it is generally more effective to prefer Blueprints over C++, except for the cases described in the previous paragraph.

4.2.2 Blueprint

Blueprints is a scripting language, as opposed to the compiled C++. The main advantage of this is a faster creation and iteration process, as small changes do not require re-compiling the code, which can be a slow and tedious process in Unreal, especially when making many small tweaks and changes while tuning any features. As a visual language, the flow of execution is clearly visible



```cpp
void ACoyote::BeginPlay()
{
    Super::BeginPlay();

    if (bSpawnAnvil)
    {
        const FVector SpawnOffset(100.0f, 0.0f, 1500.0f);
        const FVector SpawnLocation = GetActorTransform().TransformPosition(SpawnOffset);
        const FTransform SpawnTransform(FQuat::Identity, SpawnLocation);

        FActorSpawnParameters SpawnInfo;
        SpawnInfo.Owner = this;

        AAnvil* Anvil = GetWorld()->SpawnActor<AAnvil>(AnvilClass, SpawnTransform, SpawnInfo);
        if (Anvil)
        {
            Anvil->BeginFalling();
        }
    }
}
```

Figure 4.2: A comparison of logically equivalent C++ code at the top and Blueprints script below [13].

and logical mistakes are more obvious. This allows even non-programmers to make some changes without understanding the code. Additionally, as a closed system relying on pin connections, Blueprints will not allow connections that do not make sense, such as passing an integer to a vector value, which can save time by preventing mistakes that would otherwise only be found later during compilation.

Since Blueprint is a visual language, programming is done using a series of nodes, sometimes called node-based programming. The appearance of Blueprints and their comparison to C++ are shown in Figure 4.2. Instead of writing code, custom logic and behavior is created by combining a set of pre-defined nodes and functions. These nodes and functions are organized into a flowchart-like structure, which makes it easy to see the overall structure and behavior of your Blueprints. Programmers can drag, move, add, or remove nodes and connect them arbitrarily via input and output pins on each node. Each node represents a different action, condition, or operation, and the wires connecting them define the flow of execution. The Unreal editor provides a debugger for real-time previewing of values, states, and flow at every step of the execution of the Blueprints.

Custom nodes can be created in both Blueprints and C++, which allows for seamless cooperation between the two languages. A common workflow is to program computationally heavy functions in C++ as Blueprint nodes and then call them from the Blueprints. This way, both languages can be used together to their best advantage. Another approach is to create a Blueprint class that inherits from a C++ class and use the C++ class as the core, allowing Blueprints to be built on top of it.

Blueprints are great for game logic, control, and general workflow, basically everything outside of the special cases described in the C++ section above. For this reason, this work relies heavily on Blueprints.

4.3 Plugins

4.3.1 VRExpansion plugin

Unreal Engine offers an interface for VR, but it does not have many common features implemented; only a simple VR template is provided by Epic Games. This is why it seemed useful to start with the VRExpansionPlugin (VRE) [46]. VRE is a plugin that provides common VR features, such as locomotion, general VR interaction, networking, and more. After spending a considerable amount of time programming and using the plugin, it turned out that using it for BlockoutVR was not as useful as predicted. Most of the features provided by the plugin went unused, and the needed functionalities were not provided in a way that they could be effectively used. The features only that BlockoutVR does use are teleportation (but modified) and laser interactions.

Still, there is merit in using the plugin for possible future extensions of this work, especially features like multiplayer, which would be easier to implement using this plugin as most of the necessary tools are already included. BlockoutVR is based on the VRE example project, which is why there are parts of the code/blueprints from the VRE plugin in the project. Most of these parts were left in the project, even if they are unused, for the possible expansion of features.

4.3.2 RuntimeMeshLoader plugin

The RuntimeMeshLoader [22] is a simple plugin that utilizes the Assimp library to read common 3D model formats into Unreal Engine. The original plugin was loading the models into UE as procedural meshes, which had some issues, mostly related to performance and different interactions. Therefore, the plugin was modified to load static meshes, which were more suitable for the BlockoutVR use case.

4.4 BlockoutVR architecture

As a project based on the VRE example project, I tried to separate my assets from the ones provided by the VRE plugin as much as possible, but it was not

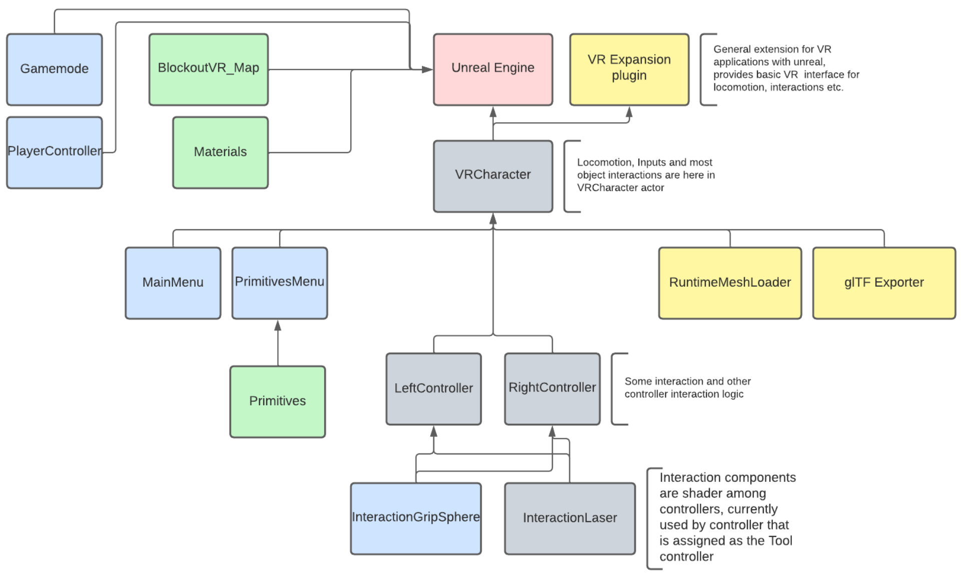

Gamemode
PlayerController
BlockoutVR_Map
Materials
Unreal Engine
VR Expansion plugin
General extension for VR applications with unreal, provides basic VR interface for locomotion, interactions etc.
VRCharacter
Locomotion, Inputs and most object interactions are here in VRCharacter actor
MainMenu
PrimitivesMenu
RuntimeMeshLoader
glTF Exporter
Primitives
LeftController
RightController
Some interaction and other controller interaction logic
InteractionGripSphere
InteractionLaser
Interaction components are shader among controllers, currently used by controller that is assigned as the Tool controller

Figure 4.3: Program architecture. Blue nodes represent Blueprint entities, grey nodes are Blueprint entities with a C++ base class, green nodes are assets, and yellow nodes are plugins. Note that this is an abstract description; some nodes can represent multiple entities or an entity with a different name (as the names here are meant to convey the function of the node).

completely possible for some of them that were used as a base for BlockoutVR actors. For simple separation, the logic (C++/blueprints) that is not directly used can be regarded as not done by me, and the one that is used and not done by me will be pointed out in most cases. Below, some specific assets will be described, but for precise separation, the BlockoutVR project can be compared with the original VRE 5.0 example project [47].

VRCharacter is a modified character actor, the name of the asset in the project is BlockoutVR_VRCharacter. This actor is the centerpiece of the BlockoutVR application and contains most of its logic and connections to other actors and components. VRCharacter is based on the Character from the VRE plugin, so it also contains quite a lot of VRE's logic. The input system described in section 4.5.2 is there to provide inputs. This actor contains both VR controllers as its components (among a few other but rather unimportant components), so most of the tools' interaction logic is there. It also contains locomotion logic, except for teleportation, which is implemented in the TeleportController (see Figure 4.4, there is one for each controller) actor, which also provides laser interactions.

VR locomotion was one of the reasons for using the VRE plugin. Unfortunately, the main locomotion type of the HDM-based movement included in the plugin was locked on ground movement on the NavMesh, which is not usable in this case. Therefore, a new movement called Ghost Movement in the project was developed to fit the one described in design 3.4.1. Similarly, the vertical moves described in 3.4.1 had to be made. Teleport was used from the VRE plugin but had to be modified to fit the control scheme of

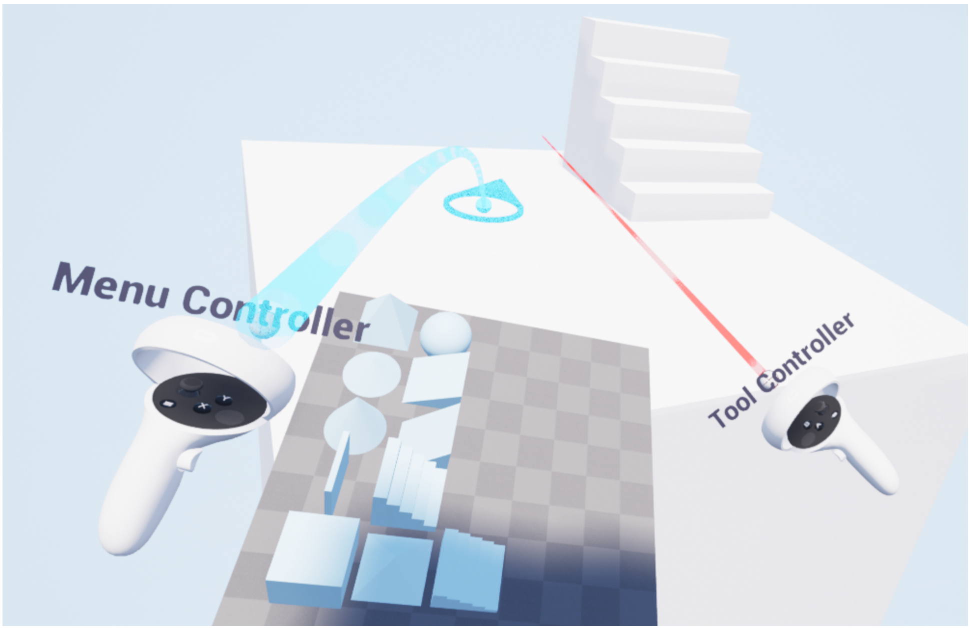


Figure 4.4: Figure showing teleport and laser functionalities provided by the TeleportController actor from the VRE plugin.

BlockoutVR, and the option to turn while teleporting was modified as well.

Importing and exporting are mostly done through plugins. Unreal provides a great toolset for exporting glTF from the application [11]. In version 5.0, it is provided as a built-in plugin, which is already part of the Unreal download package but needs to be manually enabled. From version 5.1, these functionalities are built into the engine directly. This allows for relatively easy exporting of glTF models from the application, it just needs the correct settings and provided objects to export. Importing was solved through the modification of a different plugin (see section 4.3.2).

Both the Main menu (a direct component) and the Primitives menu (a controller-attached component) are part of the VRCharacter actor.

4.5 Interfaces

4.5.1 VR headsets

VR headsets use software runtimes to provide necessary support and resources to connect to and interact with a computer, and enable the headset to display video and track the space. Historically, VR hardware manufacturers provided their own software for their headsets. Fortunately, in 2019, the Khronos group released the stable version of the OpenXR open API standard for VR [25], which is supported by multiple relevant VR headset providers.

The application was developed and tested on the Quest 2, so it is guaranteed to work properly on this headset. It is expected that other Meta devices using the same inputs should also work. In addition, Vive, Windows Mixed Reality, and Valve Index should work, but these devices have not been tested and no guarantees can be given. However, for the best accessibility, BlockoutVR uses the OpenXR API, so it should be compatible with all standard headsets. Some inputs may not work properly and may require adjustments to the Unreal input mapping to function as intended, but these should be simple and fast tweaks.

4.5.2 Input and controllers

One of the implemented features is the ability to swap the Menu and Tool controllers as an accessibility feature for left-handed users. Swapping controller inputs during runtime is a non-trivial task in Unreal. In fact, UE5 provides a plugin specifically for "complex input handling or runtime control remapping, and backward compatibility with UE4 default input system" [12]. However, after some research and testing, it seemed that this plugin was more complex (especially for VR with multiple different hardware inputs) than was needed, both in terms of provided features and setting up the inputs. As a result, I opted for a more straightforward solution. I mapped all inputs directly to the buttons, but kept a flag for which hand is the Menu controller. I also made custom events for each input used in the application and changed the flag to remap all custom input events to the other controller. When any input is used, the corresponding proper action event is called. This was still quite a lot of manual work, but it works properly and is easier compared to the other options.

4.5.3 User interfaces

The Main menu and other elements with text are implemented using Unreal Motion Graphics (UMG), a built-in visual scripting system for creating visual widgets and building user interfaces in UE. It provides interactive and highly customizable UI elements such as buttons, text fields, and sliders, among others, to create interfaces such as menus and bind these elements to the backend logic. The UMG widget `MainMenu` is set in the actor `Menu`, which is attached to the Menu controller. Then, there is the `Primitives menu`, a 3D menu for providing primitives to the user. The most important part of the interface are the controllers and their input elements, which are described in detail in section 3.2.1.

Chapter 5
Testing

This chapter describes the testing methodology. The people participating in the testing will be referred to as testers. Testers are expected to have experience with 2D modeling software, ideally with level blockout directly, but this is not required. Only BlockoutVR will be tested, and it will not be directly compared to the desktop application during the testing, as it is assumed that the testers have experience with the standard modeling software.

5.1 Testing scenarios

The testing process will involve two scenarios. The first one is a learning scenario, which involves going through all provided features and tools and building a very simple scene to familiarize the tester with the application. Every step will be accompanied by an instructor.

The second scenario will provide a 3D model of a small, level-like scene. The task will be to import the model and, using it as a reference, build a human-scale replica of the model from the provided primitives as precisely and efficiently as possible. Efficiency is defined as avoiding unnecessary actions, as the application supports using multiple tools at once, and this scenario is meant to encourage testers to use them in this way. An instructor no longer accompanies this scenario directly, but the testing is primarily for judging the efficiency of the workflow and overall experience.

Testers can be given hints if they are experiencing trouble related to the control of the application. However, hints can only be given for the controllers' inputs (as they are not directly marked in the prototype, and the tester can forget the layout). Hints cannot be given for controlling user interface elements (such as the main menu and primitives menu) as this is part of the testing.

5.2 Questionnaire

The testing ends with a questionnaire (see Table 5.1) to find out the testers' thoughts about the application. The questionnaire and previous observations will be evaluated to produce the final testing output. Any discrepancies or contradictions between the observations of the user and their responses on the questionnaire can be used to judge the tester's reliability. Additionally, any other feedback from testers is noted for evaluation, especially if there are any concerns or positive surprises that the tester may have experienced and the questionnaire did not cover.

5.3 Testing results

Due to organizational reasons and after consulting with my supervisor, testing will be carried out after the submission of the thesis. The results of the testing will be presented at the thesis defense.

BlockoutVR is user friendly.				
Strongly agree	agree	neutral	disagree	strongly disagree
My workflow in BlockoutVR is efficient.				
Strongly agree	agree	neutral	disagree	strongly disagree
I would consider using BlockoutVR for level blockout instead of a desktop program.				
Strongly agree	agree	neutral	disagree	strongly disagree
I believe that using VR for work in 3D can be advantageous compared to desktop applications with a 2D interface.				
Strongly agree	agree	neutral	disagree	strongly disagree
I am missing some features.				
Strongly agree	agree	neutral	disagree	strongly disagree
(if disagree/strongly disagree in previous answer) The missing features are among the designed features.				
Strongly agree	agree	neutral	disagree	strongly disagree
(if disagree/strongly disagree in previous answer) What specific features are you missing?				
How would you rate BlockoutVR user interface?				
Strongly positive	positive	neutral	negative	strongly negative
How would you rate your experience with BlockoutVR?				
Strongly positive	positive	neutral	negative	strongly negative

Table 5.1: Questionnaire for the testing. Using a Likert scale with 5 options.

Chapter 6
Conclusion

6.1 Summary

The goal of this work was to design an ideal world sketching/level blockout application for virtual reality, implement a prototype, and test its viability compared to standard desktop applications. Therefore, initial research began with finding all relevant papers, which provided valuable information, especially the reactions of users from conducted testing. However, this alone did not provide enough information to design the entire application. The other important data came from other existing applications focused on any part of 3D modeling. Over ten applications were tested and analyzed, seven of which were discussed in detail as they provided valuable data. The features, functionalities, and user interfaces found in these applications were used to shape the design into a complete set necessary for an ideal application of this focus.

Since virtual reality has a very different interface than a standard computer, additional issues had to be considered. Meta Quest guidelines proved to be an invaluable source of information for general VR application design, especially to avoid any pitfalls such as nausea-inducing factors that many VR applications suffer from. With these three types of sources, a meaningful design of such an application was successfully created and described in detail. The design seemed promising, but a real application had to be created for testing purposes. A prototype application called BlockoutVR was implemented using the essential parts of the design as a template. Some other ideas, such as integration with game engines, were considered and discussed.

Based on the research contained in this thesis and my experience with BlockoutVR, I believe in the potential of the application, and its ability to be a better tool than its desktop counterparts, in some cases. However, to get an independent opinion and prove its usefulness, further testing will have to be carried out.

6.2 Future possibilities

The research of academic works and existing similar projects showed state-of-the-art VR techniques and elements that were used in designing this work, but as VR is constantly changing, there are likely new improvements that can be directly applied to this work and BlockoutVR. From the perspective of the application alone, there are many possibilities for future improvements. The most obvious is implementing the rest of the features presented in Table 4.1 of the currently implemented features, which is something I am considering continuing to work on even after finishing this thesis. In addition, there are ideas presented in section 3.7. However, two big features stand out and should be expressed.

The first one is the implementation of online multi-user sessions in the application. While not essential for level blockout, some of the tested applications provided this feature, and it could be interesting for collaboration during the blockout phase.

The other feature is live synchronization with game engines, as described in section 3.6.

Bibliography

[1] *Top 3D sculpting tools for virtual reality authoring | Medium by Adobe.* [Online; accessed 27. Nov. 2022]. Nov. 2022. URL: `https://www.adobe.com/products/medium.html`.

[2] Mike Alger. "Visual design methods for virtual reality". In: *Ravensbourne. http://aperturesciencellc. com/vr/VisualDesignMethodsforVR_MikeAlger. pdf* (2015), pp. 1–98.

[3] *Arkio - Collaborative Spatial Design.* [Online; accessed 27. Nov. 2022]. Nov. 2022. URL: `https://www.arkio.is`.

[4] Brian Baker. *Brian Baker on Twitter.* [Online; accessed 16. Nov. 2022]. Nov. 2022. URL: `https://twitter.com/JrBakerChee/status/1182384066881916928`.

[5] Lee Beever, Serban Pop, and Nigel W John. "LevelEd VR: A virtual reality level editor and workflow for virtual reality level design". In: *2020 IEEE Conference on Games (CoG)*. IEEE. 2020, pp. 136–143.

[6] Costas Boletsis. "The new era of virtual reality locomotion: A systematic literature review of techniques and a proposed typology". In: *Multimodal Technologies and Interaction* 1.4 (2017), p. 24.

[7] Evren Bozgeyikli et al. "Point & teleport locomotion technique for virtual reality". In: *Proceedings of the 2016 annual symposium on computer-human interaction in play.* 2016, pp. 205–216.

[8] Julián Conesa-Pastor and Manuel Contero. "EVM: An Educational Virtual Reality Modeling Tool; Evaluation Study with Freshman Engineering Students". In: *Applied Sciences* 12.1 (2022), p. 390.

[9] Josen Daniel O De Leon et al. "Genesys: A virtual reality scene builder". In: *2016 IEEE Region 10 Conference (TENCON)*. IEEE. 2016, pp. 3708–3711.

[10] *Blueprints Visual Scripting.* [Online; accessed 8. Dec. 2022]. Nov. 2022. URL: `https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine`.

[11] *Exporting Unreal Engine Content to glTF.* [Online; accessed 31. Dec. 2022]. Dec. 2022. URL: `https://docs.unrealengine.com/5.1/en-US/exporting-unreal-engine-content-to-gltf`.

[12] *Enhanced Input.* [Online; accessed 31. Dec. 2022]. Dec. 2022. URL: `https://docs.unrealengine.com/5.0/en-US/enhanced-input-in-unreal-engine`.

[13] Alex Forsythe. *Blueprints vs. C++, How They Fit Together and Why You Should Use Both.* [Online; accessed 25. Dec. 2022]. Mar. 2021. URL: `https://awforsythe.com/unreal/blueprints_vs_cpp`.

[14] Max Glenister. *The User Experience of Virtual Reality.* [Online; accessed 6. Mar. 2022]. Mar. 2022. URL: `https://www.uxofvr.com`.

[15] *Blocks - Create 3D models in VR - Google VR.* [Online; accessed 26. Nov. 2022]. Nov. 2022. URL: `https://arvr.google.com/blocks`.

[16] *Tilt Brush by Google.* [Online; accessed 27. Nov. 2022]. Nov. 2022. URL: `https://www.tiltbrush.com`.

[17] *Gravity Sketch | 3D design and modelling software.* [Online; accessed 26. Nov. 2022]. Apr. 2022. URL: `https://www.gravitysketch.com`.

[18] MP Jacob Habgood et al. "Rapid, continuous movement between nodes as an accessible virtual reality locomotion technique". In: *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*. IEEE. 2018, pp. 371–378.

[19] Pearl Hogbash. *Pearl "ManWolfAxeBoss" Hogbash on Twitter.* [Online; accessed 16. Nov. 2022]. Nov. 2022. URL: `https://twitter.com/Pearl_Hogbash/status/1583165252362829825`.

[20] *Advanced Framework - VR, Mobile & Desktop.* [Online; accessed 2. Jan. 2023]. Jan. 2023. URL: `https://www.unrealengine.com/marketplace/en-US/product/advanced-vr-framework?sessionInvalidated=true`.

[21] Jason Jerald et al. "Makevr: A 3d world-building interface". In: *2013 IEEE Symposium on 3D User Interfaces (3DUI)*. IEEE. 2013, pp. 197–198.

[22] Christopher Jürges. *RuntimeMeshLoader.* [Online; accessed 8. Jan. 2023]. Jan. 2023. URL: `https://github.com/Chrizey91/RuntimeMeshLoader`.

[23] Julian Keil et al. "Effects of virtual reality locomotion techniques on distance estimations". In: *ISPRS International Journal of Geo-Information* 10.3 (2021), p. 150.

[24] *glTF - Runtime 3D Asset Delivery.* [Online; accessed 29. Nov. 2022]. Dec. 2020. URL: `https://www.khronos.org/gltf`.

[25] *Khronos Releases OpenXR 1.0 Specification Establishing a Foundation for the AR and VR Ecosystem.* [Online; accessed 31. Dec. 2022]. July 2019. URL: `https://www.khronos.org/news/press/khronos-releases-openxr-1.0-specification-establishing-a-foundation-for-the-ar-and-vr-ecosystem`.

[26] Eugenia M Kolasinski. *Simulator sickness in virtual environments.* Vol. 1027. US Army Research Institute for the Behavioral and Social Sciences, 1995.

[27] Eike Langbehn et al. “Application of redirected walking in room-scale VR”. In: *2017 IEEE Virtual Reality (VR)*. IEEE. 2017, pp. 449–450.

[28] Michael LaRocco. “Developing the ‘best practices’ of virtual reality design: Industry standards at the frontier of emerging media”. In: *Journal of Visual Culture* 19.1 (2020), pp. 96–111.

[29] Joseph J LaViola Jr. “A discussion of cybersickness in virtual environments”. In: *ACM Sigchi Bulletin* 32.1 (2000), pp. 47–56.

[30] Marcello Lorusso, Marco Rossoni, and Giorgio Colombo. “Conceptual modeling in product design within virtual reality environments”. In: *Computer-Aided Design & Applications, 18(2)* (2021), pp. 383–398.

[31] *Rendering | Oculus Developers*. [Online; accessed 4. Dec. 2022]. Dec. 2022. URL: `https://developer.oculus.com/resources/bp-rendering`.

[32] *User Orientation and Positional Tracking | Oculus Developers*. [Online; accessed 4. Dec. 2022]. Dec. 2022. URL: `https://developer.oculus.com/resources/bp-orientation-tracking`.

[33] *VR Accessibility Design: Controller Mapping, Input, and Feedback | Oculus Developers*. [Online; accessed 4. Dec. 2022]. Dec. 2022. URL: `https://developer.oculus.com/resources/bp-rendering/#visually-represent-controller-inputs-with-button-highlights`.

[34] Meta/Oculus. *Comfort and Usability | Oculus Developers*. [Online; accessed 7. Nov. 2022]. Nov. 2022. URL: `https://developer.oculus.com/resources/locomotion-comfort-usability`.

[35] Meta/Oculus. *Meta Quest 2: Immersive All-In-One VR Headset | Meta Store*. [Online; accessed 1. Jan. 2023]. Jan. 2023. URL: `https://www.meta.com/quest/products/quest-2`.

[36] Meta/Oculus. *Resources*. [Online; accessed 13. Mar. 2022]. Mar. 2022. URL: `https://developer.oculus.com/resources`.

[37] Meta/Oculus. *Set up Guardian for Meta Quest | Meta Store*. [Online; accessed 26. Dec. 2022]. Dec. 2022. URL: `https://www.meta.com/help/quest/articles/in-vr-experiences/oculus-features/oculus-guardian`.

[38] Meta/Oculus. *Vision | Oculus Developers*. [Online; accessed 26. Nov. 2022]. Nov. 2022. URL: `https://developer.oculus.com/resources/bp-vision`.

[39] *Microsoft Maquette on Steam*. [Online; accessed 6. Mar. 2022]. Mar. 2022. URL: `https://store.steampowered.com/app/967490/Microsoft_Maquette`.

[40] *Neos Metaverse*. [Online; accessed 28. Nov. 2022]. Nov. 2022. URL: `https://neos.com`.

[41] Patrick Owens. *Oculus Controller Diagram | Figma Community*. [Online; accessed 5. Jan. 2023]. Jan. 2023. URL: `https://www.figma.com/community/file/990323039387125706`.

[42] Anjul Patney et al. "Towards foveated rendering for gaze-tracked virtual reality". In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), pp. 1–12.

[43] *Mesh Boolean — PyMesh 0.2.1 documentation.* [Online; accessed 27. Dec. 2022]. Jan. 2021. URL: `https://pymesh.readthedocs.io/en/latest/mesh_boolean.html`.

[44] *ShapesXR—VR Creation and Collaboration Platform for Remote Teams.* [Online; accessed 27. Nov. 2022]. Nov. 2022. URL: `https://www.shapesxr.com`.

[45] *Creation Tools.* [Online; accessed 26. Nov. 2022]. Nov. 2022. URL: `https://design.sketchbox3d.com`.

[46] Morden Tral. *VR Expansion Plugin – A Virtual Reality Tool Kit.* [Online; accessed 9. Dec. 2022]. Dec. 2022. URL: `https://vreue4.com`.

[47] Morden Tral. *VRExpansionPlugin.* [Online; accessed 8. Jan. 2023]. Jan. 2023. URL: `https://github.com/mordentral/VRExpansionPlugin/tree/5.0-Locked`.

[48] *Tvori – Design and Collaboration Platform to Prototype Interfaces, Products, and Experiences, Previzes, Animatics or VR films.* [Online; accessed 26. Nov. 2022]. Aug. 2022. URL: `https://tvori.co`.

[49] *Quest 2 Controllers: Full Specification - VRcompare.* [Online; accessed 19. Nov. 2022]. Oct. 2020. URL: `https://vr-compare.com/accessory/quest2controllers`.

[50] Robert Yang. *Blockout.* [Online; accessed 27. Mar. 2022]. Mar. 2022. URL: `https://book.leveldesignbook.com/process/blockout`.

Appendix A
Application Manual

Upon application start, the user is automatically in the empty scene with all tools ready.

The manual describes the default situation where the Menu controller is the left controller, and the Tool controller is the right controller. On the left controller, the thumbstick controls direct movement in the direction of the HDM. The left trigger invokes teleporting, and on release, the user will teleport to the position if it is a valid position. While holding the left trigger, the left thumbstick can control the rotation of the user after the teleport. When the user is touching the left thumbstick, holding the left trigger moves the user vertically up, and holding the grip moves the user down.

As for the right/Tool controller, the right trigger is the main interaction button for holding/moving focused and selected objects. Focused is an object that the user aims the laser at or is within range of the gripping sphere, which is depicted as a small sphere in front of the controller. The interactions differ based on whether the hold is done by the grip sphere or laser. Both interactions are invoked with the trigger, with the grip sphere having priority. Holding an object with the grip sphere simulates a real-life hold of the object, allowing it to be arbitrarily moved and rotated. A laser hold does not rotate the object but rather changes its location to the end of the laser. However, the distance of the laser hold object can be increased or decreased by pushing the right thumbstick up or down, respectively. Then there is button A for selection. Holding A will select (and visually highlight) every object that is either pointed at with the laser or within range of the grip sphere. One press of the A button over a selected object will remove it from the selection. A fast double press of the A button will deselect all selected objects. When multiple objects are selected, the tools affect all of them. Button B is for deleting objects, either only the focused object or multiple objects from the selection. When holding any objects, moving the right thumbstick to the left or right will decrease or increase size, respectively. Lastly, there is the right grip button for making copies of objects. To use this tool, either hold any object and press the right grip to leave a copy in the place where the held object was at that moment. Alternatively, hold the right grip and use the right trigger to hold any object, which will leave the object in the scene, and pull a new copy to be held.

The Primitives menu is a 3D panel with primitives attached to the left controller. The right controller can pull primitives from the panel into the scene. Primitives can be pulled either by the grip sphere hold or the laser hold, based on which interactor was interacting with the primitive at the moment the trigger was pressed.

The user interface is a menu above the left controller. The Y button can close and open the Main menu, so it doesn't get in the way when not necessary. The menu has three tabs: the first tab, Menu, contains a button for switching controllers, which swaps all functionalities between the controllers described above. The middle tab, Model, contains buttons for importing a model. The application expects to find models in the path "..../Documents/RuntimeMeshLoader/" with each model in its own folder. Then, the application automatically shows the models available for import in the menu. Objects should be in a common 3D format, preferably in FBX. The right tab, Save, is for saving, loading, restarting, and exporting the level.

Appendix B
Application Samples

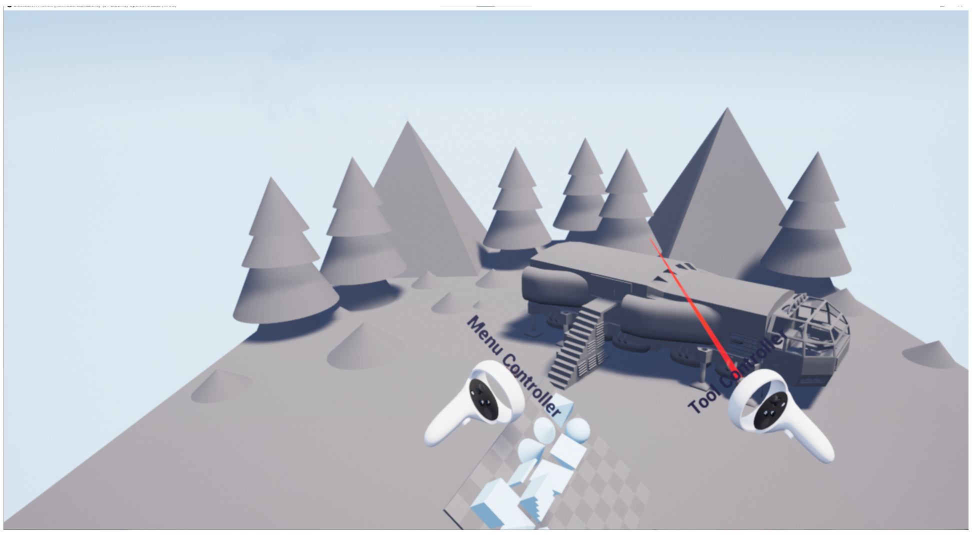


Figure B.1: BlockoutVR - A standard scene with controllers and a hidden main menu.

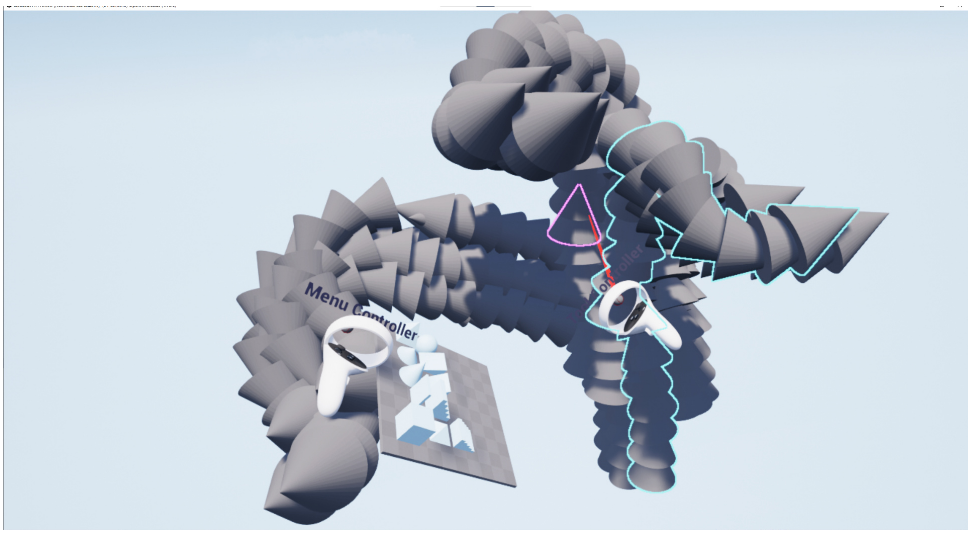


Figure B.2: BlockoutVR - Features a random creation of cones and highlighted selection (cyan) and focused object (violet).

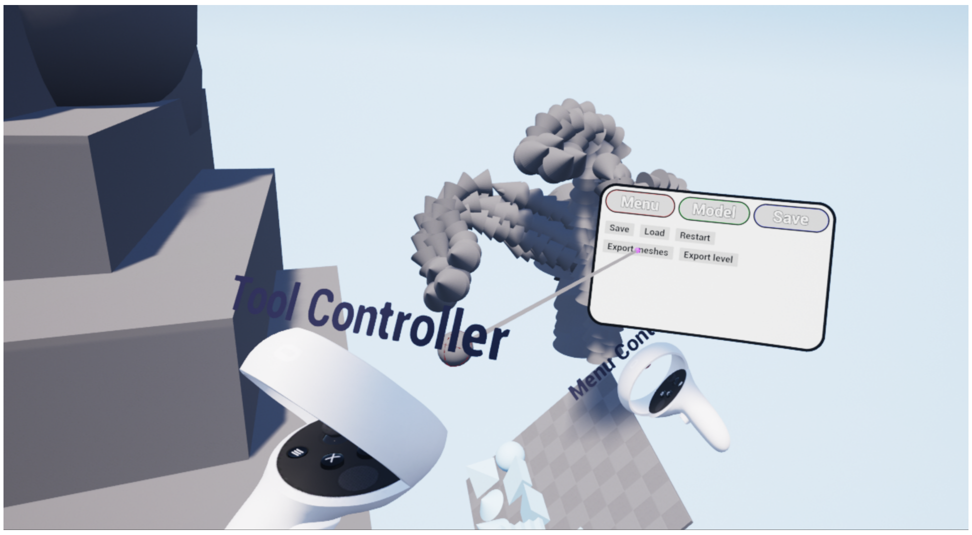


Figure B.3: BlockoutVR - Save menu on swapped controllers.

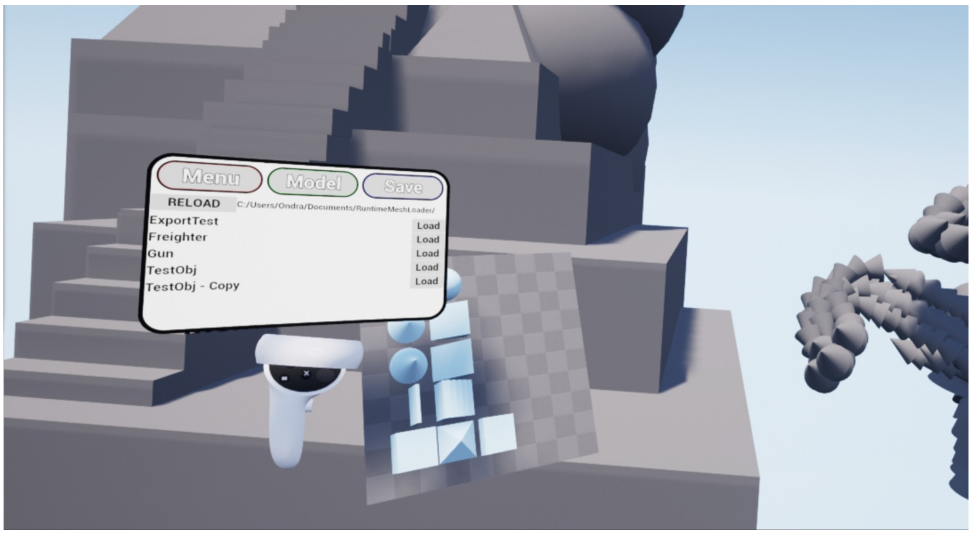


Figure B.4: BlockoutVR - Features a controller with the import model menu with another creation from primitives in the background.

Appendix C

File Attachments

`readme.txt`	the file with SD card contents
`thesis.pdf`	the thesis text in PDF format
`thesis_src`	LaTeX source files for this thesis
`BlockoutVR`	BlockoutVR project folder
`BlockoutVR.uproject`	Unreal engine project file
`Content`	main folder with content
`Plugins`	folder containing a plugins
`LICENSE.txt`	license
…	other files and folders necessary for UE project