Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

студента **Худолія Андрія Сергійовича**
(ПІБ)

академічної групи 123-20-1
(шифр)

спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)

за освітньо-професійною програмою 123 Комп'ютерна інженерія
(офіційна назва)

на тему «Створення цифрової моделі навчальної лабораторії кіберфізичних систем за допомогою VR технологій»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Гнатушенко В.В.			
спеціальної частини	проф. Гнатушенко В.В.			
розділів:				
розробка апаратної частини	доц. Ткаченко С.М.			

Рецензент				

Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:

завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)
« 27 » липня 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавр

студента Худолія А.С. **академічної групи** 123-20-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія
за освітньо-професійною програмою 123 Комп'ютерна інженерія
(офіційна назва)

на тему «Створення цифрової моделі навчальної лабораторії кіберфізичних систем за допомогою VR технологій»

затверджену наказом ректора НТУ «Дніпровська політехніка» від 29.04.2024 № 375-с

Розділ	**Зміст**	**Термін виконання**
Стан питання та постановка завдання	Аналіз ринку на схожі проекти та пошук найкращих практик розробки	
Розробка 3д моделі лабораторії	Створення безпосередньо детальну 3д модель лабораторії в програмному забезпеченню Blender	
Розробка VR середовища	Створення інтерактивного VR середовища з використанням моделі лабораторії	

Завдання видано _____ проф. Гнатушенко В.В.
(підпис керівника) (прізвище, ініціали)

Дата видачі 01.05.2024

Дата подання до екзаменаційної комісії 01.07.2024

Прийнято до виконання _____ Худолій А.С
(підпис студента) (прізвище, ініціали)

ABSTRACT

Explanatory note: 64 pages, 13 sources, 71 images.

Object of research: CREATION OF DIGITAL MODEL OF AN EDUCATIONAL LABORATORY OF CYBER-PHYSICAL SYSTEMS USING VR TECHNOLOGIES

This bachelor's thesis focuses on the development of a digital model of an educational laboratory for cyber-physical systems utilizing Virtual Reality (VR) technologies. The research emphasizes creating an interactive and immersive educational environment that leverages the advanced capabilities of VR to enhance learning experiences. The study explores the integration of VR into educational settings, detailing the process from conceptualization to implementation. Key aspects include the design of VR environments, the development of interactive 3D models using Blender, and the deployment of these models in a VR setting. This project not only demonstrates the potential of VR in education but also addresses practical challenges such as hardware requirements and software configurations. The findings suggest that VR technology can significantly enrich educational methodologies, providing a more engaging and effective learning platform.

Keywords: VR, 3D MODELLING, C#, UNITY, TEXTURING, BLENDER, VR INTERACTION FRAMEWORK, SHADER NODE, UV-UNWRAP, VFX, SHADER GRAPH.

LIST OF TERMS, SYMBOLS, ABBREVIATIONS AND TERMS

VR - Virtual Reality

3D models - 3 dimensional models

C# - object-oriented programming language

GIT - distributed version control system

Shading - applying materials(or colors) for objects.

UV-unwrap - is the process of 'unfolding' a mesh so that you can create a 2D texture which fits the 3D object.

URP - Universal Render Pipeline

HDRP - High Definition Render Pipeline

SUMMARY

This bachelor's thesis explores the integration of Virtual Reality (VR) technologies in educational settings, specifically within a cyber-physical systems laboratory at the Dnipro University of Technology. The initiative focuses on transcending the limitations of traditional educational laboratories, where logistical and safety constraints often hinder learning potential.

The core aim of the project is to facilitate a dynamic and interactive learning environment through VR. This environment enables students to manipulate complex systems and machinery in a risk-free virtual space, enhancing both the safety and educational value of laboratory interactions. Students are afforded opportunities to engage with educational content in ways that are not feasible in a conventional laboratory setting, promoting an active learning experience.

The development process of the VR laboratory involved detailed 3D modeling of the lab equipment and environment using Blender software, ensuring realistic and interactive virtual models. These models were then integrated into a VR platform using the Unity Engine, renowned for its robust capabilities in creating immersive virtual experiences.

Throughout the project, several challenges were addressed, including optimizing VR software to run smoothly on diverse hardware setups and ensuring the VR environment was user-friendly and accessible. These challenges involved not only technical adjustments but also a deep analysis of how such interactions influence educational outcomes.

Findings from this research indicate that VR can significantly enhance educational methodologies by making learning more engaging and effective. The immersive nature of VR allows for a more intuitive and hands-on approach to understanding complex theories and principles.

TABLE OF CONTEST

INTRODUCTION

The integration of Virtual Reality (VR) technology in educational environments represents a transformative advancement in how instructional content is delivered and experienced. This thesis presents a comprehensive examination of the application of VR in cyber-physical laboratory settings, focusing specifically on the creation of a digital model of an educational laboratory. This project is being carried out with the support of the Dnipro University of Technology.

The project aims to enhance traditional educational paradigms by introducing an immersive VR environment where students can interact with complex cyber-physical systems in a controlled, virtual space. This approach not only mitigates the logistical and safety challenges associated with physical laboratories but also enables a dynamic learning experience that is both scalable and adaptable to various educational needs.

Through the use of VR technologies, the project explores the development of detailed 3D models and interactive elements that mimic real-world laboratory equipment and scenarios. This digital transformation is facilitated by the use of Blender for 3D modeling and the Unity Engine for VR integration, ensuring a seamless and realistic user experience. The ultimate goal of this thesis is to demonstrate the efficacy of VR in improving student engagement and learning outcomes, thereby paving the way for broader adoption of VR-based educational tools in the future.

1. THE STATE OF THE ISSUE AND THE STATEMENT OF THE TASK

1.1. Characteristics of Virtual Reality (VR) Technologies, Their Capabilities, Usage in Education, Trends, and Development Prospects

1.1.1. Characteristics of Virtual Reality

Virtual Reality (VR) refers to a simulated experience that can either closely resemble the real world or differ significantly from it. This technology commonly utilizes a headset to immerse users in a three-dimensional, computer-generated environment. A key feature of VR is immersion, which makes users feel as though they are physically present in the virtual space. Additionally, VR allows for interaction with the environment and virtual objects using tools like controllers, hand tracking, or voice commands. To maintain the illusion of presence, the VR system must render visuals in real-time, which requires significant computational power. Advanced VR systems also provide sensory feedback, such as haptic responses to simulate touch, and some even offer experiences related to smell and taste[2].

1.1.2. Capabilities of VR Technologies

The capabilities of Virtual Reality (VR) are vast and continue to grow with ongoing technological advancements. One notable capability is simulation and training, where VR can replicate real-world environments and scenarios. This is especially useful in sectors such as aviation, military, and healthcare, where immersive, hands-on practice is crucial. Additionally, VR excels in visualization, allowing complex data and concepts to be portrayed in three dimensions, which can greatly enhance understanding and analysis. It also supports remote collaboration by enabling teams to work together in a shared virtual space, improving communication and teamwork across distances. In the realm of entertainment, VR provides immersive gaming experiences and allows for virtual tours of museums, historical sites, and other attractions, offering unique and engaging ways to explore new places and experiences[7].

1.1.3. Usage of VR in Education

The use of Virtual Reality (VR) in education has demonstrated significant potential to enhance learning outcomes. VR creates immersive learning environments where students can virtually explore historical sites, travel through space, or delve into the human body. This makes abstract concepts more tangible and engaging, transforming how information is consumed and understood. Additionally, VR enables interactive labs, where students can conduct experiments and perform complex procedures in a virtual, risk-free setting. This not only provides a safe learning environment but also reduces costs associated with physical lab setups[3].

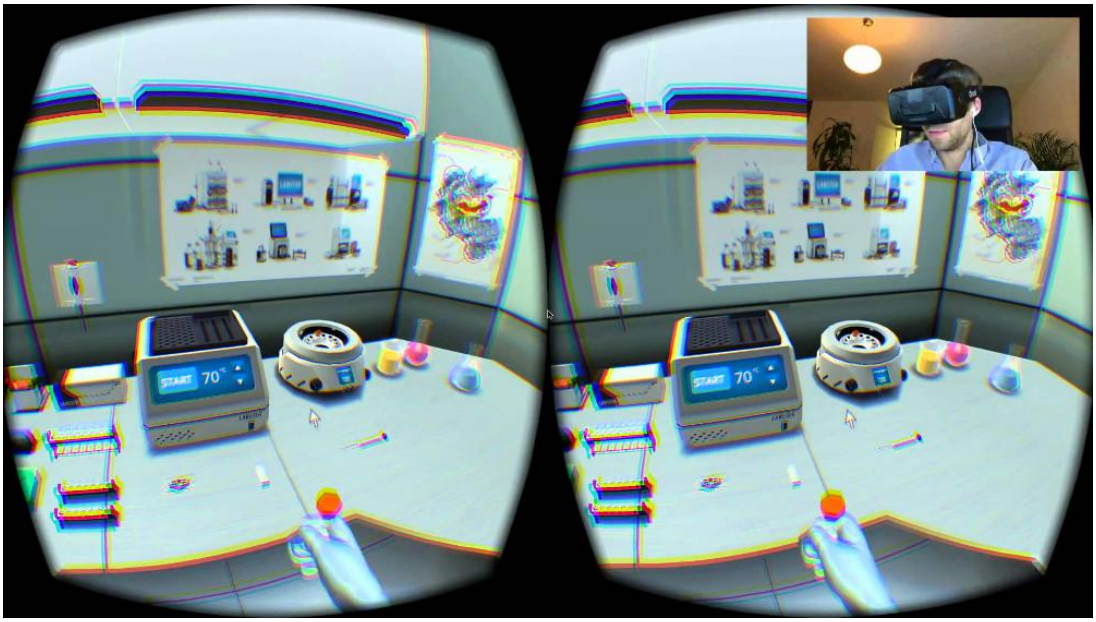

Image 1.1 – VR app is using as a physical-experiments simulation

VR also plays a crucial role in skills training by simulating real-world tasks. This allows students in fields such as surgery, engineering, and the arts to practice and hone their skills without the risks and costs typically involved in physical training. Furthermore, VR offers remarkable benefits in special education by enabling personalized learning experiences. It allows for the creation of customized educational content that meets the unique needs of students with disabilities, making learning more accessible and effective for everyone[6].

1.1.4. Trends in VR Technologies

The Virtual Reality (VR) landscape is constantly evolving, fueled by technological progress and creative uses. A key development is the emergence of standalone VR headsets, such as the Oculus Quest, which do not require connection to a PC or console. This advancement makes VR more accessible and user-friendly. Social VR is also gaining traction with platforms like VRChat and AltspaceVR that facilitate virtual social interactions, offering new ways for people to socialize and network.

Additionally, there is a growing emphasis on developing VR content specifically for educational purposes. This includes virtual classrooms and educational games that enrich learning experiences. In the corporate world, businesses are increasingly utilizing VR for training, product design, and virtual meetings, recognizing the advantages in productivity and efficiency it brings.

Moreover, the integration of Augmented Reality (AR) with VR to create Mixed Reality (MR) environments is an emerging trend. These MR environments blend physical and digital elements seamlessly, enhancing the user experience by making it more immersive and interactive. This combination promises to open up even more innovative applications and opportunities within the VR domain.

1.1.5. Development Prospects of VR Technologies

The future of VR technology looks incredibly promising, driven by several key areas of development. Improved hardware, such as advances in display technology, reduced latency, and enhanced processing power, are set to make VR experiences more realistic and comfortable. As the cost of VR hardware continues to decrease, its adoption is expected to expand across educational institutions and businesses, which will lead to wider usage and acceptance of the technology. Additionally, tools and platforms for creating VR content are becoming more sophisticated and user-friendly, allowing educators and developers to produce high-quality VR experiences more efficiently. VR's versatility will also see it finding applications in various fields including medicine, architecture, and tourism, showcasing its broad impact. Furthermore, ongoing research and innovation in VR technologies are pushing the boundaries even further, exploring new

ways to leverage VR for human benefit and enhancing our interaction with digital environments.

1.2. Examination of Similar Projects: Advantages, Disadvantages, and Differences. Analysis of Cyber-Physical Laboratories

In Ukraine, apparently there are no really similar projects to the one being made in this thesis. Sometimes idea comes close, but production is different in some way.

For example, in Ternopil National Technical University was created a project with a virtual museum of Ivan Pulyui[12][Image 1.3]. Moreover, they provided a separate models of physic experiments[Image 1.2]. It is displayed in 3D by using service Sketchfab, what gives an opportunity to rotate, scale an object. Also there are displayed some additional information about the parts.

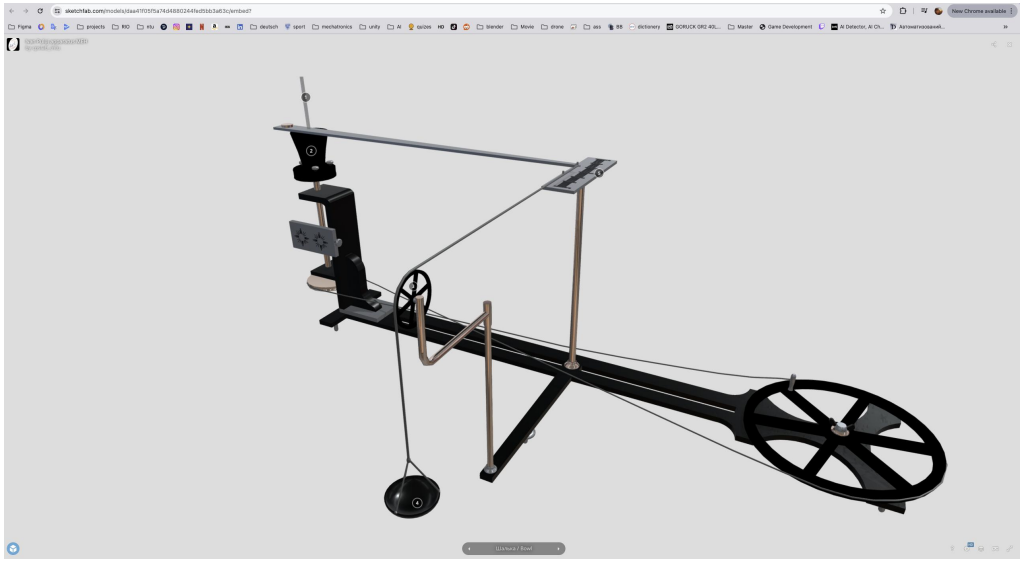

Image 1.2 – 3D model of an experiment

The Cyber-Physical Systems Laboratory at Ternopil Ivan Puluj National Technical University, established in November 2018, is a collaborative academic unit focusing on complex systems that integrate cybernetic components with physical processes for data acquisition, processing, secure storage, and transmission. Led by Assoc. Prof. O. Kramar, the lab collaborates with IT companies like ELEKS and Dataengi and has produced research on augmented reality for cybersecurity training and digitalization in agriculture. The lab also engages in developing demonstration equipment for promoting scientific and technological advancements.

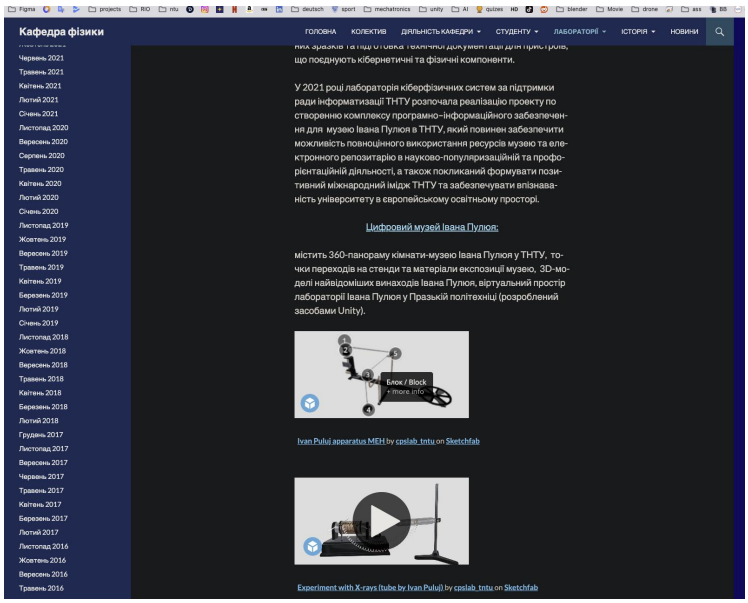

Image 1.3 – Museum website

But implementation of museum is made as google-street like interaction. It's just a standing camera with a 360 view. This doesn't allow to interact with anything in the room except buttons which were created specifically. Despite this, it is a very solid project which gives user tons of useful information about the provided person.

Image 1.4 – 360 interactive view on musem room

There also is a research about how interactive technologies can impact and be used for study purposes, written by professors from Kryvyi Rih National University[Image 1.5]. The research paper discusses the use of immersive technologies, such as virtual reality

(VR) and augmented reality (AR), in the education of future engineers. The study focuses on the integration of these digital tools into higher education to enhance the learning experience for engineering students by providing them with a highly realistic simulation of industrial environments. The paper argues that VR and AR can significantly improve the quality of education by helping students understand complex concepts and processes in a more interactive and engaging manner. The research highlights the successful implementation of these technologies in various educational settings, demonstrating their effectiveness and safety. The study also emphasizes the need for training educators to use immersive technologies effectively in their teaching practices.

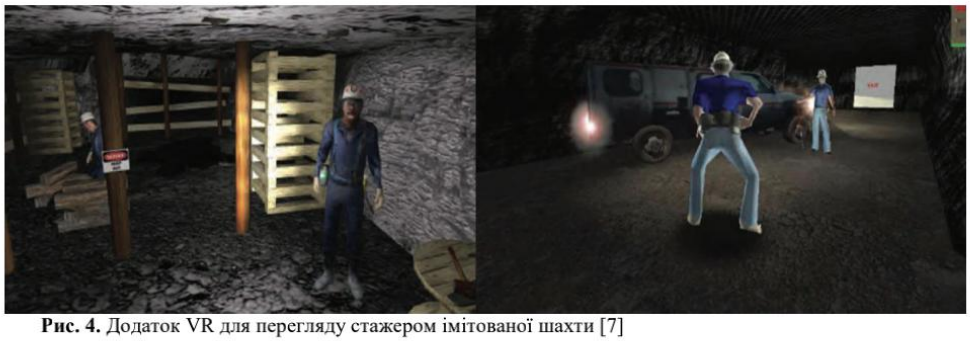

Рис. 4. Додаток VR для перегляду стажером імітованої шахти [7]

Досвід навчальних закладів та підприємств Китаю полягає у використанні VR для навчання безпеки аварійно-рятувальних робіт. Науковці [8; 9] розробили хмарну систему VR для навчання інженерів, яка включає апаратне забезпечення VR, панорамну систему відображення на основі проекцій, VR окуляри, дисплей, планшет та інші пристрої (рис. 5).

Рис. 5. Віртуальна система навчання та експериментальної лабораторії Китайського університету гірничої справи та технологій [9]

Image 1.5 – All available positions

Another research was made by Elif Damla Karakolcu about best practices of making a digital twin of an existing room/lab, called “BioRevolution #7 - Digital Twins in the Lab”[11].

The article discusses the transformative role of digital twins in laboratory settings, emphasizing their utility in enhancing efficiency, accuracy, and time management within scientific research. By simulating physical experiments virtually, digital twins allow for precise prediction and testing of outcomes, streamlining the scientific discovery process. They integrate with laboratory information management systems to ensure seamless data flow and maintain the integrity of simulations. The piece also highlights emerging startups that are innovating in this space, leveraging digital twins to optimize various scientific and industrial processes.

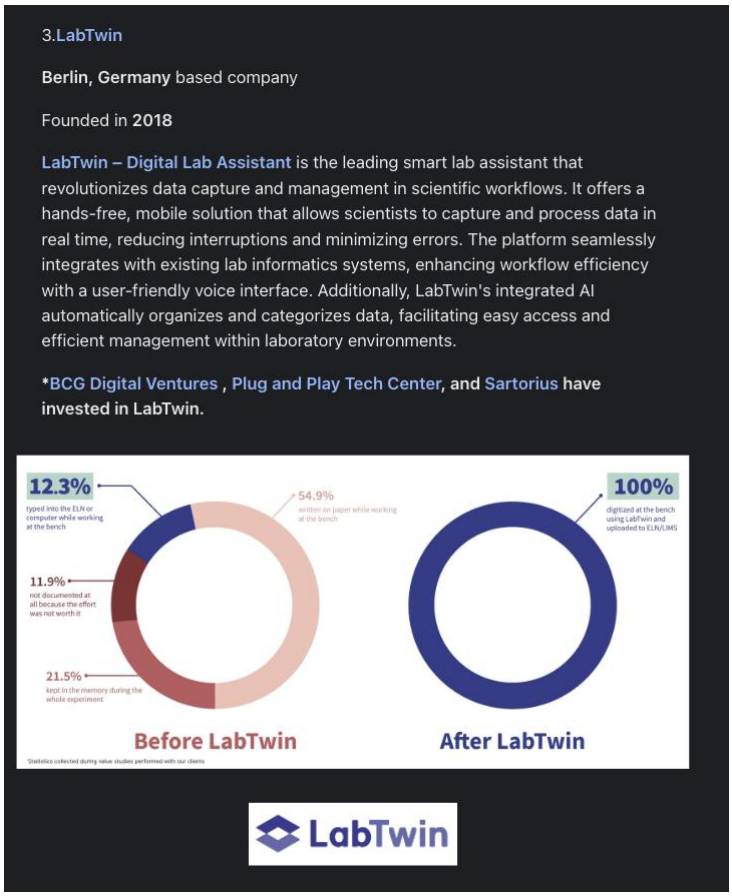

Image 1.6 – All available positions

In the end it’s clearly visible that such VR laboratory experience wasn’t made yet in Ukraine. Unique, one-of-a-kind project.

1.3. Task Formulation: Defining Objectives, Goals, and Positive Aspects of the Project

The goal of the thesis is implement a real-life laboratory in full-scale VR environment, using available modern technologies.

The main objectives are:

- Creating a good-quality model of the lab room itself and of all objects inside
- Making experience as interactive, as possible, what means that user should be able to interact with almost every object in the project.
- Deploy current project to Meta's Quest platform to be able to play natively and to spread the project easier.
- Make it as user-frendly as possible by providing an interactive tutorial how to use the project and to interact with objects.
- Make it interesting and fun to play. Beside the learning it should be a place of just enjoying the proccess of using VR.

A positive aspect for sure will be expanding the procedures of the educational process in the university. The virtual environment adds a completely new bubble where users can play, develop, and learn freely without worrying about all health protection stuff. Furthermore, it opens completely new doors in a way of education, because from now on it will not be needed to come to the lab on-site but rather work remotely. First of all, it is going to help students from the occupied regions and on the other hand it will be possible to host events for other professors.

1.4. Conclusions for the Chapter

The analysis of Virtual Reality (VR) technologies in the first chapter demonstrates that VR is a rapidly evolving field with significant potential in educational applications. VR technologies offer immersive and interactive experiences that can greatly enhance the learning process by providing realistic simulations and hands-on experiences that are difficult to achieve through traditional methods. The exploration of current trends and capabilities of VR technologies highlights their increasing accessibility and effectiveness, which are essential for modern educational practices. The insights gained from this

analysis confirm that the adoption of VR in education can lead to more engaging and effective learning environments.

2. DEVELOPMENT OF THE 3D MODEL TWIN OF THE LABORATORY

2.1. Digitization of the Laboratory: Methods of Photo Data Collection

For the image references was used a website of 360 Google-like tour of the laboratory. This website can be reached by this url: https://nmu.org.ua/ua/rooms3d/room-5-31.php This gives a great overall view of the lab, it's possible to see general sizing of all items in the room and great for initial room prototyping. The website provides a bunch of locations from different angles in the room, so it's straightforward to understand the scale of the room from all perspectives.

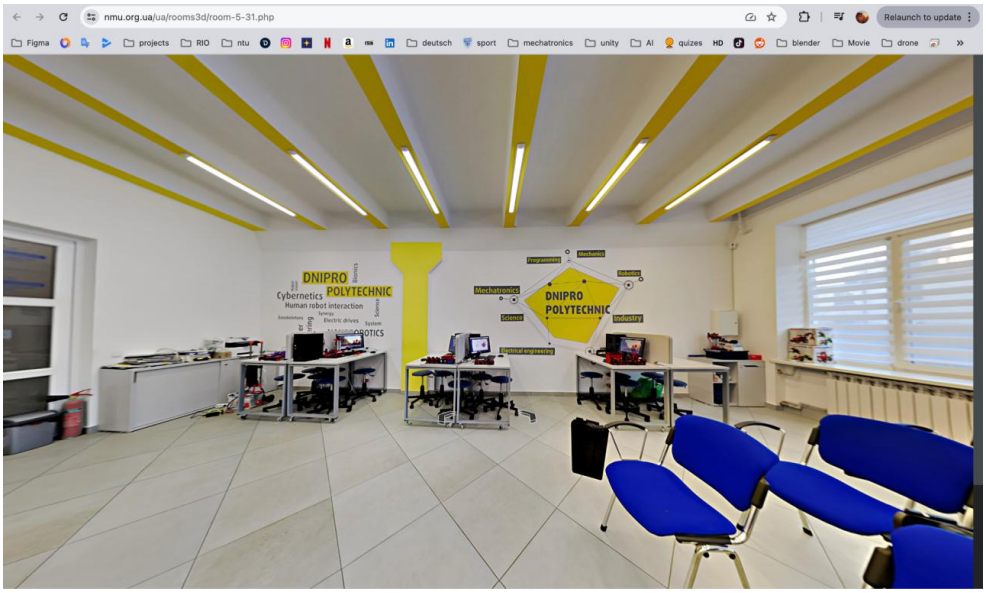

Image 2.1 – View of the 360-tour lab website

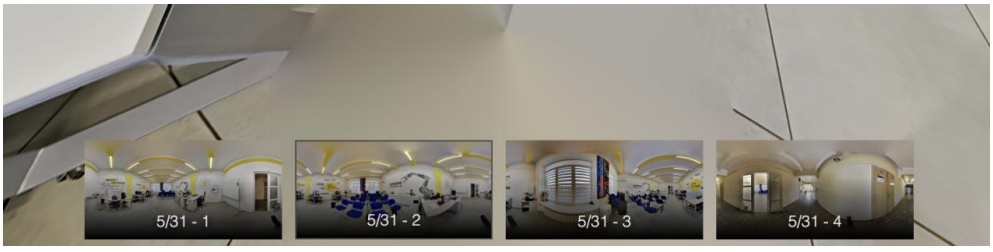

Image 2.2 – All available positions

Later on, for more detailed modeling of particular models, individual photos of these objects should be taken. From 3 axes: top, front, left - gives a precise enough look at the object, to be able to model it with enough detailing. Ideally, these photos should be taken without any tilt and be perfectly parallel to the object because this will lead to a better result.

2.2. Creation of 3D Models Using Blender Software: Advantages, Capabilities, and Development Process

Blender is an open-source, professional-grade 3D modeling software widely used in various industries for creating 3D models, animations, and visual effects. Its robust feature set and flexibility make it an ideal choice for creating detailed 3D models for Virtual Reality (VR) environments. This section will cover the advantages, capabilities, and development process of using Blender for 3D model creation.

2.2.1. Advantages of Using Blender

Blender is a free software, so there are no fees to use it, making it available for students, educators, and professionals[Image 2.3]. As an open-source program, it benefits from ongoing improvements and personalization by a large community of developers. Blender provides a comprehensive set of tools for tasks like modeling, texturing, sculpting, rigging, animating, rendering, and compositing. It also includes advanced features like particle systems, and simulations for fluids, smoke, and physics-based rendering. Blender works on Windows, macOS, and Linux, which makes it widely accessible. It supports various file formats, helping you work smoothly with other software. You can also customize Blender with Python scripting to automate tasks and create your own tools and add-ons. Plus, a vast library of add-ons enhances Blender's capabilities to meet specific needs and workflows.

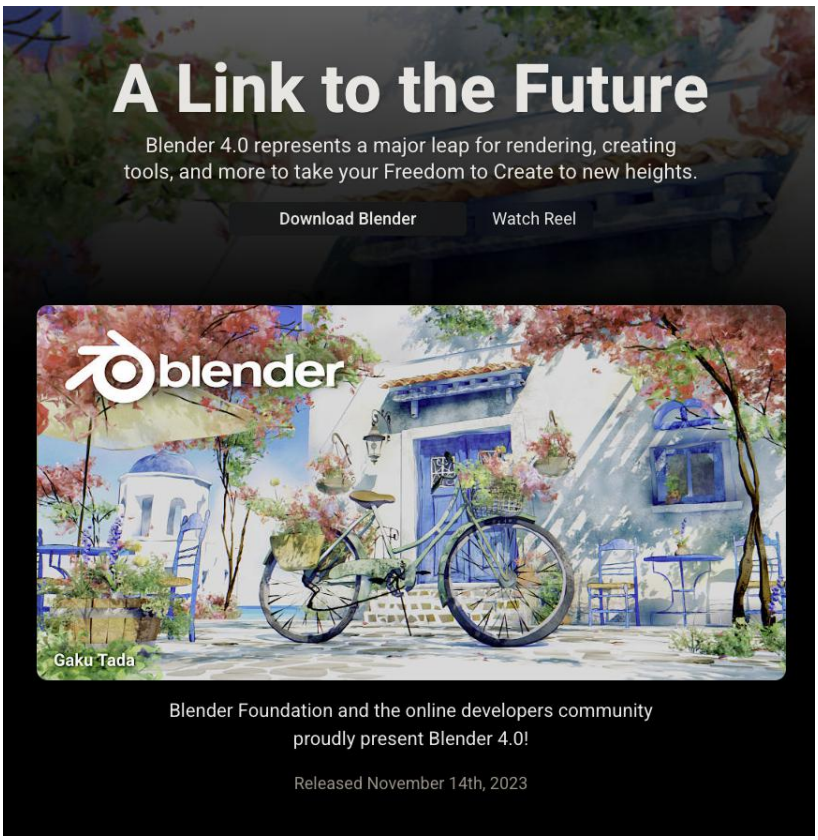

Image 2.3 – All available positions

2.2.2. Capabilities of Blender

Blender's modeling tools offer a range of features including basic shapes, modifiers, and mesh editing options, allowing you to create complex and intricate 3D models. It includes advanced techniques like sculpting, retopology, and procedural modeling which give you more flexibility and control during the modeling process[Image 2.4]. For texturing, Blender supports painting directly on textures, UV mapping, and procedural textures, which help you create realistic and detailed surfaces. The shader editor uses a node-based system to build complex materials, improving the visual quality of models. Blender’s animation capabilities feature keyframing, skeletal systems, and motion capture integration, suitable for animating both characters and objects.

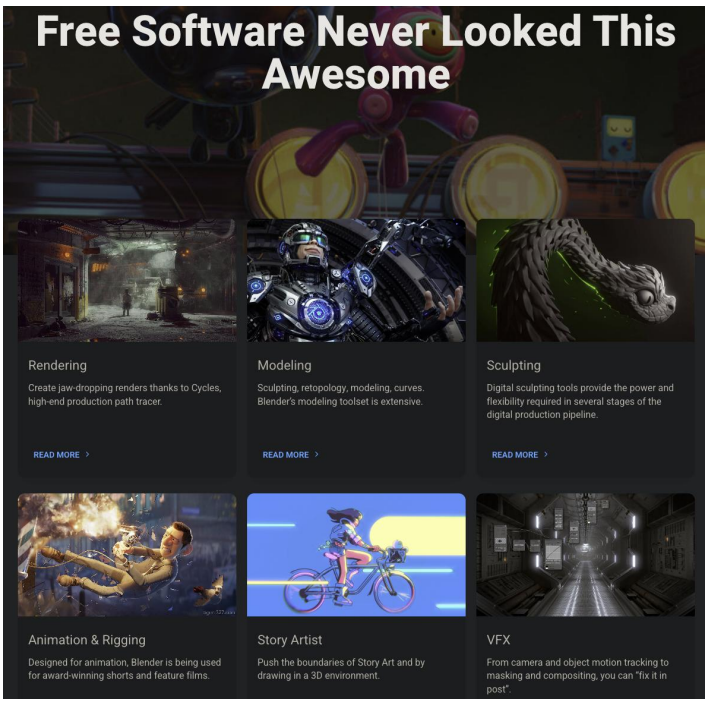

Image 2.4 – All features of Blender

Blender comes with two powerful rendering engines: Cycles, which is a path-tracing engine for high-quality results, and Eevee, which is a real-time engine for quicker previews. These engines support various effects related to lighting, shading, and cameras, making your renders more realistic and visually appealing.

Additionally, Blender's physics engine includes simulations for rigid and soft bodies, cloth, fluids, smoke, and particles, adding dynamic realism to your models and scenes. These simulations can be incorporated into animations to create natural interactions and movements.

2.2.3. Developing process

The modeling started with a simple blackout[Image 2.5]. This strategy helps set proportions up right from the start. Although it is easy to see what should be paid more attention later on.

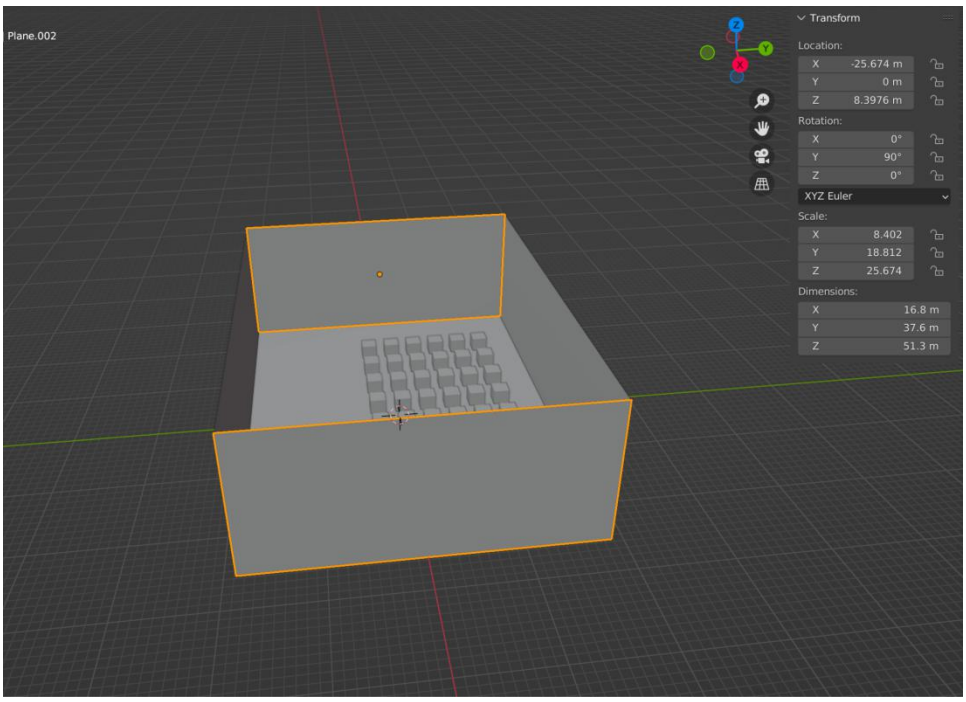

Image 2.5 – Blackout model of the lab

Then, it is possible to proceed to add some big details. In Image 2.6 it's illustrated that the room appeared to have windows, detailed ceilings, walls, and some blocks for future objects.

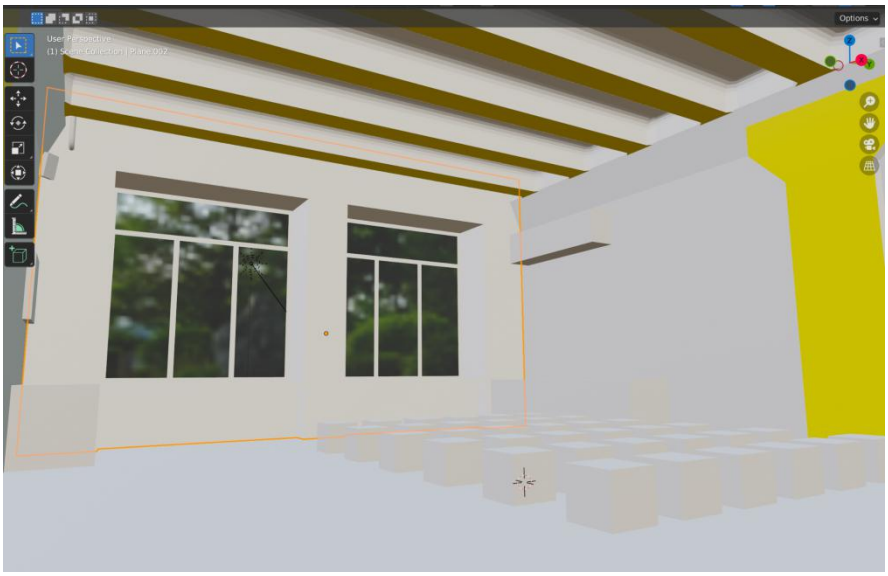

Image 2.6 – Room with ready wall boundaries

Most importantly, everything should be done step-by-step, because otherwise, work will turn into a mess. There are just to many details to replicate in the room, so it won't be possible to finish a model fast enough with proper planning. It goes from biggest to the smallest, it's the same for painters - they start with a few big colors and then start adding more and more details to it.

The following Image 2.7 shows a room with the most important details. At first sight, it's not even visible that half of the objects are still simple cubes. It works because the composition was built correctly and now the only thing left is to add more and more smaller details to the model.

Image 2.7 – Scratch model of the lab

By the end of modelling, shading and painting, model will take its final look[Image 2.8]. When two images put into comparison, it's pretty easy to see all differences which they are having. Starting from details, there are simply more objects which you can find in real room: pcs, Fischertechnik boxes[13], their models and so on. The most important touch is lighting and materials - they contribute the most for giving such a good look.

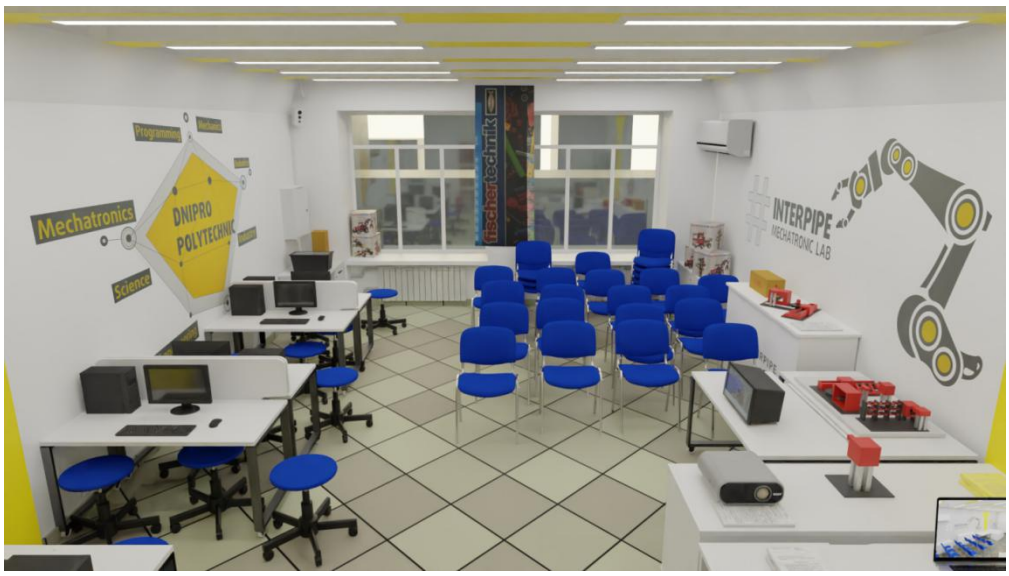

Image 2.8 – Final look of the room model

Now on, let's take in-detail look of the modeling process of two objects - the Cabinet and one low-poly Fischertechnik's model[13]. Most of the modeling starts from simple cube[Image 2.9]. Then, in edit Edit Mode, by using simple vertex interactions, they were extruded, shrinked, scaled and moved to form a blackout model of the target object[Image 2.10].

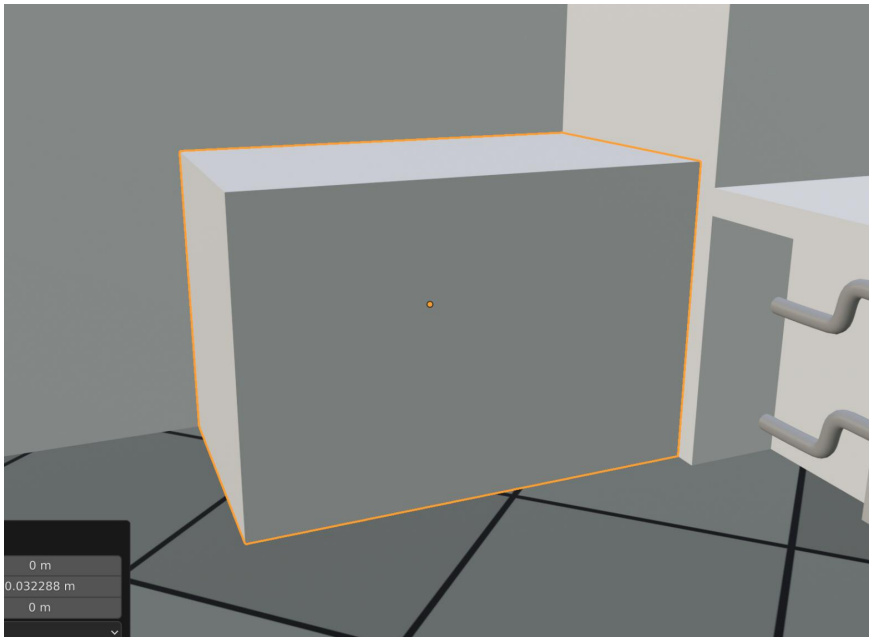

Image 2.9 – Start form of the Cabinet

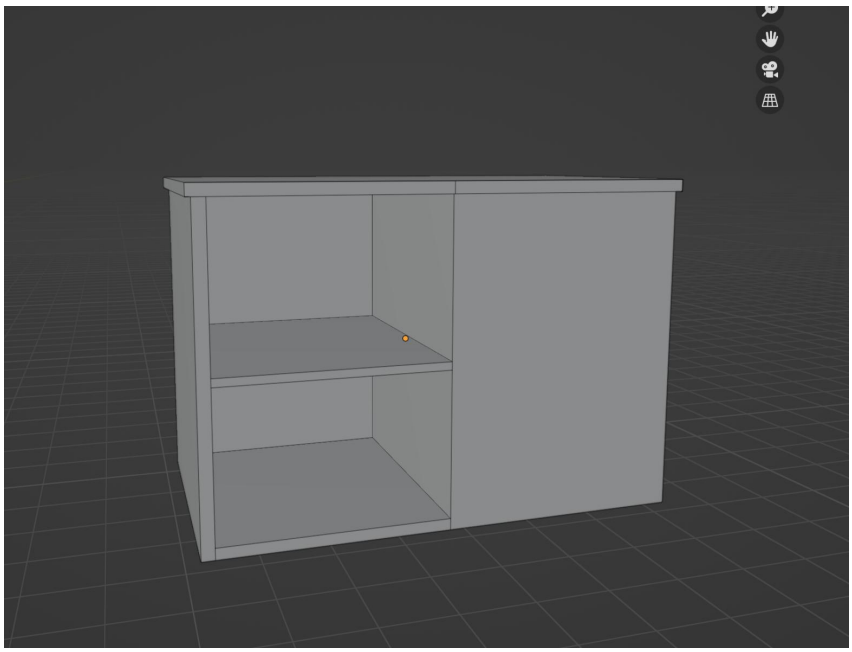

Image 2.10 – Blackout stage of the model

And now it’s possible to procede adding neat details, such as curves for the door, keyhole, space inside of the object. Additionally, after applying a proper material for the object, it’s possible to say that it is ready to use.

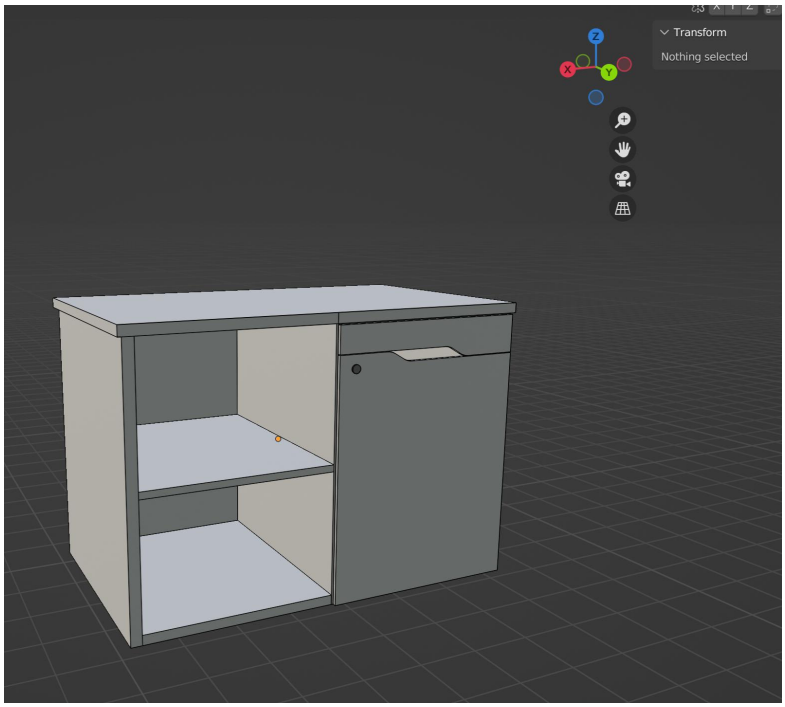

Image 2.11 – Final look of the Cabinet

Let’s proceed to the second model - Fischertechnick’s Fabrik 4.0[13]. The first step was to find at least one reference image[Image 2.12].

Image 2.12 – All available positions

By following the exact same pattern, the red model is being made. By starting making an outline of the future object, it’s easy to see what and where is located, if everything is up to scale and if objects are on their own places.

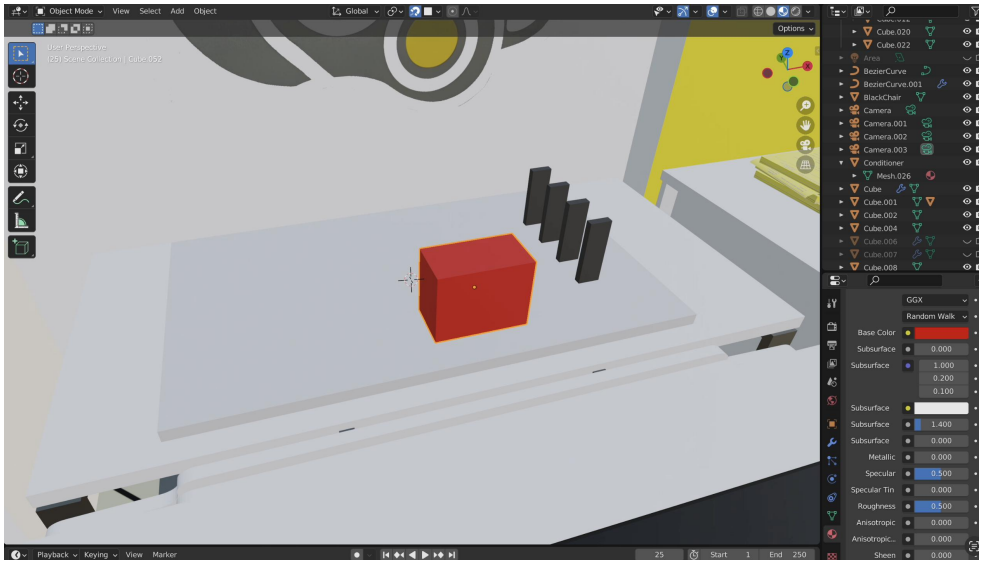

Image 2.13 – Simple squares to form overall picture of the model

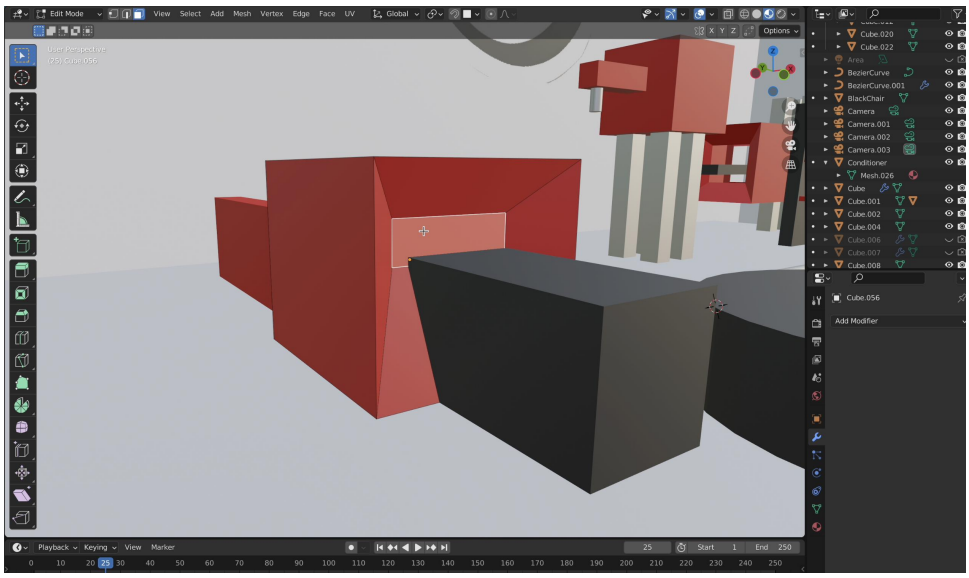

Image 2.14 – Editing by deforming meshes

During the mesh edit, using such tools as extruding, cutting a hole, moving merging verticies, it's becoming clear to see the final result[Image 2.15]. Including that's a VR project and not just a render - current level of detail is fair enough. Even though it would be cool to see all details of original object, unfortunately, effort-time graph is exponentially uneven. Means that it will take a lot of time to create a decent-level 3D model of a current reference object. Nevertheless, this object will become a great start point for future detail improvements.

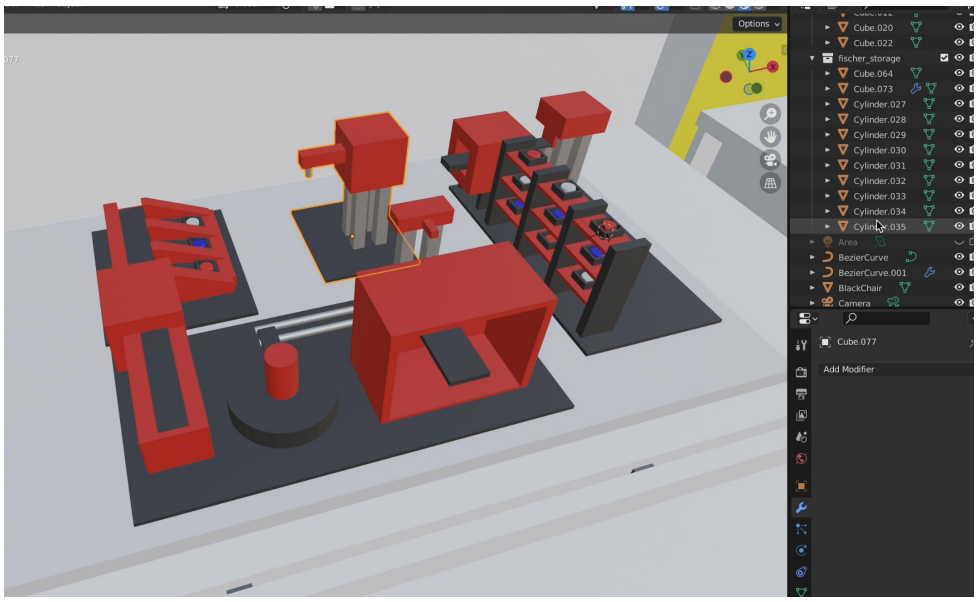

Image 2.15 – Finished low-poly model

2.3. Development of Materials for Objects: Parameters and Features

Creating realistic and engaging materials for 3D objects is a crucial aspect of developing virtual environments. Materials define the surface properties of 3D models, including their color, texture, reflectivity, and more. This section outlines the key parameters and features involved in the development of materials for objects in virtual reality (VR) applications.

The development of materials starts with defining their basic properties, such as color and texture. The base color, also known as the diffuse or albedo color, gives the material its primary hue, which can be a solid color or derived from a texture map. Textures are essential in adding details and realism to the surface of 3D models. Common texture maps include the diffuse map for base color, the normal map for simulating small surface details, the bump map for creating the illusion of depth, the specular or gloss map for defining shininess and reflectivity, and the ambient occlusion map for adding depth by simulating light interaction in surface crevices.

Reflectivity and shine are critical parameters for making materials look realistic. Specular highlights are the bright spots that appear on shiny surfaces when illuminated. The intensity and sharpness of these highlights can be adjusted to simulate different types of surfaces. Glossiness or roughness determines how smooth or matte the surface appears, with lower glossiness making it look matte and higher glossiness making it shiny.

Transparency and opacity are also important in material development. Opacity defines how transparent or opaque a material is, which can be controlled using opacity maps. Refraction, the bending of light as it passes through transparent materials, adds realism to materials like glass and water.

Emissive properties are used to make materials appear as if they are emitting light. This is useful for creating glowing objects such as screens or lights. Subsurface scattering (SSS) is essential for materials like skin, wax, and certain plastics, where light penetrates the surface, scatters within, and exits at different points, giving a soft, realistic appearance.

The metallic parameter determines whether a material behaves like a metal or a non-metal. Metals have high reflectivity and specific light absorption characteristics, while non-metals rely more on diffuse reflection.

Physically-Based Rendering (PBR) is a modern approach to material creation that simulates the physical properties of surfaces more accurately. PBR materials use parameters such as albedo, metallic, roughness, and normal maps to achieve realistic results, ensuring consistent material appearance under different lighting conditions, which is ideal for VR applications.

Shader nodes are used to create complex materials by connecting different functional nodes in a node-based editor. Blender's shader editor allows users to build intricate materials using a combination of diffuse, specular, and other shader nodes. Texture baking involves pre-calculating complex details and lighting effects and storing them in texture maps, which reduces the computational load during real-time rendering and improves performance in VR applications.

Material libraries can speed up the development process by providing ready-to-use materials that can be customized. Many software applications and online repositories offer extensive libraries of PBR materials, which can be a valuable resource for quickly achieving realistic and high-quality surfaces in VR projects.

Optimizing materials for VR is essential to ensure smooth performance without compromising visual quality. This involves balancing texture resolution, minimizing the number of texture maps, and using efficient shaders. Level of detail (LOD) techniques can be employed to reduce the complexity of materials based on the viewer’s distance from the object, ensuring optimal performance.

The workflow for developing materials involves several steps. It starts with gathering references and real-world examples to understand the characteristics and behaviors of different surfaces. Creating base textures using software like Photoshop, Substance Painter, or directly within Blender is the next step. These textures should accurately represent the surface details, colors, and patterns.

Once the base textures are ready, material parameters are defined in the 3D software by assigning the base color, normal, roughness, and other texture maps. The parameters are adjusted to achieve the desired look and feel. The material is then applied to the 3D model and tested under various lighting conditions to ensure it looks realistic and performs well in different scenarios. Optimization for VR involves reducing texture sizes,

minimizing shader complexity, and ensuring efficient rendering. Finally, the material is exported in a suitable format for the target VR platform, ensuring all necessary assets, such as textures and animations, are included.

In conclusion, developing high-quality materials for 3D objects in VR applications involves understanding and manipulating various parameters and features to achieve realistic and immersive results. By leveraging advanced tools and techniques, developers can create materials that not only look realistic but also perform efficiently in VR environments, enhancing the overall user experience.

2.3.1. Painting process

Normally during the modelling objects stay gray or white, that's not the desired result. For example, the same room that was seen in Image 2.5, look, looks that at the moment completely grayish[Image 2.16].

To be able to create and assign correct materials for the objects, refer back to the lab's navigation website, where we can see the colors of all objects. Furthemore, not only colors should match, but also other parameters, such as metallic, glossy, roughness, and so on. For some, even needed to create a more complicated shader, by combining a few particular nodes together.

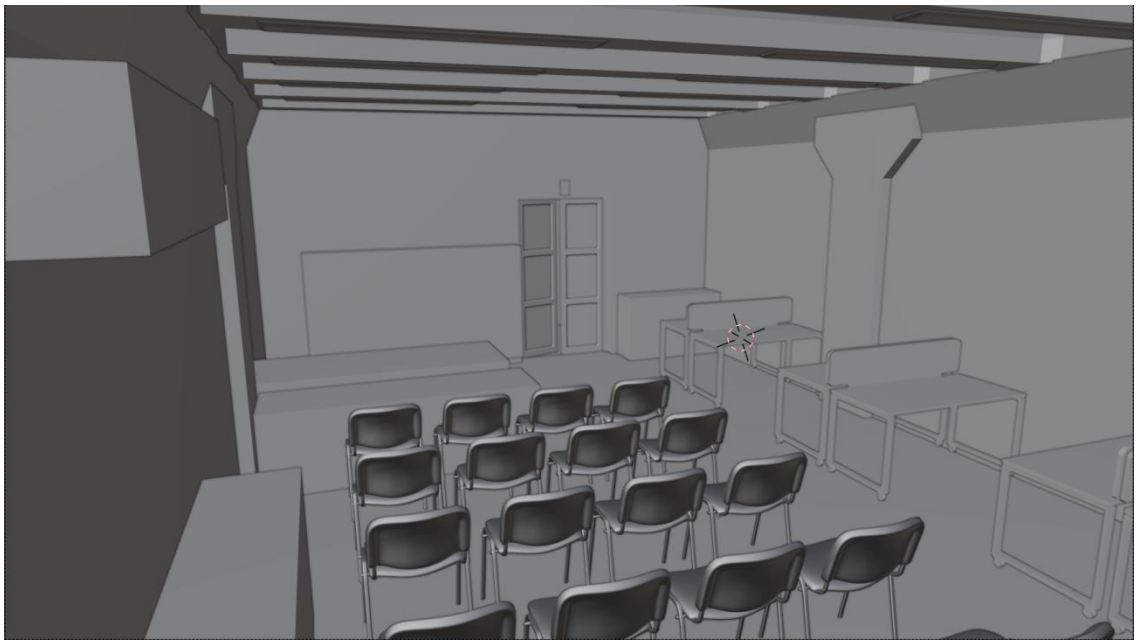

Image 2.16 – Scratch model of the lab

Here, for office chairs was imported a special material for that “fluffy” part. That is great that blender provides a such flexibility with material creation and modification[Image 2.17].

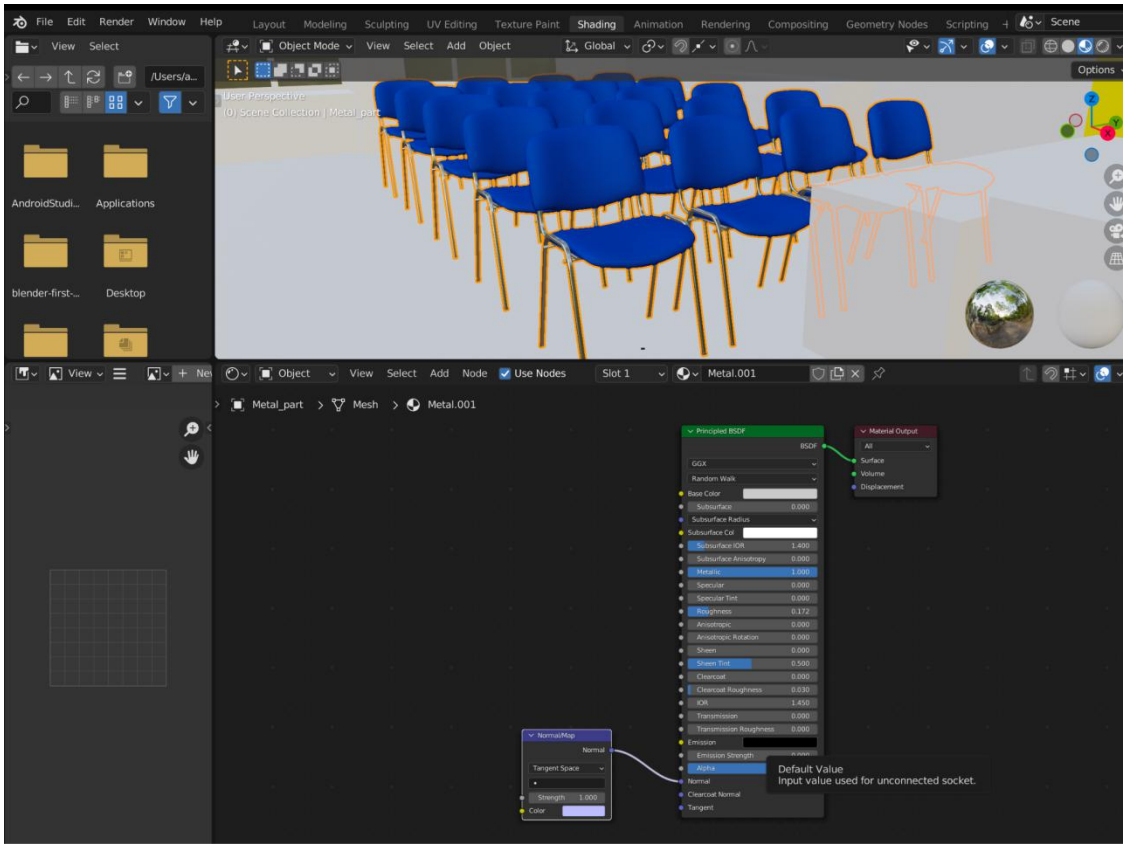

Image 2.17 – Shading node-tree example

On the other hand, some objects are required for custom materials which will have real-life photos inside. For example, these cubes. Photos were taken from real cubes in the room from each angle. Then, in Photoshop they very cropped, straightened and merged all together to one final texture. Then, in blender, by using the image as a color input, its possible to make a UV-unwrap on the image. The process is basically about taking all verticies from an object and then assigning each face each part of the image accordingly[Image 2.18].

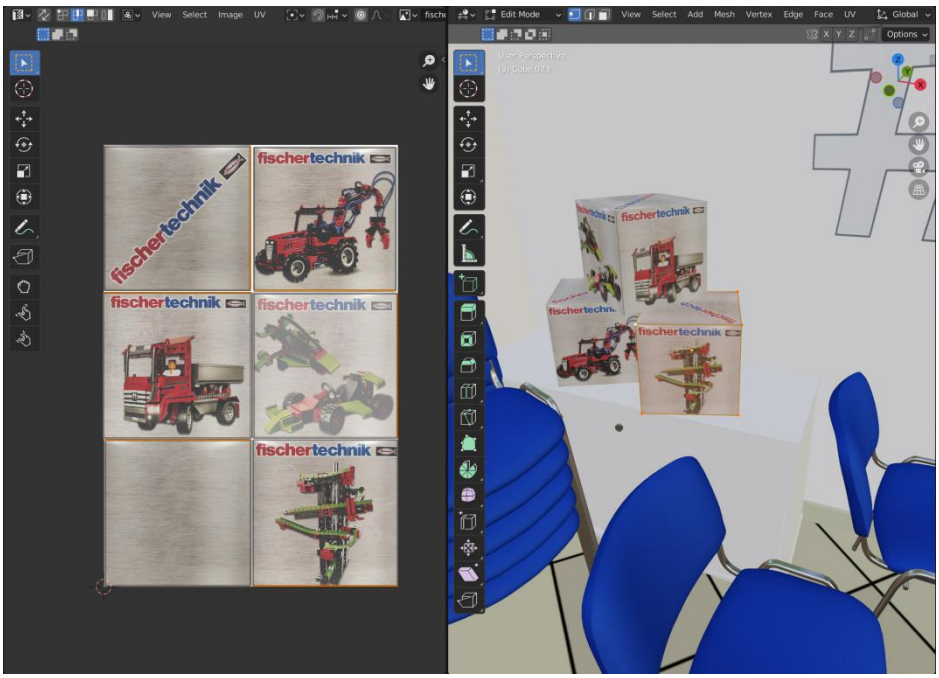

Image 2.18 – Shading node-tree example

If UV-unwrap was made incorrectly or wasn't been made at all - result will be unappropriated[Image 2.19]. The texture will be an unscaled, only on one side of an object. To fix it it's needs to be unwrapped and then manually assigned to the image's position.

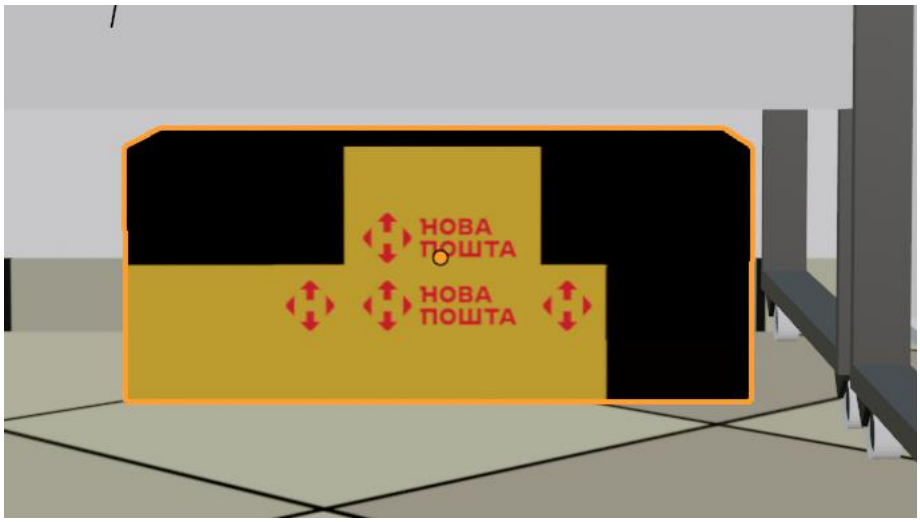

Image 2.19 – False UV-unwrap

Another option how to add some texture to an object is to create a paint map. Steps stays the same as from the previous UV-unwrapping thing. Later on, user needs to create a canvas and assign it to the material property of the object. Then, if everything was made successfully, it's possible to paint on the canvas and result will be show on the object itself[Image 2.20].

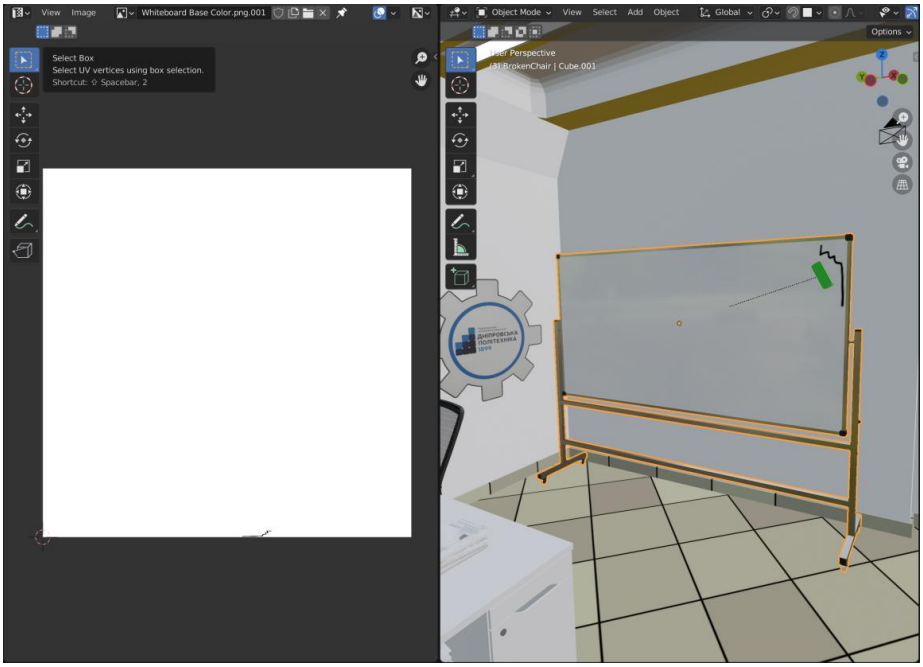

Image 2.20 – Canvas painting

And as always, base for every material goes built-in node-programming system. Which is great basically for everything. It's easy to make some simple materials, as well as some more complicated procedural. For example, in particular case, the background color of image was not a satisfied result, so it was needed to add a custom background color[Image 2.21]. To do so, mix shader node was connected.

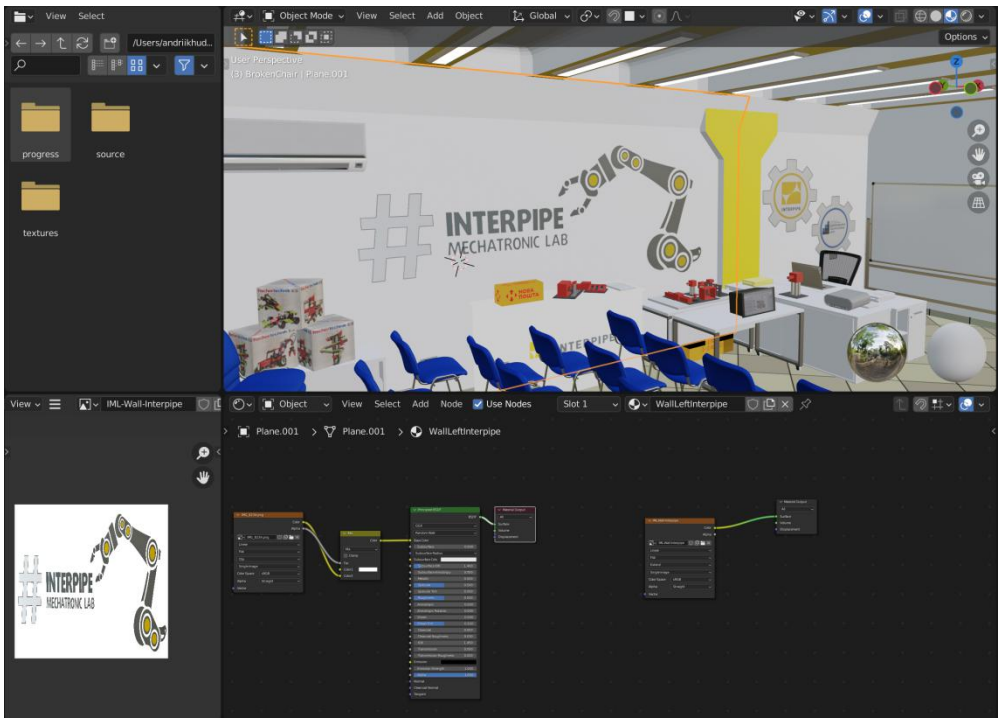

Image 2.21 – Shading node-tree example

While this node system is great in Blender, then it's well recognized in other software, such as in Unity. It can understand a simple material features, such as color and similar, but when things come to more complicated, even just passing colors from an image texture, it doesn't work in unity, unfortunately.

In order to export object with materials to unity, firstly, it's needed to bake a texture with native cycles render. And then, on the end, it will give an image with unwraped texture color of selected object[Image 2.22].

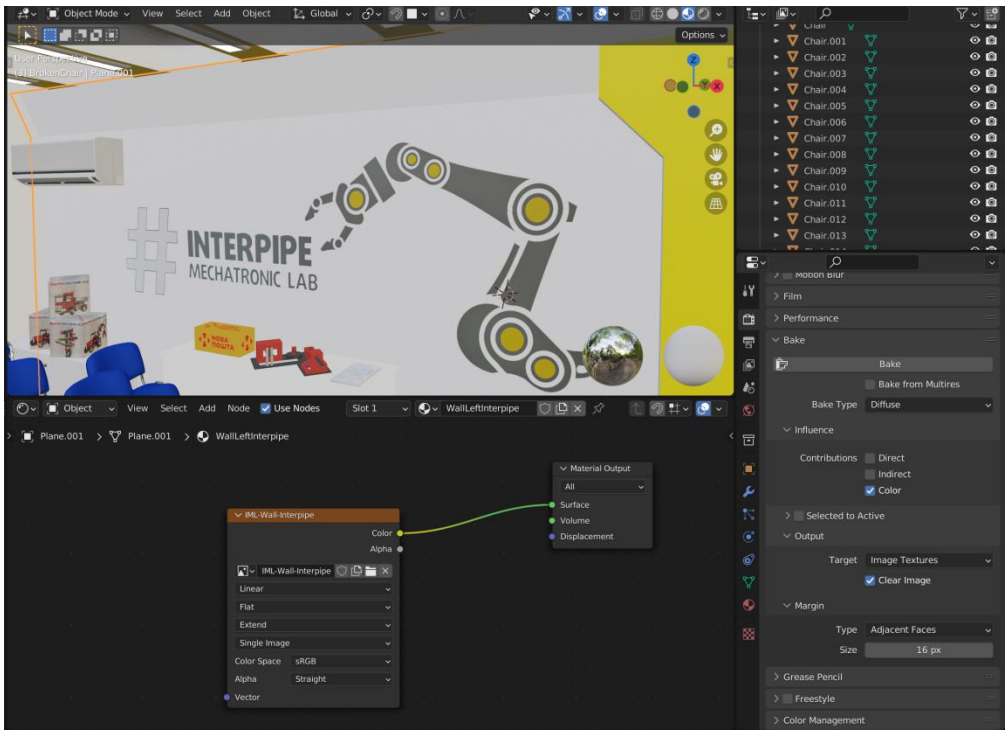

Image 2.22 – Shading node-tree example

2.4. Conclusions for the Chapter

The second chapter's focus on the preparation of materials for VR environment development underscores the critical importance of accurate and detailed digitization processes. The collection of photo data and the subsequent creation of 3D models using Blender software reveal the complexities and challenges associated with producing high-quality VR content. The advantages of using Blender include its robust capabilities for detailed modeling and its wide range of features that support realistic simulations. The development of materials for these models requires careful consideration of various parameters to ensure they meet the standards necessary for an immersive VR experience. These preparatory steps are vital for the successful implementation of VR technologies.

3. Development of the VR Environment

3.1. Selection of Software for VR Development

Virtual reality (VR) development has advanced significantly in recent years, leading to a diverse range of software tools that cater to different aspects of creating immersive

experiences. The selection of appropriate software is pivotal for the success of VR projects, whether for gaming, education, or industrial simulation. Among the various options, Unity has been chosen as the preferred engine for its versatility and comprehensive support. This article outlines key considerations and details why Unity is a leading choice for VR development.

1. Platform Compatibility: It is essential to choose software that supports multiple VR platforms such as Oculus Rift, HTC Vive, and PlayStation VR. Unity excels in this area, offering extensive cross-platform capabilities that ensure VR applications can reach a broad audience.

2. User Experience Design: The software should offer robust tools for designing intuitive and engaging user interfaces. Unity's integrated development environment allows for rapid prototyping and iteration on user interactions, which is crucial for VR.

3. Graphics and Performance: High-quality graphics are at the heart of creating immersive VR environments. Unity provides powerful graphic features hat do not compromise performance, along with optimization tools that help streamline VR applications for various hardware specifications.

4. Development Environment: A flexible and powerful development environment is necessary. Unity scores highly with its drag-and-drop functionality, extensive asset libraries, and real-time testing capabilities, enhancing productivity and simplifying the development process.

5. Support and Community: A strong support network and an active community can be invaluable resources. Unity boasts a large community of developers, comprehensive documentation, and an asset store filled with ready-made assets and plugins, which can significantly accelerate development timelines.

Unity stands out in the VR development landscape for several compelling reasons. Its ability to deploy on multiple VR platforms is a critical advantage, ensuring that developers can create applications that are accessible on various devices, maximizing reach and usability. Unity provides a wide range of development tools that cater to both

beginners and experienced programmers, featuring a user-friendly interface and the option to use powerful scripting languages like C#. The Unity Asset Store is another valuable resource, offering thousands of assets and tools that can be integrated into projects, which not only saves time but also enhances the quality of the final VR experience. Additionally, with one of the largest communities of technology enthusiasts and developers, Unity offers unparalleled support through forums, tutorials, and user groups. This community is an excellent resource for troubleshooting and learning, making the development process smoother and more efficient. Furthermore, Unity is continually updated with the latest technological advancements, and its commitment to incorporating new features and improvements ensures that VR applications developed with Unity are using cutting-edge technology.

3.2. Environment Configuration for VR Development

HDRP was initially chosen for its high-fidelity graphics capabilities, designed to produce cinematic quality and realism. This pipeline is highly beneficial for projects that require photorealistic visuals, such as detailed simulations or advanced gaming environments.

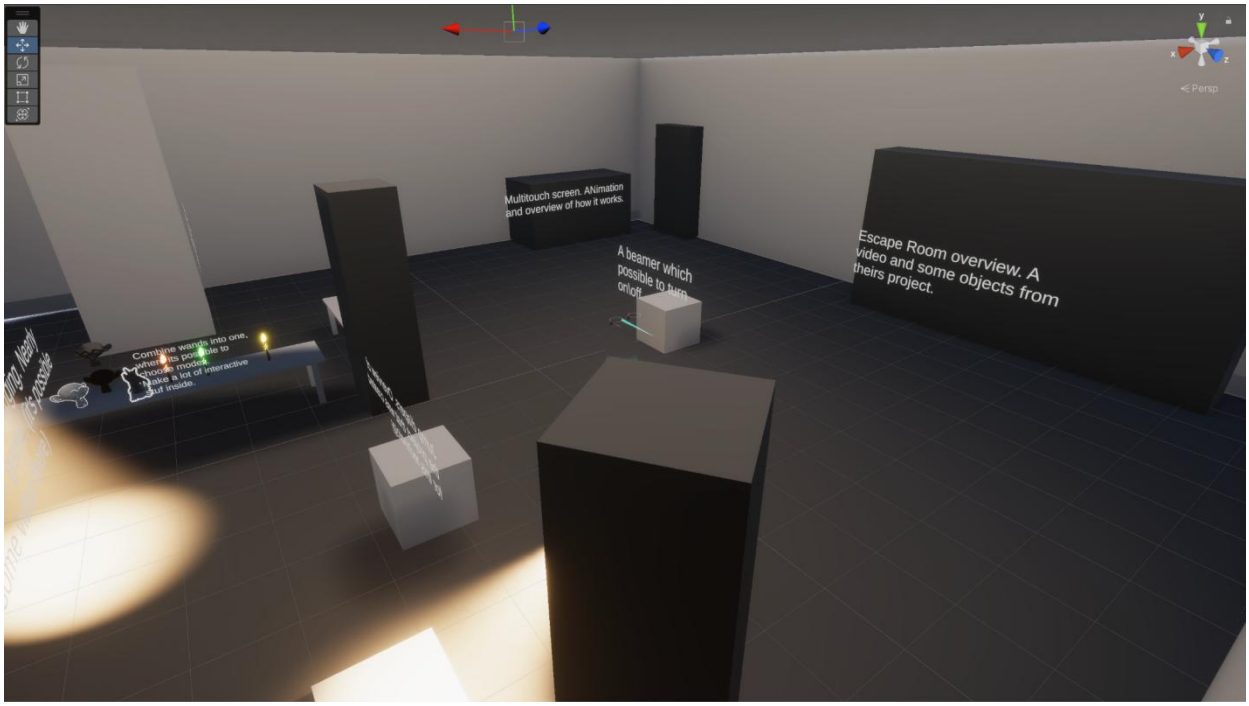

Image 3.1 – Blackout

Pros of HDRP:

- Superior Visual Quality: Capable of delivering highly realistic visuals, making it suitable for projects demanding the highest level of graphic detail.
- Advanced Lighting: Features complex lighting models, including volumetric effects that enhance the depth and realism of environments.
- Detailed Graphic Control: Offers extensive customization options for graphics settings, allowing for precise control over visual effects and performance.

Cons of HDRP:

- Demanding Hardware Requirements: Requires robust hardware specifications, which can be a limitation for less powerful systems like standalone VR headsets.
- Increased Complexity: The complexity of setting up and maintaining HDRP can lead to longer development times and potentially higher costs.

As the project progressed, it became evident that HDRP's requirements were not fully compatible with the hardware limitations of the Oculus Quest 2. Consequently, the decision was made to switch to URP, which is better suited for ensuring performance efficiency and compatibility with less powerful devices.

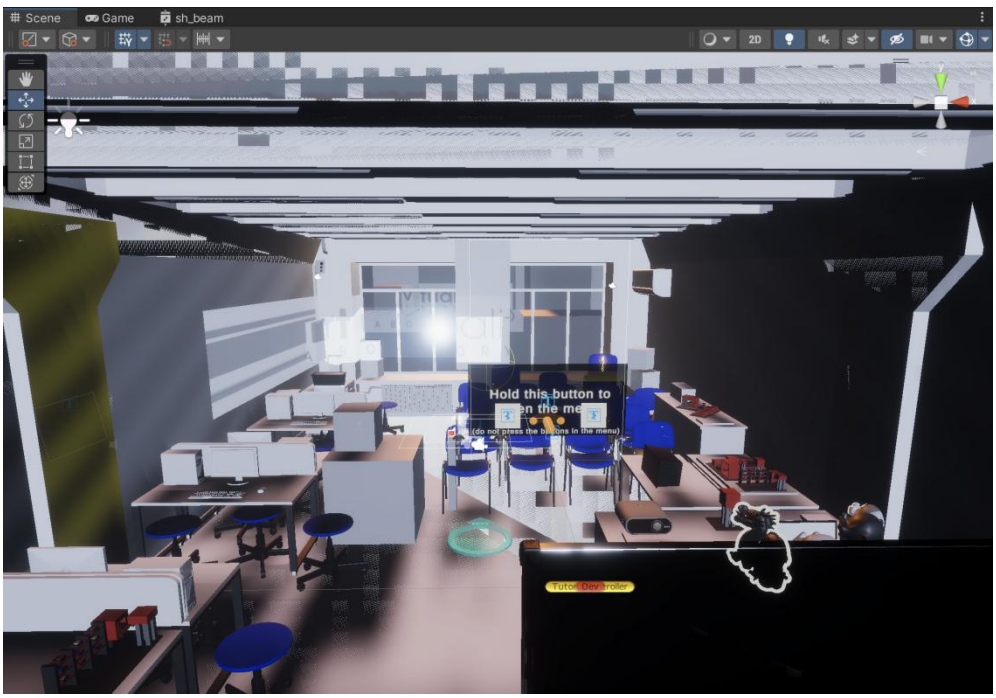

Image 3.2 – Shading node-tree example

Pros of URP:

- Optimized Performance: Lighter and more efficient, URP is designed to perform well on a wide range of hardware, including devices with limited processing capabilities like the Quest 2.
- High Scalability: Adapts seamlessly across different platforms, from mobile devices to desktops, facilitating easier cross-platform development.
- Adequate Visual Quality: While it doesn't match HDRP's level of detail, URP still provides satisfactory visual quality for immersive VR experiences.

Cons of URP:

- Reduced Photorealism: URP does not support some of the high-end visual effects possible with HDRP, which may affect projects requiring the utmost in visual fidelity.
- Limited Advanced Features: Some advanced lighting and shadowing techniques available in HDRP are not supported, which could restrict creative possibilities.

3.2.1. URP setup

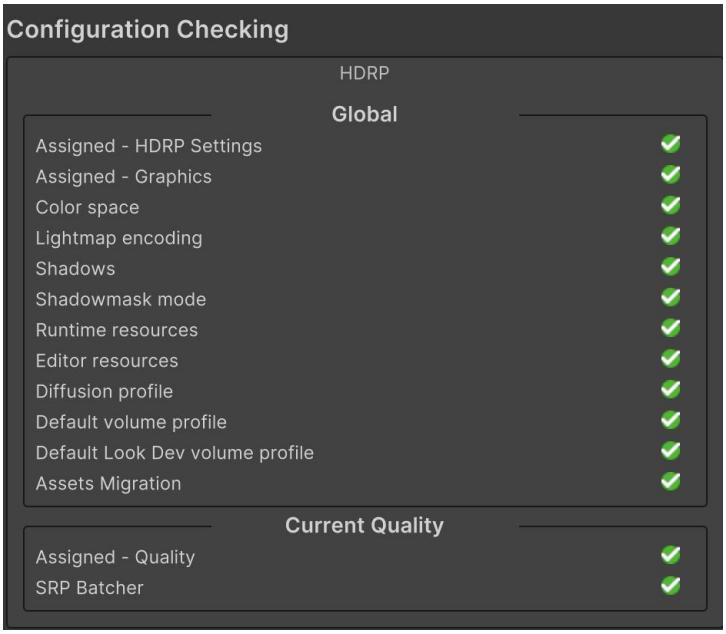

Image 3.3 – URP/HDRP startup configuration

Here are the steps followed to establish a base URP project in Unity:

Unity was launched, and a new project was initiated by selecting "Universal RP" from the template options to create a URP project.

Once the project was set up, the "Window" menu was accessed, followed by a selection of "Package Manager". In the Package Manager, a search for "URP" was conducted to ensure that the "Universal Render Pipeline" package was installed and up-to-date.

A new Scene was created by navigating to "File" -> "New Scene", and a new "Directional Light" was added to the Scene through "GameObject" -> "Light".

To configure the URP settings, "Edit" -> "Project Settings" -> "Graphics" was selected, setting the "Scriptable Render Pipeline Settings" to "UniversalRP-HighQuality".

A new folder named "Materials" was created within the "Assets" folder to house all the Materials for the Scene.

A new Material was generated by going to "Assets" -> "Create" -> "Material", naming it "Default Material", and choosing "URP/Lit" as the Shader.

Using the Inspector window, the properties of the Material, such as color, metallic, and smoothness, were adjusted to modify its appearance in the Scene.

The Scene was saved by selecting "File" -> "Save Scene".

Following these steps enabled the setup of a basic URP project in Unity, complete with a primary light source, configured URP settings, and a Material for rendering objects in the Scene. This setup provided a foundation for further adding and customizing objects to craft a unique project..

3.2.2. HDRP pipeline profiles

In the URP graphics pipeline, several profiles are available that help optimize rendering performance and tailor rendering settings for various scenarios. Here's an overview of some of the most common profiles within URP:

Image 3.4 – URP pipelines

Quality Settings: This profile allows users to fine-tune the overall quality of the rendering. It includes options for adjusting texture quality, anti-aliasing, and shadow quality. Users can use the Quality Settings profile to find a balance between visual quality and performance, tailored to specific needs.

Lighting Settings: The Lighting Settings profile is designed for tweaking the lighting and shadows in a scene. It encompasses settings like the intensity and color of the sun, shadow quality, and the level of global illumination. Adjustments in this profile can significantly alter the mood and atmosphere of a scene.

Volume Settings: This profile manages the post-processing effects in a scene, such as color grading, bloom, and ambient occlusion. The Volume Settings profile is crucial for defining the visual style of a scene and enhancing the rendering with additional depth and detail.

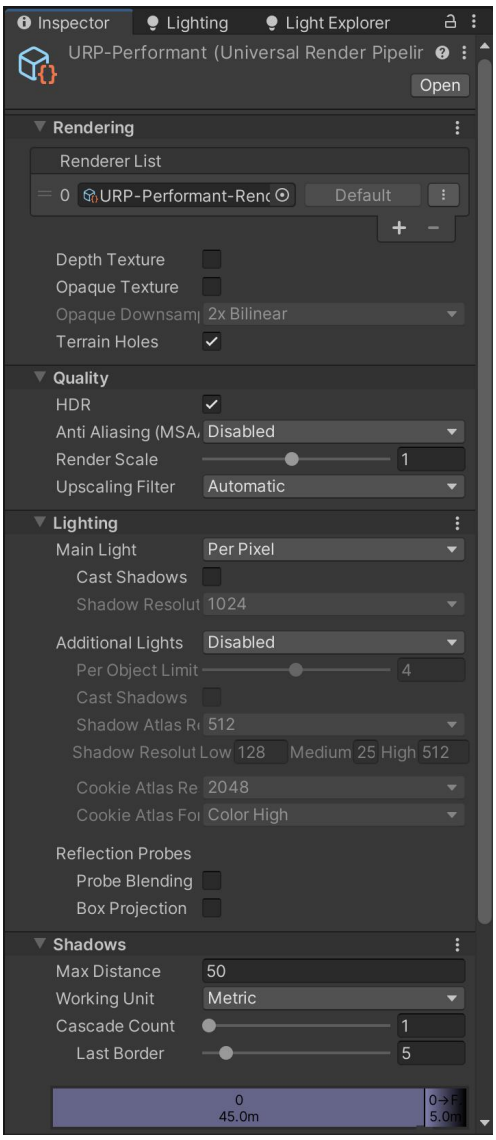

Image 3.5 – In-depth settings of render pipeline

Shader Settings: Used to control the behavior of individual shaders, this profile includes adjustments for material detail levels, rendering modes for transparent objects, and reflection levels on reflective surfaces. The Shader Settings profile helps in optimizing the visual quality and performance of specific objects within a scene.

VFX Settings: Focused on the behavior of visual effects, the VFX Settings profile includes adjustments for particle systems, post-processing effects for particles, and detail levels for specific visual effects. This profile is key to enhancing the visual impact and performance of particular effects in a scene.

Overall, the various profiles offered in URP equip users with a comprehensive toolkit for optimizing rendering performance and crafting customized visual styles for

their projects. By tweaking the settings within these profiles, users can achieve immersive graphics that are perfectly suited to their unique requirements.

3.2.3. Lighting setup

In the project, the goal was to create an visually beautiful experience using Unity's URP lighting capabilities. Recognizing that visuals should not compromise performance, especially in VR where a smooth frame rate is crucial, the lighting setup was carefully optimized to balance visual realism with optimal performance for VR platforms.

Baking Indirect Lighting:

To enhance visual fidelity while maintaining performance, light baking was utilized to precompute indirect lighting in the scenes. This strategy reduced the need for real-time calculations, ensuring smoother and more consistent frame rates. The team explored both real-time and baked global illumination options within HDRP, predominantly choosing baked lighting solutions to maintain a stable and immersive VR experience.

Optimizing Real-Time Shadows:

Shadows are critical for adding depth and realism to the VR environment, but they can be resource-intensive. The following strategies were implemented to optimize real-time shadows:

The number of shadow cascades was carefully managed to balance shadow quality and performance.

Shadow resolution and distance were adjusted considering the close proximity of VR users to objects.

Shadow masks were used to restrict shadow casting to specific objects, significantly reducing the load in complex scenes.

Image 3.6 – Optimized lighting setup

Lighting Quality and Performance Settings:

Adjustments to the quality of individual lights were made to optimize performance without sacrificing visual appeal:

Light cookies were employed judiciously to add visual depth without adversely affecting performance.

Light intensity and range were tailored to scene needs to avoid performance drains from excessive lighting.

Light layer culling was leveraged, enabling selective rendering of lights based on camera proximity to reduce unnecessary computations.

Dynamic Lights and Batching:

To maximize performance, the number of dynamic lights in scenes was minimized. Given their computational expense, static or mixed lighting setups were favored. HDRP's light modes—such as "Realtime," "Mixed," and "Baked"—were strategically used based on specific project needs. Techniques like GPU instancing and light baking were also employed to lower draw calls and enhance performance.

Lighting for Performance Variants:

With the varied performance capabilities of VR platforms in mind, the project aimed to accommodate different devices by creating performance variants of scenes. This allowed users to select from different lighting settings or detail levels, catering to both high-end and lower-end VR hardware.

By rigorously optimizing the lighting setup, the project successfully delivered an immersive and visually captivating experience without sacrificing performance. Regular testing across various VR devices helped fine-tune the project, ensuring it achieved performance goals while providing users with a memorable and comfortable VR experience.

The initial model was one-sided plane. As seen on the image[Image 3.7], Red are faces facing in the opposite direction, which means, they are transparent in unity and will pass the lights throw. To fix that, model should be solid, so for all plane objects, Solidify modifier was applied and given result fixed all the problems[].

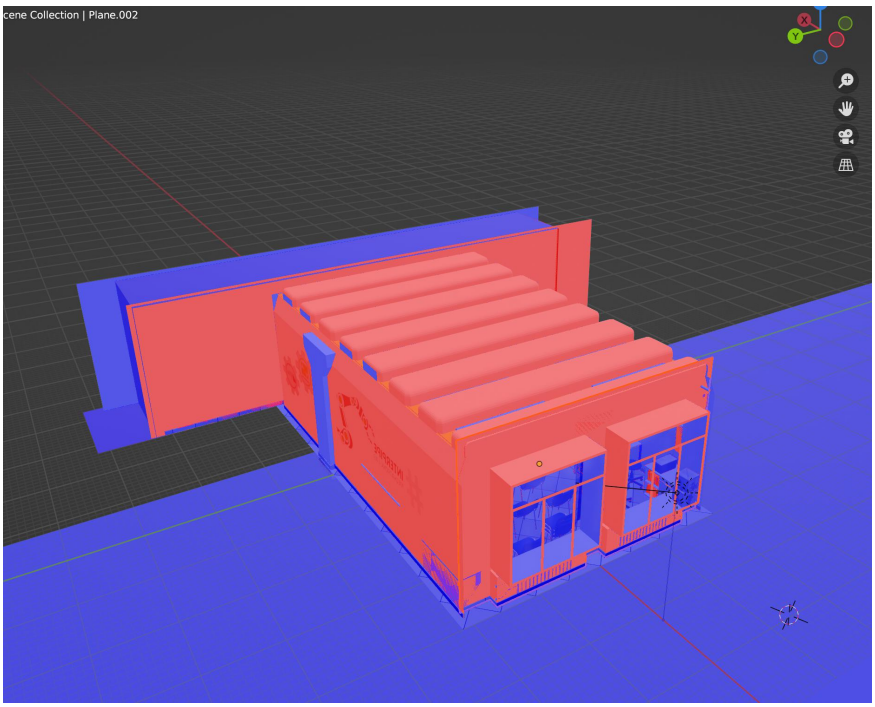

Image 3.7 – Face orientation for room's boundaries

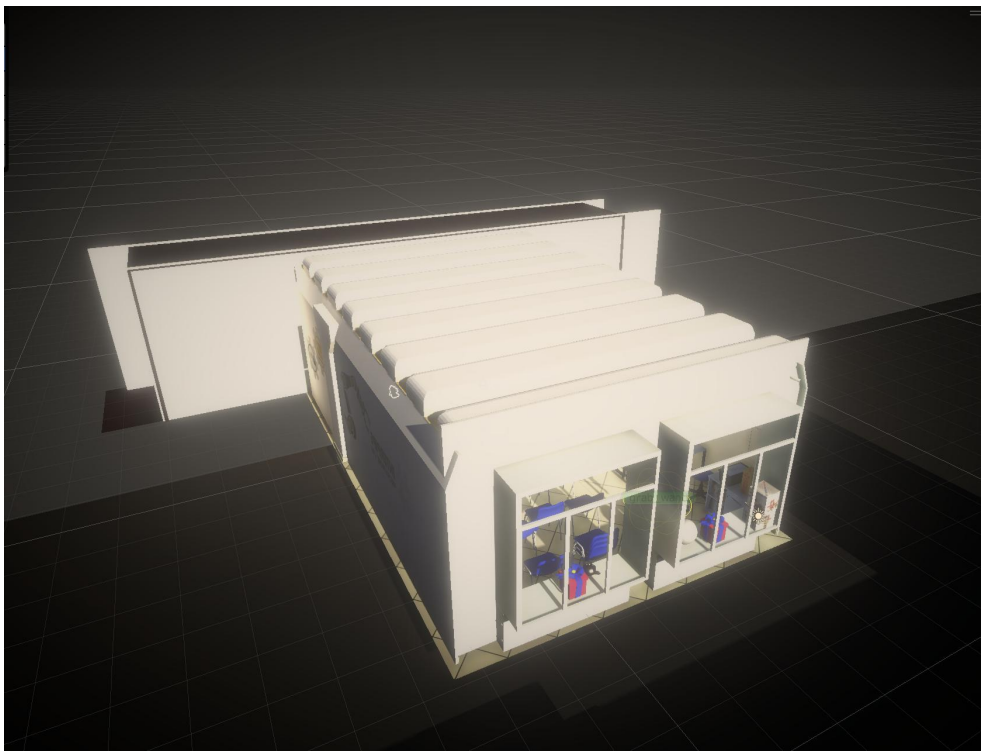

Image 3.8 – Boundaries after solidifying

3.3. VR interaction framework

To incorporate interactive features into the VR project, the VR Interaction Framework was utilized[10].

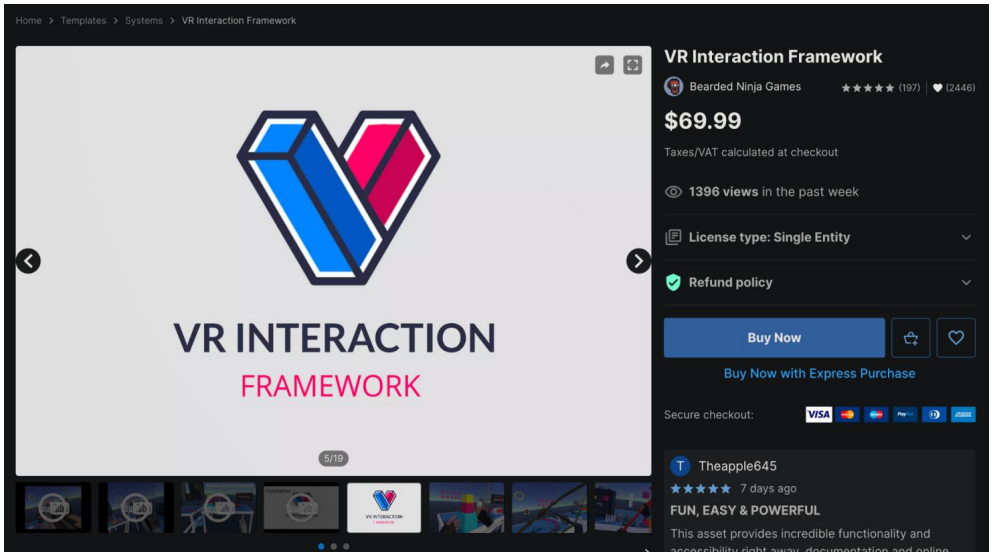

Image 3.9 – VR Framework page in AssetStore

The VR Interaction Framework comes equipped with an array of pre-built components and tools designed to simplify the implementation of various VR interactions. These interactions include grabbing and manipulating objects, interacting with UI elements, and navigating through menus and complex systems. Additionally, the framework offers numerous customization options and settings, allowing developers to tailor the interaction behaviors to meet specific project requirements.

One of the primary advantages of using the VR Interaction Framework is the significant reduction in time and effort required to develop high-quality VR interactions. The availability of pre-built components, coupled with comprehensive documentation and resources, enables developers to rapidly and efficiently create engaging and immersive VR interactions.

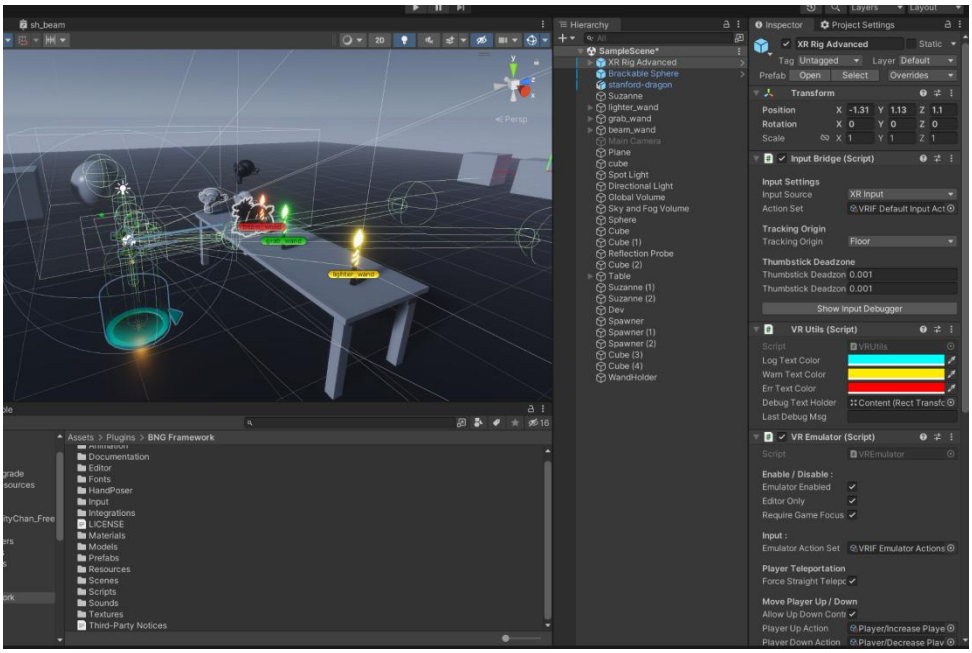

Image 3.10 – VR Interaction character's rig

The framework is also highly optimized for VR, ensuring a smooth and responsive user experience even on less powerful hardware. The VR Interaction Framework proved to be an invaluable tool for implementing interactive features in the VR project, making it a recommended choice for developers engaged in VR application development.

For character input, the “XR Rig advanced” was employed, providing fundamental controls and the flexibility to switch between different hand or character models.

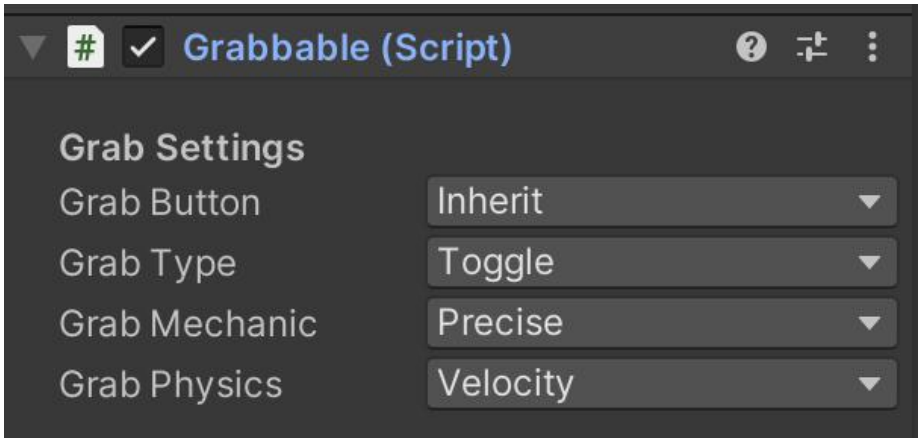

Image 3.11 – Script which allows to grab objects with hands

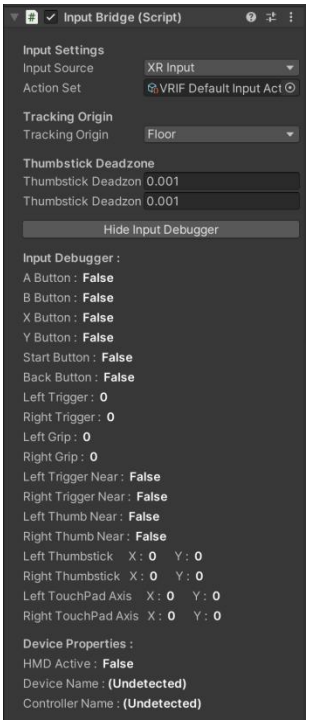

Image 3.12 – Input Bridge

The central script used is the “Input bridge,” which facilitates debugging of the input device. Additionally, to enable object grabbing, a “Grappable” script was added to the respective object, further enhancing the interaction capabilities within the VR environment.

3.4. Custom wrappers\bridges for VR input

To develop a clear and flexible control system for Wands in the VR project, I crafted an abstract Wand bridge script. This script extends the GrabbableEvents class from the VR Interaction Framework (VRIF) and controls a boolean value "isActive" to manage whether the wand is active.

```
public override void OnTriggerDown()
{
    _isActivated = !_isActivated;

    if (_isActivated) OnWandActivated();
    else OnWandDisabled();

    OnWandHolding(_isActivated);

    base.OnTriggerDown();
}

public override void OnRelease()
{
    if (disableWhenDropped)
    {
        OnWandDisabled();
        OnWandHolding(false);
    }

    base.OnRelease();
}
```

Image 3.13 – Wand Bridge

```
private void Update()
{
    TouchpadParser();
}

private void TouchpadParser()
{
    if(thisGrabber == null) return;

    Vector2 touchpadInput = Vector2.zero;

    if (thisGrabber.HandSide == ControllerHand.Left)
        touchpadInput = input.LeftThumbstickAxis;

    if (thisGrabber.HandSide == ControllerHand.Right)
        touchpadInput = input.RightThumbstickAxis;

    OnThumbstickAxis(touchpadInput);
}
```

Image 3.14 – Touch-pad bridge

Moreover, I included a parser within the same script to handle touchpad input controls. The challenge here was that VRIF provides separate getters for touchpad inputs—left and right. Since the specific input depends on which arm the wand is equipped on, the script needed to first identify the wand's arm location and then select the correct joystick input.

This Wand script is designed to act as a Parent class. Its main role is managing the visual aspects of the wand based on its operational state. Occasionally, it also offers methods or variables to other scripts as needed. This structure not only makes the wand

controls intuitive but also ensures they are responsive, enhancing the overall interactivity and immersion of the VR experience.

3.5. Adaptation of Models for VR Interaction

Since all models which are modeled in Blender should be exported to Unity, there are some extra steps that's needs to be done. First of all and the most important, fix Face orientation on object. Ideally, everything should look blue with "face orientation" filter turned on[Image 3.15]. Occasionally, it's not a case, so now it should be done manually by developer.

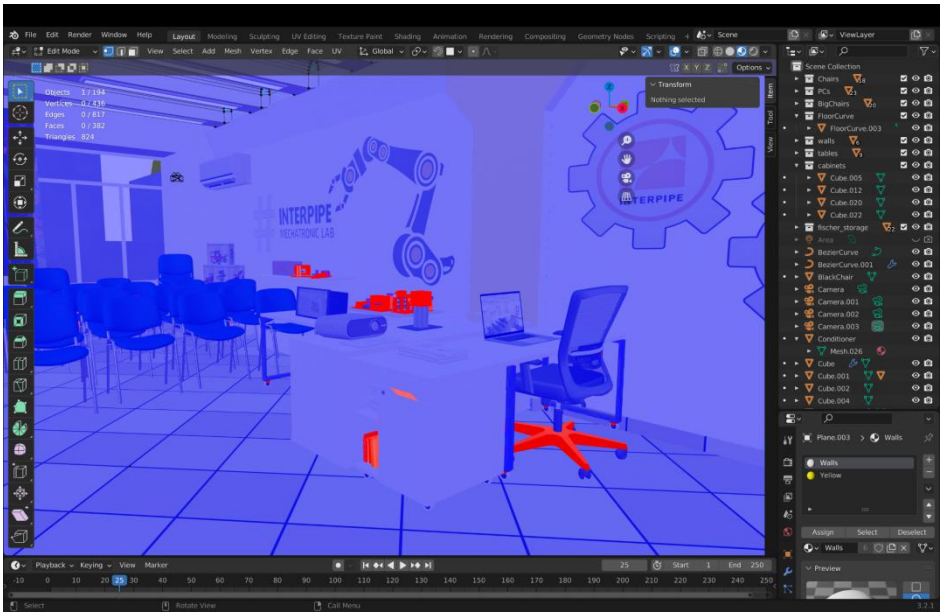

Image 3.15 – Wrong face orientation look in Blender

On the Image 3.16 it's clearly visible what's happening in Unity when faces have a wrong orientation position. The part of an object which is red on the first image[Image 3.15], appears transparent in Unity. The fix is the simple Blender command "Invert orientation", but it doesn't work for all, so it's need to be done piece by piece.

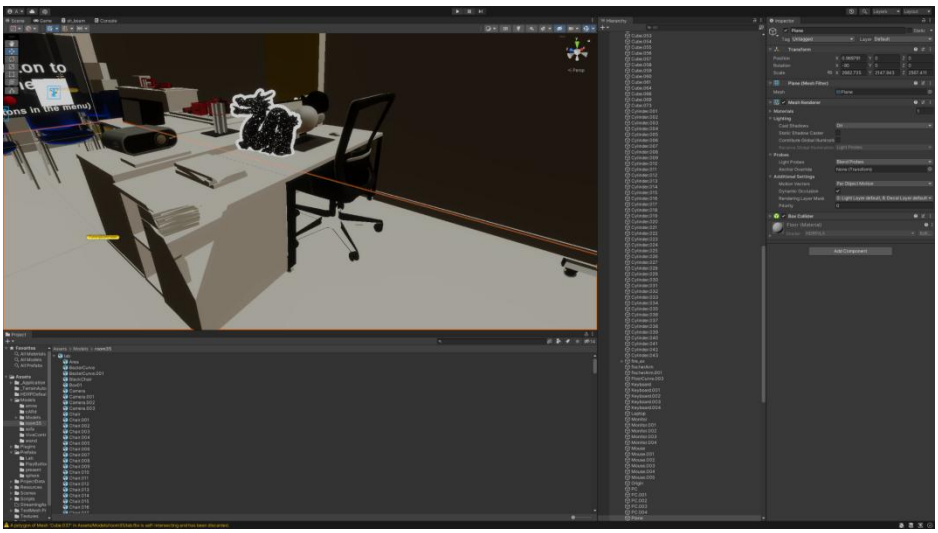

Image 3.16 – Wrong face orientation look in Unity

The following image show more clearly direction of each face[Image 3.17]. The goal is to make all of them to out of the object and not inside.

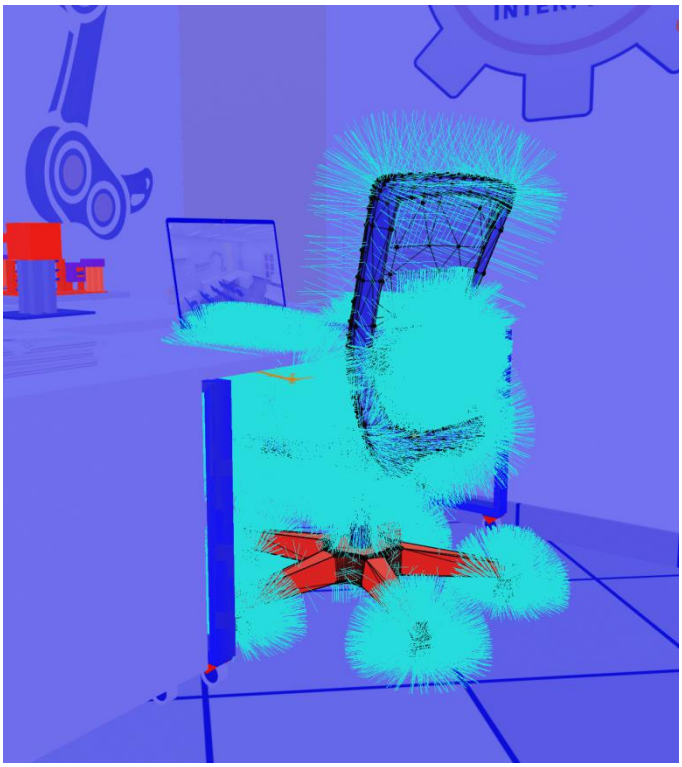

Image 3.17 – Shading node-tree example

3.6. Addition of Interactive Elements

3.6.1. HDRP projector controller

Since our lab setup includes a projector, it was necessary to integrate it into the project. However, we faced a challenge with the High Definition Render Pipeline (HDRP) in Unity, as it lacks a "correct" projector component. Typically, a projector component should simulate light projecting from a single point (the lens) and expand as it moves toward the wall. Unfortunately, while this works as expected in the default render pipeline, it does not function correctly in HDRP, and the reasons for this discrepancy are unclear.

Moreover, the projector in our setup is only used to display text. To mimic the lighting effect that a real projector would have, a SpotLight was added alongside the projector component.

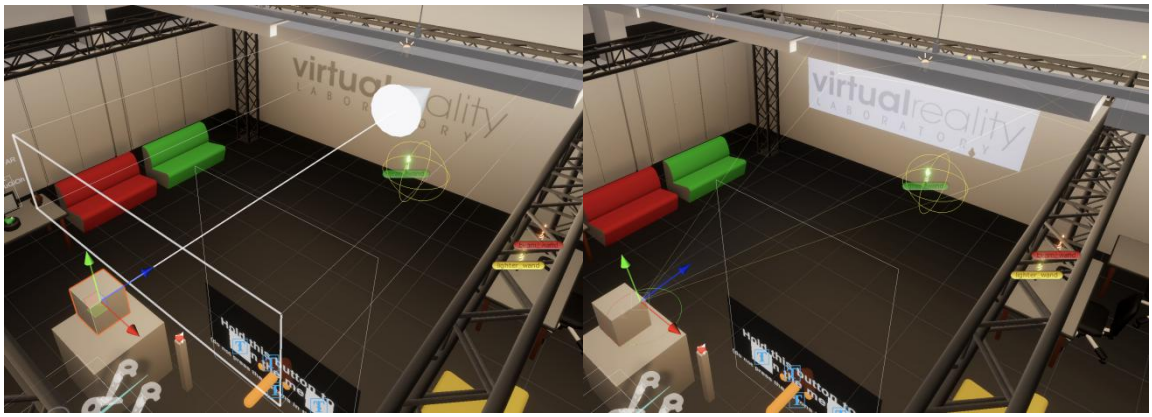

Image 3.18 – Projector illumination

To work around the limitations of the HDRP regarding the projector, we developed a bypass code. This code dynamically adjusts the size of the projector component based on its distance to the object in front of it. This solution helps maintain the realism and functionality of the projector within our HDRP-based project, ensuring that the projected image behaves as closely as possible to a real-world scenario.

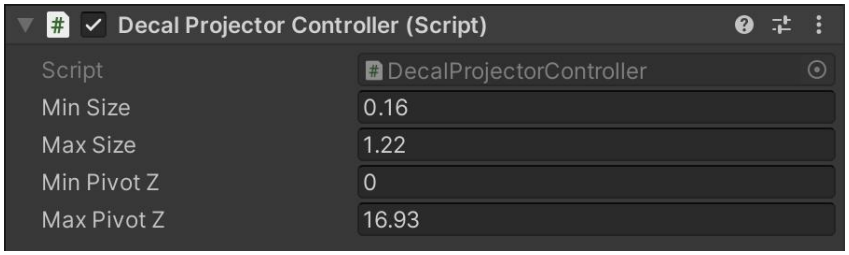

Image 3.19 – Custom decal controller

3.6.2. Physical interaction

A critical feature of the project was to enable interaction with nearly every object, necessitating the use of numerous rigidbodies and colliders. A challenge arose when using Mesh collider and Rigidbody components together; for these components to function properly, the collider must be "Convex." This configuration simplifies the collider's shape, as depicted in the second image, which is not suitable for objects with complex or curved shapes.

To overcome this limitation, a straightforward solution was devised. Instead of relying on a single, complex mesh collider, two box colliders were placed around each object. These box colliders were carefully positioned to closely match the original shape of the mesh. This approach allowed the project to maintain interactive capabilities without compromising the physical accuracy and realism of the objects.

Image 3.20 – 2 simple box colliders

Image 3.21 – Mesh collider with convex mesh

But on some objectsMesh collider still was applied because it was easier to use this one except:

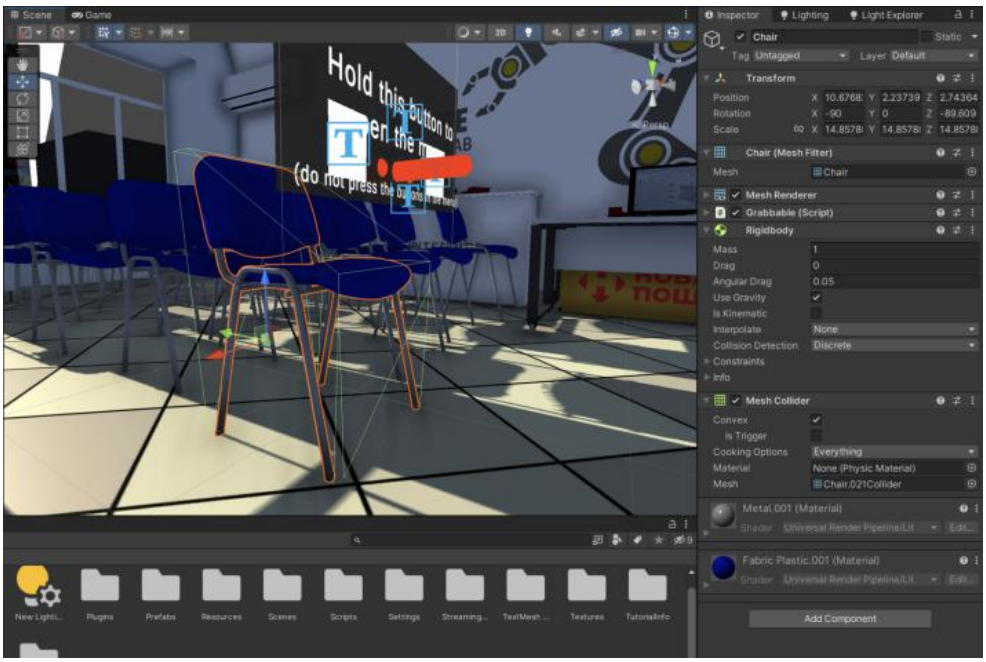

Image 3.23 – appropriate convex mesh colliders

Another example, where there were used multiple box colliders instead of mesh collider. It's a table, and it should be hollow, convex mesh collider won't do that. And a chair, because it should stay underneath the table[Image 3.24].

Image 3.24 – Chair's colliders

3.6.3. Objects destruction

System of dynamic object breaking was implemented. The logic is not that complicated - there is one normal object, and when it hits something with velocity > threshold, then Script changes it on the broken variation of the same object[Image 3.25].

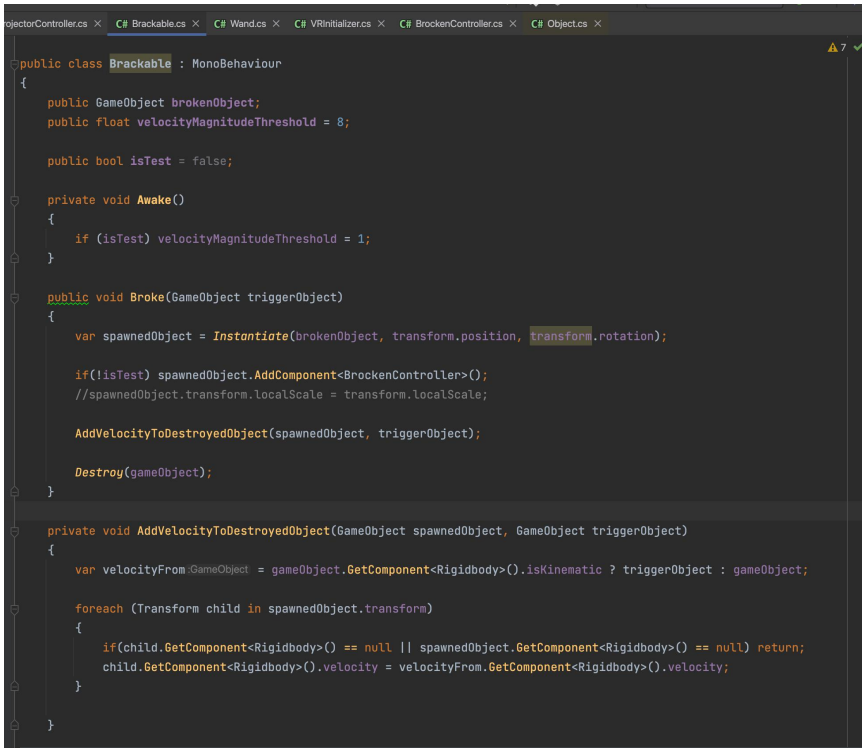

Image 3.25 – Breakable script

Beforehand, a broken model of an object should be prepared. It is done by using some plugin in Blender, which spawns a banch of objects in an area and then culls them off by applying a boolean modifier with original object. Result of the work are Image 3.26 and Image 3.27.

Image 3.26 – Breakable script

Image 3.27 – Breakable script

The same was done to the windows. The only exception was to remove their velocity when destroyed, because by the idea windows should stay in their frames and not fly away.

Image 3.28 – Normal windows

Image 3.29 – Broken windows

3.6.4. Tutorial

The tutorial is a key part of the project and a lot of effort went into making it. Users can start the tutorial in two ways: by pressing a red button or by choosing an option from a menu.

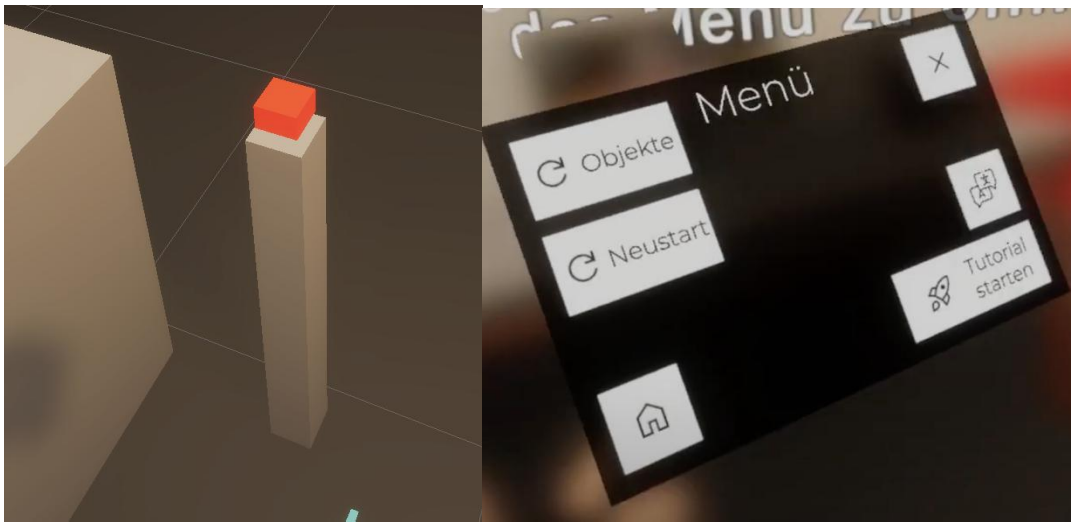

Image 3.30 – Tutorial and Menu

To press the red button, users simply place their arm on it, just like they would in real life. If they need to, they can restart the tutorial from the menu, making it easy to use.

The tutorial is divided into several chapters:

"Press All Buttons"

"Move to Target"

"Menu Buttons"

"Pick Up a Wand"

"Lift an Object with a Wand"

Once these chapters are completed, the tutorial ends. It teaches users how to interact with the virtual reality setup and the project's features. There is a main controller script that controls how the tutorial runs and sends signals to a local controller for each chapter, making sure everything runs smoothly.

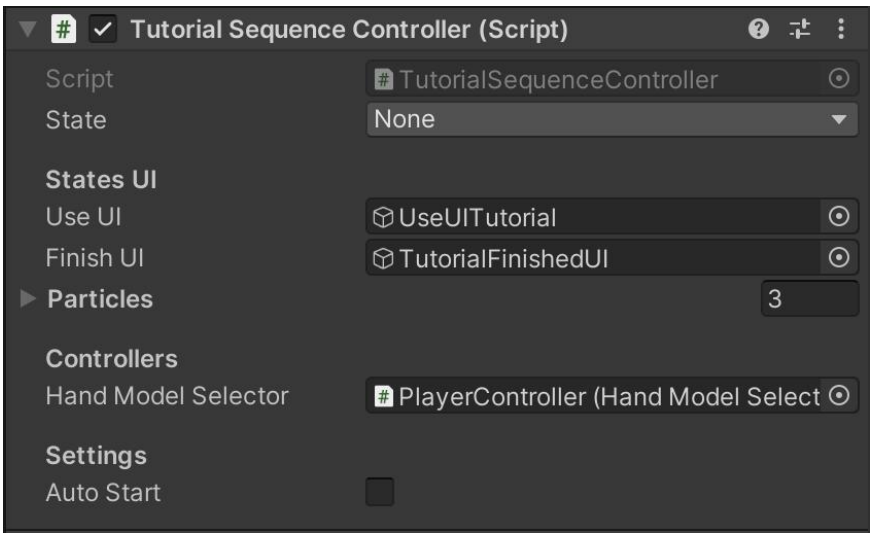

Image 3.31 – Tutorial controller

3.7. Optimization of the Project for VR

3.7.1. Render pipeline optimization

To ensure smooth VR gameplay, unnecessary post-processing effects were disabled and their usage was carefully controlled to strike a balance between visual quality and efficiency.

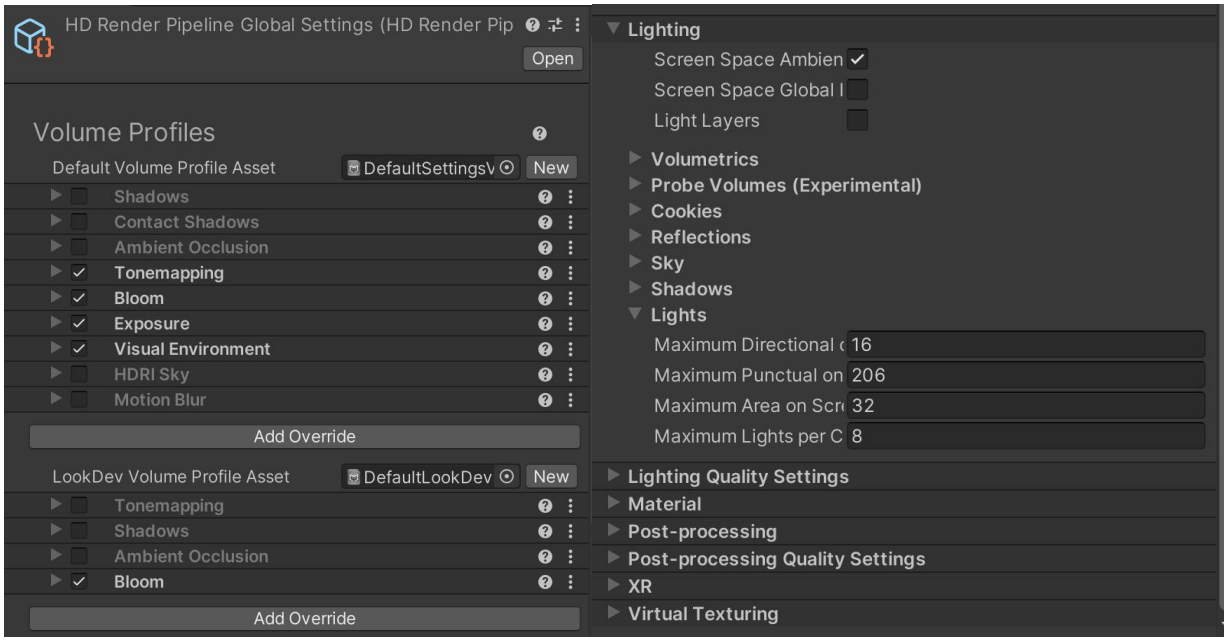

Image 3.32 – Render pipeline optimization

Disabling Motion Blur:

Motion blur, while adding a touch of realism, can be demanding on resources in VR settings. It was completely turned off to reduce GPU stress, helping each frame render more quickly and smoothly. This step significantly improved the VR experience by reducing motion sickness and enhancing user satisfaction.

Managing Ambient Occlusion:

Ambient occlusion, known for its performance impact in complex scenes, was tactically managed. Alternatives like baked ambient occlusion were used where feasible, and adjustments were made to its range and intensity to keep visuals appealing without slowing down performance.

Quality Settings for Post-Processing:

The HDRP allows for the customization of quality settings for various post-processing effects. These settings were adjusted based on the needs of specific scenes and VR platforms. In less critical scenes, effects such as bloom, color grading, and chromatic aberration were dialed down to save GPU resources while still offering a pleasant visual experience.

Conditional Rendering of Effects:

Post-processing effects were conditionally rendered, being activated only during crucial moments or when necessary for storytelling. This selective approach minimized their impact on performance while ensuring visually impactful moments were enhanced when needed.

Performance Profiling and Testing:

Regular performance checks on various VR devices helped identify and address any bottlenecks related to post-processing effects. This ongoing process ensured that the VR project delivered consistent performance across different platforms, enhancing the user experience.

User Experience Feedback Loop:

User feedback from playtesting was actively gathered and proved crucial. It helped gauge how post-processing effects affected the VR experience. With this information, further refinements were made, ensuring that the removal of certain effects didn't detract from the overall immersion or visual quality.

In summary, by carefully disabling unnecessary post-processing effects and optimizing others, the project struck an ideal balance between stunning visuals and optimal performance. This careful optimization process led to a comfortable and engaging VR experience that remained immersive and enjoyable, free from performance-related distractions.

3.7.2. Lightning optimization

Achieving impressive lighting was crucial for creating an immersive and captivating experience. However, there was also a strong focus on the performance implications, especially in VR, where a smooth and consistent frame rate is key to user comfort. Thus, lighting optimization was approached with an emphasis on efficiency, balancing realism with performance.

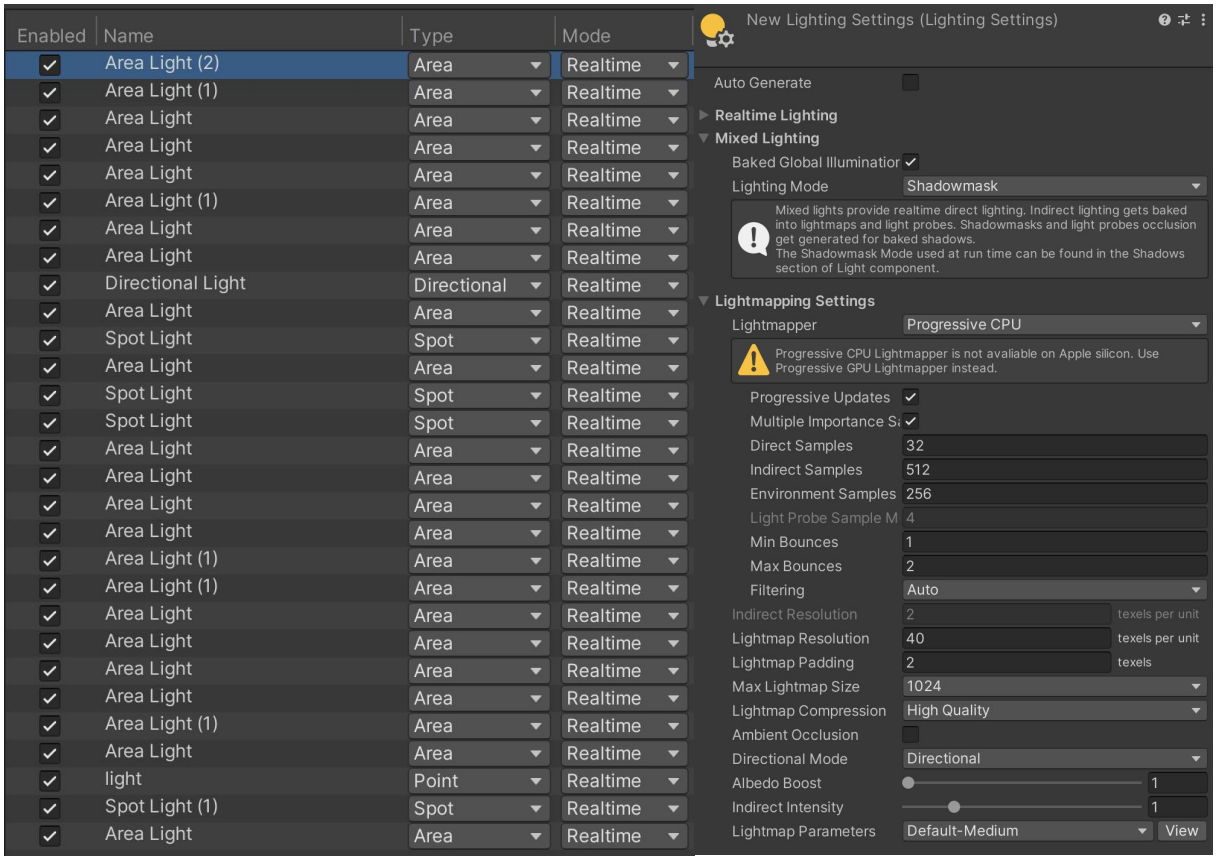

Image 3.33 – Exploring lights

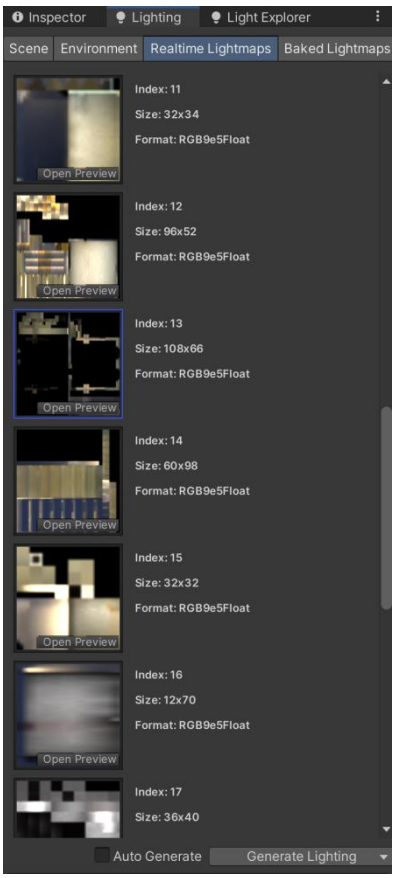

Image 3.34 – Baked realtime lightmaps

Careful Use of Real-Time Lighting:

Real-time lighting offers dynamic and visually appealing effects but can be resource-intensive. The use of dynamic lights was limited, placing them strategically to highlight key areas and enhance the atmosphere without affecting performance. This balance helped maintain dynamism in the VR world while controlling performance impact.

Optimization of Real-Time Shadows:

Real-time shadows contribute depth and realism to the VR experience but require substantial resources. Techniques used to optimize real-time shadows included:

Shadow Cascades: Adjusting the number of shadow cascades helped find a balance between shadow quality and performance, with fewer cascades lightening the processing load while preserving visual fidelity.

Shadow Resolution and Distance: Lower resolution shadows were used for distant or small objects to conserve GPU resources, adjusting settings based on distance and object size.

Shadow Masking: For certain objects or areas, shadow masks were used to restrict the shadow-casting region, reducing the number of shadows rendered in complex scenes and enhancing performance.

Lightmap Atlasing and Compression:

Lighting was further optimized using lightmap atlasing and compression techniques. Combining multiple lightmaps into a single texture and employing compression formats that balance quality with memory usage minimized storage needs for lighting data, freeing up resources for other VR elements.

Light Probes for Dynamic Objects:

Light probes were incorporated for dynamic objects to accurately approximate illumination and reflections without real-time calculations. This was particularly useful in scenes with a mix of static and dynamic elements, reducing reliance on real-time lighting and boosting overall performance.

3.7.3. Objects optimization

A straightforward yet effective object optimization technique was used: all objects not intended to be moved during gameplay were designated as static. This adjustment significantly improved performance and stabilized frame rates. By marking objects as static, the Unity engine recognized that these elements would remain fixed in position throughout the game. This allowed Unity's optimization systems to precompute and optimize the rendering process for these objects, reducing GPU load and enhancing overall performance.

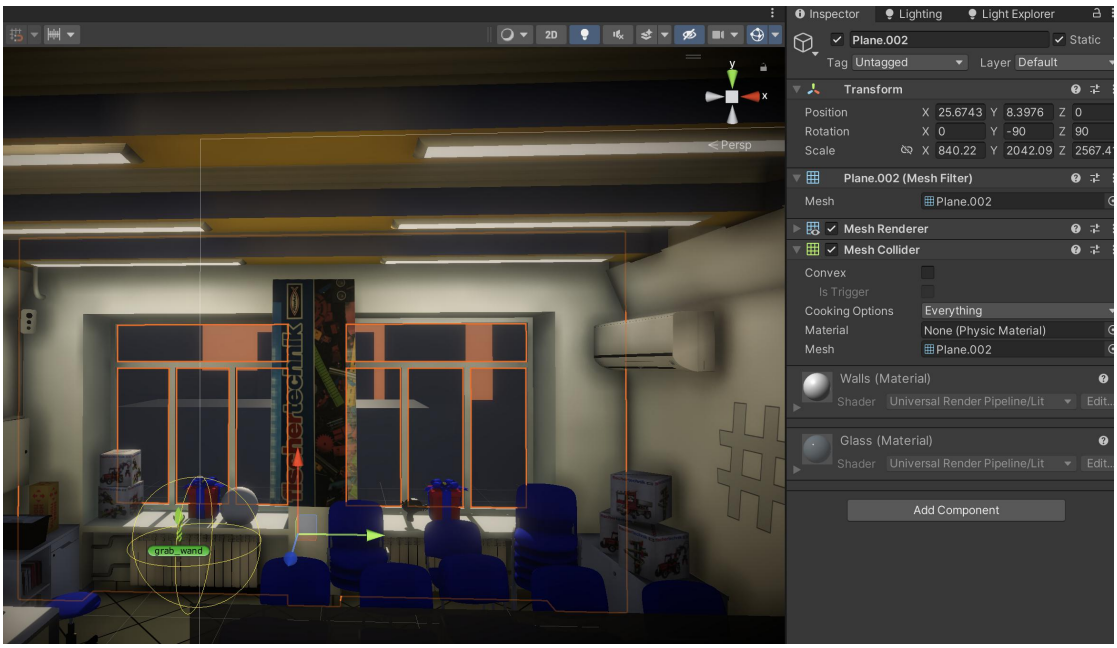

Image 3.35 – Static object

By applying this method, objects such as static environmental elements, buildings, and terrain were rendered efficiently, without additional overhead. This optimization played a crucial role in creating a smoother and more enjoyable virtual reality experience, enabling the system to allocate more resources to dynamic and interactive elements, like player interactions and real-time objects.

The decision to designate non-moving objects as static balanced visual quality and performance, contributing to a more immersive and fluid virtual reality environment. This strategy, along with other techniques focused on dynamic objects and rendering, facilitated the delivery of an optimized and engaging virtual reality experience, immersing users in a seamless virtual world.

3.8. Version control system

To manage the project effectively and keep a detailed log of code modifications, Git was integrated into the workflow. The chosen platform for hosting the Git repository was official github website[15]. For interacting with Git, GitHub Desktop was used as the client interface.

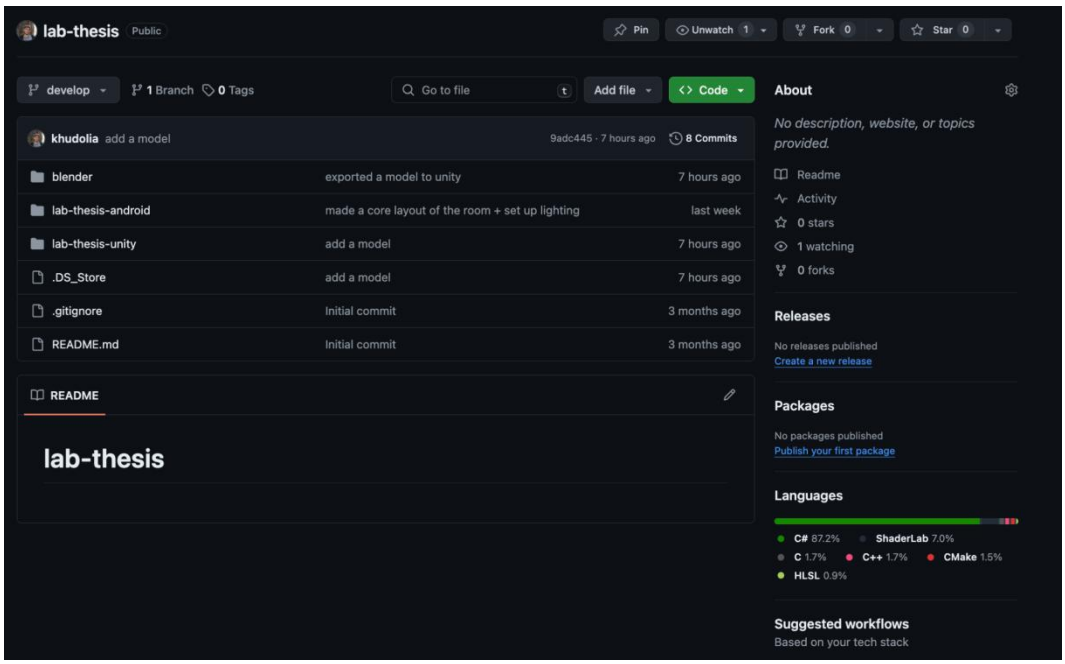

Image 3.36 – Github

The project started with the "git clone" command, which downloaded a copy of the repository from GitHub to a local computer. As modifications were made to the project, the "git add" command was used to stage selected files and changes for committing, allowing precise control over what changes would be included in the next commit. After staging the changes, the "git commit" command was used to permanently record these changes in the repository, typically with a brief message describing the modifications, added using the "-m" flag.

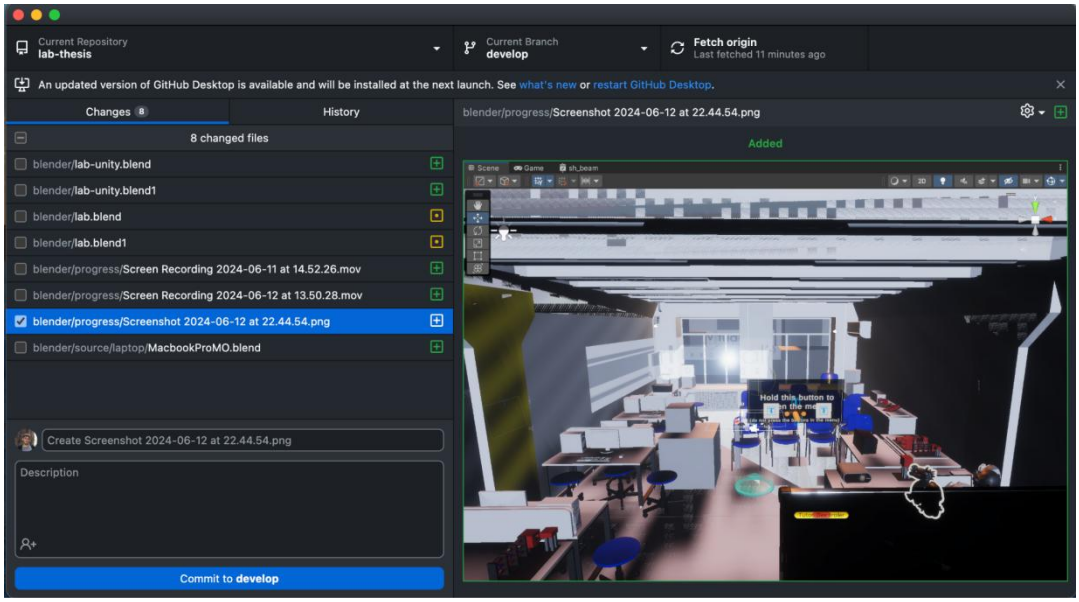

Image 3.37 – Github Desktop

To save changes, the "git push" command was used to upload the updates to the GitHub repository. When updates were available from other sources, the "git pull" command was used to fetch and merge these new changes into the local copy, keeping the local project synchronized with the latest developments.

3.9. Deploying app to Meta Quest 2

First, Oculus support should be enabled by navigating to the XR Plugin Management section in the Project Settings and selecting the Oculus checkbox under Plug-in Providers[8].

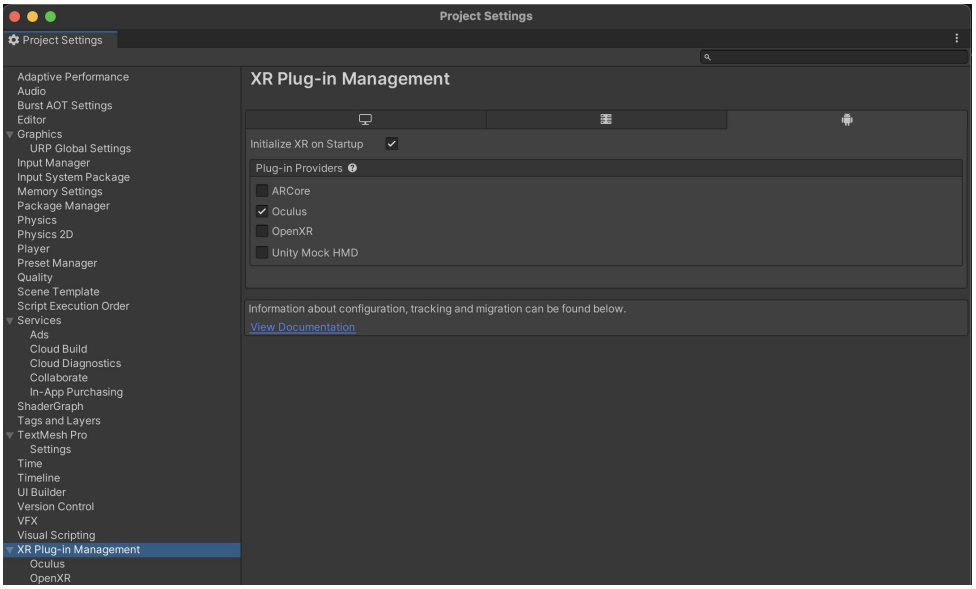

Image 3.38 – Enabling Oculus support

In the Player Settings, the Virtual Reality Supported option is activated, and Oculus is added to the list of supported devices. The graphics settings are also adjusted to optimize VR performance, by selecting a lower graphics tier.[Image 3.39].

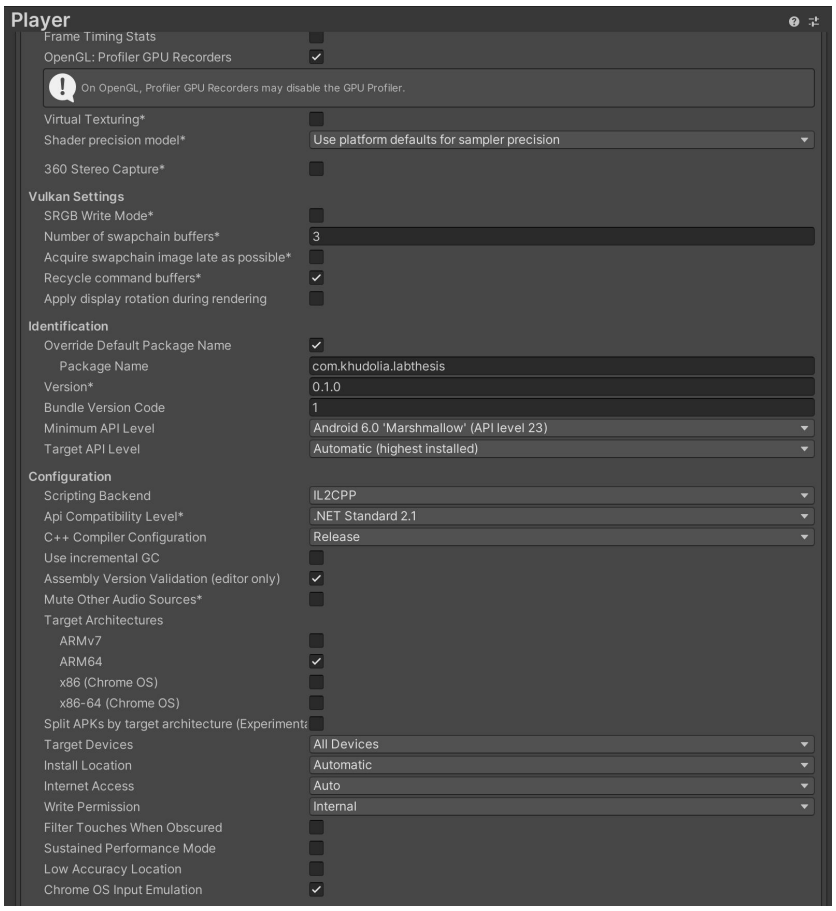

Image 3.39 – Changing graphic project settings

Before starting the development process, a Meta developer account needs to be created. This is crucial for enabling Developer Mode on the Meta Quest 2. The process begins by visiting the Meta for Developers website, where a new developer account is set up by registering with a Meta (formerly Facebook) account. During this setup, either a new organization is created or an existing one is joined. This step is necessary to activate Developer Mode on Meta devices.

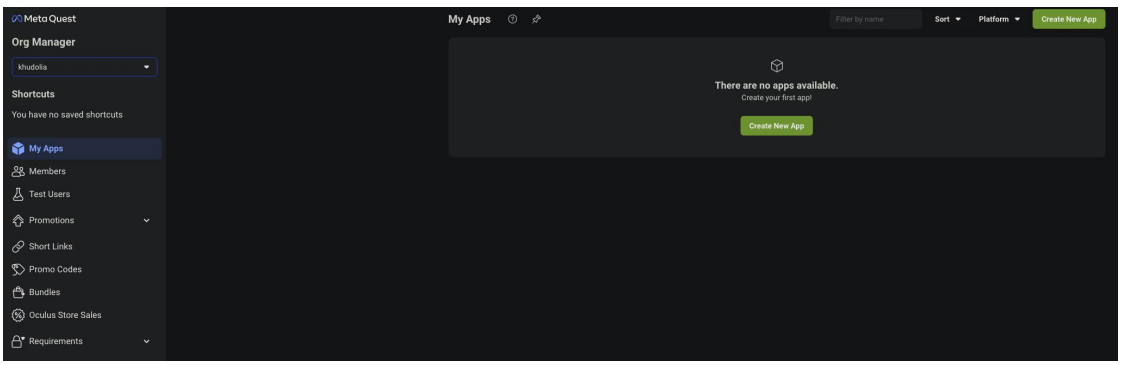

Image 3.40 – Organization setup for Meta Account

Then using the Oculus mobile app, Developer Mode was activated. This involved navigating to the device settings, selecting the connected Meta Quest 2, and switching on Developer Mode[Image 3.43].

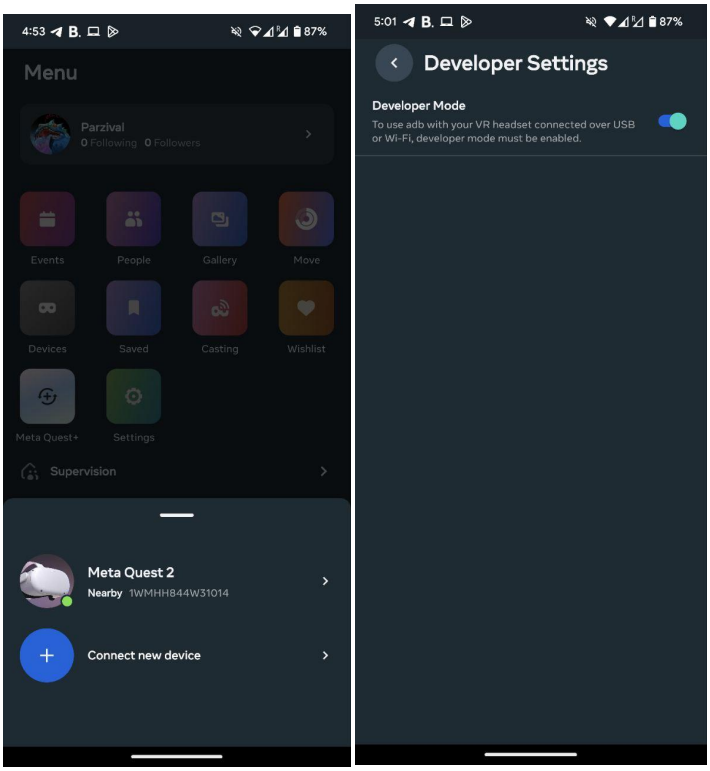

Image 3.41 – Enabling developer mode through an app

Then to prepare for building the APK, the project platform was switched to Android in the Build Settings. In the Player Settings under Other Settings, the Minimum API Level was set to at least 23 (Android 6.0), and the Target Architectures included ARM64. A unique Package Name was defined as com.khudolia.labthesis. The APK was then built by selecting the Build option in the Build Settings, specifying a location to save the APK file, and allowing the build process to complete[Image 3.42].

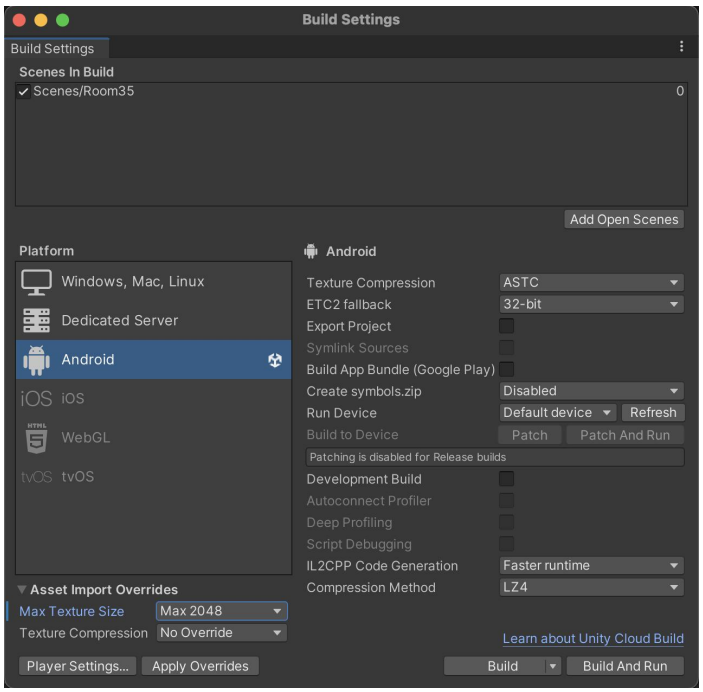

Image 3.42 – Build settings

In order to move the compiled .apk file to the Quest headset it's needed to use a 3-party app called SideQuest[14]. SideQuest was installed on a computer, and Developer Mode was enabled on the Meta Quest 2 through the Oculus mobile app. The Meta Quest 2 was then connected to the computer with a USB cable, and USB debugging was permitted on the headset. With SideQuest open, the "Install APK file from folder" button was used to locate and select the built APK file. SideQuest managed the sideloading, installing the APK onto the Meta Quest 2. The installation progress was visible through the SideQuest interface.

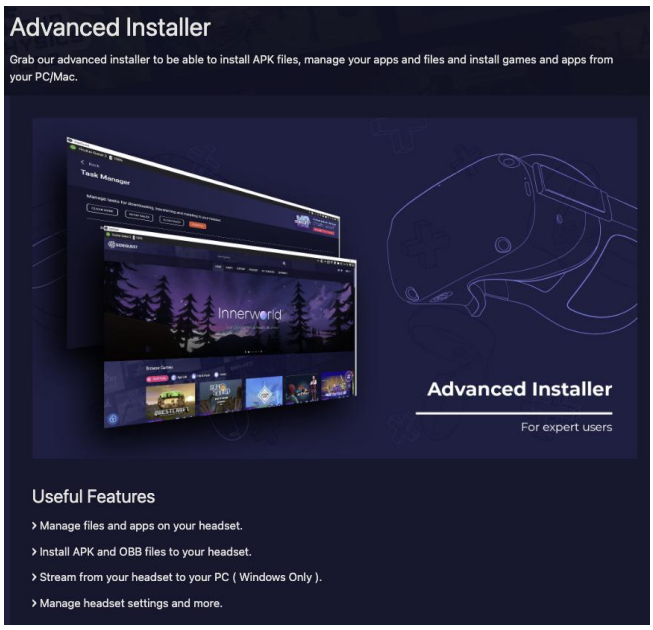

Image 3.43 – SideQuest app which was used

Once the installation was complete, the app was launched on the Meta Quest 2 by going to "Unknown Sources" in the app library and selecting the installed application, verifying that it functioned as expected.

CONCLUSION

This research comprehensively explores the potential and implementation of Virtual Reality (VR) technologies in educational settings, focusing particularly on the development and application of VR within cyber-physical laboratories. The study is divided into four key areas: an analysis of VR technologies, the preparation of materials, the development of VR environments, and the synthesis of findings.

The initial analysis of VR technologies highlighted their transformative potential in education. VR offers immersive and interactive learning experiences that significantly enhance student engagement and comprehension. By examining current trends and capabilities, the research identified the increasing accessibility and sophistication of VR technologies, making them a viable tool for modern educational methodologies.

In preparing materials for VR environments, the research underscored the importance of detailed digitization and modeling processes. The collection of photo data and the creation of 3D models using Blender software revealed the intricate nature of developing high-quality VR content. This phase emphasized the necessity of precision and thoroughness to ensure the realism and effectiveness of the VR simulations.

The development phase detailed the technical requirements and processes involved in creating a functional VR environment. The selection of appropriate software, configuration of the development setup, and adaptation of models for interactive VR use were critical steps. The addition of interactive elements and the optimization of the project for performance ensured that the VR environment was not only engaging but also efficient and user-friendly.

Finally, the synthesis of findings from all chapters demonstrated that the integration of VR into educational frameworks holds substantial promise. VR technologies can significantly enhance learning outcomes by providing realistic, hands-on experiences that traditional methods cannot offer. However, successful implementation requires careful planning, detailed development, and continuous optimization to address technical and logistical challenges.

In conclusion, this research validates the significant educational benefits of VR technologies. By meticulously analyzing, preparing, developing, and synthesizing VR

environments, this study contributes to the growing body of knowledge on the application of VR in education. The findings underscore the need for ongoing innovation and strategic integration of VR to fully realize its potential in enhancing educational experiences and outcomes.

LIST OF REFERENCES

1. Методичні рекомендації до виконання кваліфікаційної роботи бакалавра студентами галузі знань 12 Інформаційні технології спеціальності 123 Комп'ютерна інженерія / Л.І. Цвіркун, С.М. Ткаченко, Я.В. Панферова, Д.О. Бешта, Л.В. Бешта. – Д.: НТУ «ДП», 2023. – 62 с.

2. Iot for all (рекомендації) – [Електронний ресурс] – Режим доступу до ресурсу: https://www.iotforall.com/

3. Freina, L., & Ott, M. (2015). A literature review on immersive virtual reality in education: State of the art and perspectives. Proceedings of the 11th International Scientific Conference eLearning and Software for Education (eLSE), Bucharest, Romania.

4. Jensen, L., Konradsen, F., & Bødkesvang, T. (2018). A review of the use of virtual reality head-mounted displays in education and training. Education and Information Technologies, 24, 1515-1529.

5. Merchant, Z., Goetz, E. T., Cifuentes, L., Keeney-Kennicutt, W., & Davis, T. J. (2014). Effectiveness of virtual reality-based instruction on students' learning outcomes in K-12 and higher education: A meta-analysis. Computers & Education, 70, 29-40.

6. Radianti, J., Majchrzak, T. A., Fromm, J., & Wohlgenannt, I. (2020). A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. Computers & Education, 147, 103778.

7. Slater, M., & Sanchez-Vives, M. V. (2016). Enhancing Our Lives with Immersive Virtual Reality. Frontiers in Robotics and AI, 3, 74.

8. Oculus - Information on the latest VR hardware and software developments from a leading VR company. https://www.oculus.com/

9. Google Scholar - A broad database for finding academic papers on VR and education. https://scholar.google.com/

10. VRIF documentation: https://wiki.beardedninjagames.com/

11. https://www.linkedin.com/pulse/biorevolution-7-digital-twins-lab-elif-damla-karakolcu-acspf/ - Digital Twins in the Lab

12. https://physics.tntu.edu.ua/labs/cpsl/ - digital museum

13. https://www.fischertechnik.de/de-de/produkte/industrie-und-hochschulen/simulationsmodelle/554868-lernfabrik-4-0-24v - references of educational modules from company Fischertechnik.

14. SideQuest software - https://sidequestvr.com/.

15. Project's repository - https://github.com/khudolia/lab-thesis

APPENDIX A. FRAGMENT OF PROGRAM CODE

Brackable.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography;
using UnityEngine;

public class Brackable : MonoBehaviour
{
    public GameObject brokenObject;
    public float velocityMagnitudeThreshold = 8;

    public bool isTest = false;

    private void Awake()
    {
        if (isTest) velocityMagnitudeThreshold = 1;
    }

    public void Broke(GameObject triggerObject)
    {
        var spawnedObject = Instantiate(brokenObject, transform.position, transform.rotation);

        if(!isTest) spawnedObject.AddComponent<BrockenController>();
        //spawnedObject.transform.localScale = transform.localScale;

        AddVelocityToDestroyedObject(spawnedObject, triggerObject);
```

```
            Destroy(gameObject);
        }

        private void AddVelocityToDestroyedObject(GameObject spawnedObject, GameObject triggerObject)
        {
            var velocityFrom = gameObject.GetComponent<Rigidbody>().isKinematic ? triggerObject : gameObject;

            foreach (Transform child in spawnedObject.transform)
            {
                if(child.GetComponent<Rigidbody>() == null || spawnedObject.GetComponent<Rigidbody>() == null) return;
                child.GetComponent<Rigidbody>().velocity = velocityFrom.GetComponent<Rigidbody>().velocity;
            }

        }

        void OnCollisionEnter(Collision collision)
        {
            if (collision.relativeVelocity.magnitude > velocityMagnitudeThreshold)
                Broke(collision.gameObject);
        }
    }
```

TutorialSequenceController

```
using System.Collections;
using System.Collections.Generic;
using Helpers;
```

```
using Tutorial;
using UnityEngine;

public enum TutorialState
{
    None,
    PressAllButtons,
    //Menu,
    Walk,
    GrabObject,
    Finish
}

public class TutorialSequenceController : MonoBehaviour
{
    public TutorialState state;

    [Header("States UI")] public GameObject useUI;
    public GameObject finishUI;
    public List<GameObject> particles;

    [Header("Controllers")] public HandModelSelector handModelSelector;
    private TutorialState _previousState;
    private GameObject _previousActivatedUI;

    [Header("Settings")] public bool autoStart = false;

    private PressAllButtonsStep _allButtonsStep;
    private MoveStep _moveStep;
    private GrabAndUseObjectStep _grabAndUseObjectStep;
```

```
private MenuStep _menuStep;

private bool _isCanceled = false;
private void Start()
{
    _allButtonsStep = GetComponent<PressAllButtonsStep>();
    _moveStep = GetComponent<MoveStep>();
    _grabAndUseObjectStep = GetComponent<GrabAndUseObjectStep>();
    _menuStep = GetComponent<MenuStep>();

    useUI.SetActive(false);

    handModelSelector.SwapToHands();

    if(autoStart)
        StartTutorial();
}

void Update()
{
    if(_isCanceled) return;

    if (state != _previousState)
    {
        switch (state)
        {
            case TutorialState.PressAllButtons:
                handModelSelector.SwapToControllers();
                StartAllButtonsTutorial();
                break;
```

```
            // case TutorialState.Menu:
            //     StartMenuTutorial();
            //     break;
            case TutorialState.Walk:
                StartWalkTutorial();
                break;
            case TutorialState.GrabObject:
                handModelSelector.SwapToHands();
                StartGrabTutorial();
                break;
            case TutorialState.Finish:
                StartCoroutine(DisplayFinishMessage());
                break;
        }


        _previousState = state;
    }
}

public void StartTutorial()
{
    _isCanceled = false;
    state = TutorialState.PressAllButtons;
}

public void CancelTutorial()
{
    _isCanceled = true;
    useUI.SetActive(false);
```

```
    _menuStep.FinishTutorial();
    _moveStep.FinishTutorial();
    _grabAndUseObjectStep.FinishTutorial();

    handModelSelector.SwapToHands();
    state = TutorialState.None;
}

private void StartAllButtonsTutorial()
{
    _allButtonsStep.StartTutorial();
    ActivateUI(null);
}

public void FinishAllButtonsTutorial()
{
    state = state.Next();
}

private void StartWalkTutorial()
{
    _moveStep.StartTutorial();
    ActivateUI(null);
}

public void FinishWalkTutorial()
{
    state = state.Next();
}
```

```
private void StartGrabTutorial()
{
    ActivateUI(null);
    _grabAndUseObjectStep.StartTutorial();
}

public void FinishGrabTutorial()
{
    state = state.Next();
}

private void StartMenuTutorial()
{
    _menuStep.StartTutorial();
}

public void FinishMenuTutorial()
{
    state = state.Next();
}

private IEnumerator DisplayFinishMessage()
{
    foreach (var particle in particles)
        particle.SetActive(true);

    finishUI.SetActive(true);
    yield return new WaitForSeconds(5);
```

```
        foreach (var particle in particles)
            particle.SetActive(false);

        finishUI.SetActive(false);
    }

    private void ActivateUI(GameObject ui)
    {
        if (_previousActivatedUI != null)
            _previousActivatedUI.SetActive(false);

        if (ui != null)
            ui.SetActive(true);

        _previousActivatedUI = ui;
    }
}
```

DecalProjector

```
using UnityEngine;
using UnityEngine.Rendering.Universal;

public class DecalProjectorController : MonoBehaviour
{
    public float minSize = 1f; // Minimum size of the projector
    public float maxSize = 5f; // Maximum size of the projector
    public float minPivotZ = -0.5f; // Minimum pivot Z position
    public float maxPivotZ = 0.5f; // Maximum pivot Z position

    private DecalProjector decalProjector;
```

```
    private RaycastHit hitInfo;

    private float _aspectRatio;
    private float _aspectRatioZ;

    private void Start()
    {
        decalProjector = GetComponent<DecalProjector>();

        _aspectRatio = decalProjector.size.x / decalProjector.size.y;
        _aspectRatioZ = decalProjector.size.z / decalProjector.size.y;
        if (decalProjector == null)
        {
            Debug.LogError("DecalProjector component not found!");
            return;
        }
    }

    private void Update()
    {
        // Send a ray forward from the decal projector
        if (Physics.Raycast(transform.position, transform.forward, out hitInfo))
        {
            float distance = hitInfo.distance;

            // Calculate the normalized distance between the minimum and maximum distance
            float normalizedDistance = Mathf.Clamp01(distance / maxSize);

            // Calculate the new size based on the normalized distance
```

```
            float newSize = Mathf.Lerp(minSize, maxSize, normalizedDistance);

            // Update the size of the decal projector
            decalProjector.size = new Vector3(newSize * _aspectRatio, newSize , newSize * _aspectRatioZ);

            // Update the pivot Z position of the decal projector
            decalProjector.pivot = new Vector3(decalProjector.pivot.x, decalProjector.pivot.y, newSize/2 + gameObject.transform.localScale.z /2 + 14f);
        }
    }
}
```

VrInitializer

```
using System.Collections;
using UnityEngine;
using UnityEngine.XR.Management;

public class VRInitializer : MonoBehaviour
{
    private void Start()
    {
        EnableXR();
    }
    private void OnDestroy()
    {
        DisableXR();
    }
    public void EnableXR()
    {
```

```
            StartCoroutine(StartXRCoroutine());
        }
        public void DisableXR()
        {
            Debug.Log("Stopping XR...");
            XRGeneralSettings.Instance.Manager.StopSubsystems();
            XRGeneralSettings.Instance.Manager.DeinitializeLoader();
            Debug.Log("XR stopped completely.");
        }

        public IEnumerator StartXRCoroutine()
        {
            Debug.Log("Initializing XR...");
            yield return XRGeneralSettings.Instance.Manager.InitializeLoader();

            if (XRGeneralSettings.Instance.Manager.activeLoader == null)
            {
                Debug.LogError("Initializing XR Failed. Check Editor or Player log for details.");
            }
            else
            {
                Debug.Log("Starting XR...");
                XRGeneralSettings.Instance.Manager.StartSubsystems();
            }
        }
    }
```

ArcController

```
using UnityEngine;
```

```
using wands.grab;

public class ArcController : MonoBehaviour
{
    [Header("Beam Parameters")] public Transform startBeam;
    public Transform beam2;
    public Transform beam3;
    public Transform endBeam;
    public Transform pointerBeam;

    public Transform startTarget;

    private GrabbableController _grabbableController;

    private bool _isObjectSelected;

    private void Awake()
    {
        _grabbableController = gameObject.GetComponent<GrabbableController>();
    }

    private void Update()
    {
        startBeam.position = startTarget.position;

        var rayOutput = _grabbableController.RayOutput;
        _isObjectSelected = _grabbableController.selectedObject != null;

        UpdateHoverBeam(Vector3.Distance(rayOutput.point, transform.position));
```

```
            if (_isObjectSelected)
            {
                UpdateHoverBeam(.0f);
                SetEndPosition(_grabbableController.selectedObject.transform.position);
                gameObject.GetComponent<GrabWand>().beam.SetActive(true);
            }
            else
            {
                gameObject.GetComponent<GrabWand>().beam.SetActive(false);
            }
        }

        private void SetEndPosition(Vector3 position)
        {
            endBeam.position = position;

            var wandPosition = transform.position;
            var targetPointerPosition = _grabbableController.targetPointer.transform.position;
            beam2.position = _CalculateQuadraticBezierPoint(.3f, wandPosition, targetPointerPosition, position);
            beam3.position = _CalculateQuadraticBezierPoint(.6f, wandPosition, targetPointerPosition, position);
        }

        private void UpdateHoverBeam(float distance)
        {
            pointerBeam.localScale = new Vector3(pointerBeam.localScale.x, distance, pointerBeam.localScale.z);
            pointerBeam.transform.localPosition = new Vector3(0, distance, 0);
```

```
        }

        private Vector3 _CalculateQuadraticBezierPoint(float t, Vector3 p0, Vector3 p1,
Vector3 p2)
        {
            float u = 1 - t;
            float tt = t * t;
            float uu = u * u;
            Vector3 p = uu * p0;
            p += 2 * u * t * p1;
            p += tt * p2;

            return p;
        }
    }
```

GrabbableController

```
using System;
using UnityEngine;

namespace wands.grab
{
    public class GrabbableController : MonoBehaviour
    {
        public GameObject raycastPointer;
        public GameObject targetPointer;
        public RaycastHit RayOutput;

        [HideInInspector] public GameObject selectedObject;
        private GameObject _lastHittedObject;
```

```
        private float _targetObjectPointDistance;
        private float _dragVelocity = 1f;
        private void Awake()
        {
            GetComponent<GrabWand>().OnThumbstickAxisCallback = OnThumbstickAxisCallback;
        }

        void FixedUpdate()
        {
            var ray = new Ray(raycastPointer.transform.position, raycastPointer.transform.forward);

            if (Physics.Raycast(ray, out RayOutput, 1000))
            {
                OnObjectCollided(RayOutput.transform.gameObject);
            }
            else
            {
                if (!IsGrabbing())
                {
                    ReleaseLastObject();
                    _lastHittedObject = null;
                    selectedObject = null;
                }
            }

            if(targetPointer.transform.localPosition.y != _targetObjectPointDistance)
```

```
            targetPointer.transform.localPosition = new Vector3(0,
_targetObjectPointDistance, 0);

        if (selectedObject != null)
        {
            float distance =
                Vector3.Distance(selectedObject.transform.position,
targetPointer.transform.position);
            if (Vector3.Distance(selectedObject.transform.position,
targetPointer.transform.position) < 0.2f)
            {
                selectedObject.GetComponent<Rigidbody>().velocity = Vector3.zero;
                selectedObject.GetComponent<Rigidbody>().angularVelocity =
Vector3.zero;

                selectedObject.transform.position = targetPointer.transform.position;

                _dragVelocity = 1f;
                return;
            }

            var movingVector = (targetPointer.transform.position -
selectedObject.transform.position).normalized;
            Vector3 dir = movingVector * 8f * _dragVelocity;

            _dragVelocity += .008f;
            selectedObject.GetComponent<Rigidbody>().velocity = dir;
        }
    }
```

```
        // GameObject here is always != null
        private void OnObjectCollided(GameObject hittedObject)
        {
            if (!IsGrabbing() && selectedObject != null) selectedObject = null;

            if (selectedObject != null) return;


            // The wand is pointed to the new object
            ReleaseLastObject();

            if (hittedObject.GetComponent<Rigidbody>() == null ||
hittedObject.CompareTag("Ungrappable"))
            {
                _lastHittedObject = null;
                return;
            }

            ControlSelect(hittedObject, SelectableType.Hover);


            if (IsGrabbing()) OnWandReadyToGrab(hittedObject);
            else UpdateObjectPhysicsParameters(hittedObject, true);

            _lastHittedObject = hittedObject;
        }

        private void OnWandReadyToGrab(GameObject o)
        {
```

```
            SetTargetObjectPointDistance(Vector3.Distance(o.transform.position,
gameObject.transform.position));

            selectedObject = o;

            UpdateObjectPhysicsParameters(selectedObject, false);
            ControlSelect(selectedObject, SelectableType.Selected);
        }

        private void ReleaseLastObject()
        {
            UpdateObjectPhysicsParameters(_lastHittedObject, true);
            ControlSelect(_lastHittedObject, SelectableType.Unselected);
        }

        private void UpdateObjectPhysicsParameters(GameObject gameObject, bool
isPhysics)
        {
            if (gameObject == null || gameObject.GetComponent<Rigidbody>() == null)
return;

            gameObject.GetComponent<Rigidbody>().useGravity = isPhysics;
        }

        private void ControlSelect(GameObject gameObject, SelectableType type)
        {
            // Removed due to materials overlapping on some objects.
            //
            // Need to be fixed in the future
            //
```

```
            /* if (gameObject == null || gameObject.GetComponent<Rigidbody>() ==
null) return;

            if (gameObject.GetComponent<Selectable>() == null)
                gameObject.AddComponent<Selectable>();

            gameObject.GetComponent<Selectable>().UpdateStatus(type);*/
        }

        private void SetTargetObjectPointDistance(float position)
        {
            _targetObjectPointDistance = Math.Clamp(position, 2f, 10000);
        }

        private void OnThumbstickAxisCallback(Vector2 input)
        {
            SetTargetObjectPointDistance(_targetObjectPointDistance + input.y * .5f);
        }

        private bool IsGrabbing() => GetComponent<GrabWand>().isActive;
    }
}
```