Projecting Art into Virtual Reality

Creating artistic scenes through parametrization utilizing a modern game-engine

Runar Tistel

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Western Norway University of Applied Science

Department of Informatics,
University of Bergen

June 2018

Western Norway
University of
Applied Sciences

Abstract

Recent advancements in virtual reality both on the hardware and software front have made high-quality virtual reality experiences both cheaper, and easier to obtain. This thesis aims to explore how virtual reality can be used as a new medium for digital artists, and how virtual reality as a medium changes how a user experiences art. A successful attempt is made to create a solution for creating and parameterizing artistic scenes inside virtual reality through a visual node-based scripting language. This thesis presents the results of applying this solution to two works of art by Hariton Pushwagner. In addition, a secondary solution attempts to translate The Persistence of Memory by Salvador Dali into an interactive virtual reality experience. An analysis of response from a public viewing session of the scenes resulting from the solution is provided. The virtual reality scenes produced by the solution transmit scale, space, and movement particularly well. Several respondents reported that virtual reality as a medium placed them in an active viewing position, creating a very different experience in comparison to viewing the original two-dimensional artwork.

Acknowledgements

I would like to thank my supervisors at Western Norway University of Applied Science (HVL), Harald Soleim, Atle Birger Geitung, and Daniel Patel for their guidance and support throughout my work on this thesis. Their frequent feedback and assistance has helped me greatly, and without their contribution, this thesis would not have been possible. I would also like to send a heartfelt thanks to Fredrik Malmedal Salhus and Torkell Bernsen from the Faculty of Fine Art, Music and Design, University of Bergen (UiB). They have contributed heavily to this thesis through their support, and insight into the world of modern art. They also helped organize the public viewing session of our results at their faculty, and helped spread the word about my work. Their involvement is greatly appreciated. Finally i would like to express my condolences to the family and friends of Hariton Pushwagner, who sadly passed away 24th April 2018, during my work on this thesis.

Contents

List of Figures

Listings

Glossary

Filmbox Proprietary file format, used to store 3D object data together with motion and other metadata.. 31

Wavefront .obj Geometry definition file format. The format is open, and has been adopted by most 3D graphics applications in one form or the other.. 31

Acronyms

2D 2 Dimensional. 2, 12, 23, 60, 62, 63, 65–67, 69, 70

3D 3 Dimensional. 2, 3, 12, 13, 23, 29, 30, 39, 40, 42, 44, 47, 50, 57, 60, 63, 66–71

6-DOF 6 Degrees Of Freedom. 24, 25

API Application Programming Interface. 20, 21

CPU Central Processing Unity. 17, 18, 56

CSG Constructive Solid Geometry. 70

CV1 Consumer Version 1. 27

FPS Frames Per Second. 55, 56, 68

GPU Graphics Processing Unit. 16–19, 30

HISMC Hierarchical Instanced Static Mesh Component. 33

HMD Head Mounted Display. 8–11, 24, 26, 27, 55, 62, 70

HUD Head-Up Display. 55

IR Infrared Radiation. 25

LOD Level of Detail. 19, 56, 68

MTPL Motion to Photon Latency. 55, 56

OO Object Oriented. 22

RMC Runtime Mesh Component. 31, 33, 35, 37, 38

UE Unreal Engine. 22, 23, 29–31, 34, 42, 50

VR Virtual Reality. 2, 3, 5, 8–10, 13, 24, 26, 27, 29, 30, 44, 46, 47, 53–56, 59–71

Chapter 1

Introduction

1.1 Thesis Outline

Introduction Provides a brief and concise introduction to the thesis. Some context is given for the work that has been done. The goal of the written thesis is conveyes, and and an overview of the related work is provided.

Background The needed background information for this thesis is described. Information surrounding technologies used is provided, including the implementation and usage of virtual reality.

Solution The solutions that were attempted in this thesis are outlined and described.

Result The result of the scenes produced by our solution are described.

Evaluation After displaying the results, an evaluation is performed based on performance data. We also evaluate the ease of which we are able to parametrize a work of art, and the re-create it in 3 dimensions. We also evaluate user feedback from a public demo, and how other users experience our results.

Conclusion Based on our result and evaluation, a conclusion is made, answering our research questions stated in section 1.4.

1.2 Motivation

Usually when a classical artist beings a new work, he is restricted to the media available to him. The media can for instance be some form of colored particles (charcoal, ink, oil paint, etc.) projected onto a two-dimensional base. Even three-dimensional scenes are usually projected into two dimensions using perspective projection. Virtual Reality (VR) is maturing both in its hardware and software aspects, new avenues are opening for digital artists. We want to see how we can employ VR for representing art. We want to use the computer and its graphical abilities as the medium to present spatial art. This is done by combining the power of modern hardware computation with the immersive imaging of head mounted VR displays. Utilizing the power of a fully featured game engine found in Unreal Engine 4.0 [1], an attempt is made to construct complex 3D environments using techniques such as parametric design and hardware instancing.

1.3 Goal

The goal with this work is to expand classical and modern art, from two-dimensional paintings, to 3 Dimensional (3D) scenes. This is done by re-projecting existing 2 Dimensional (2D) images into 3D. In addition, we want to add new dimensions to the work, by adding animation(temporal dimension), and sound. The ambitious goal was to create a high level abstracted meta-language for describing art. With this language it should be possible to create 3D art in VR by describing the shapes and visual style of the artwork in a compact parametric way. To test the expressability of the language, some known works of art should be re-created inside 3D. Successfully recreating the art in 3D, will demonstrate that the method is powerful enough for it's intended purpose. As a secondary goal, another method will be used to attempt recreating a known work of art into VR. This method should attempt the recreation of a work of art, that differs from the other works that have been attempted in both shape and visual style.

1.4 Research Questions

For this thesis, two research questions are asked:

1. How can a two dimensional work of art be interpreted through virtual reality?
2. What happens when a two dimensional work of art is transformed through these processes?

1.5 Art

In this thesis, a few works of art are referenced. The artists behind these works are Hariton Pushwagner[2], and Salvador Dali[3].

1.5.1 Hariton Pushwagner

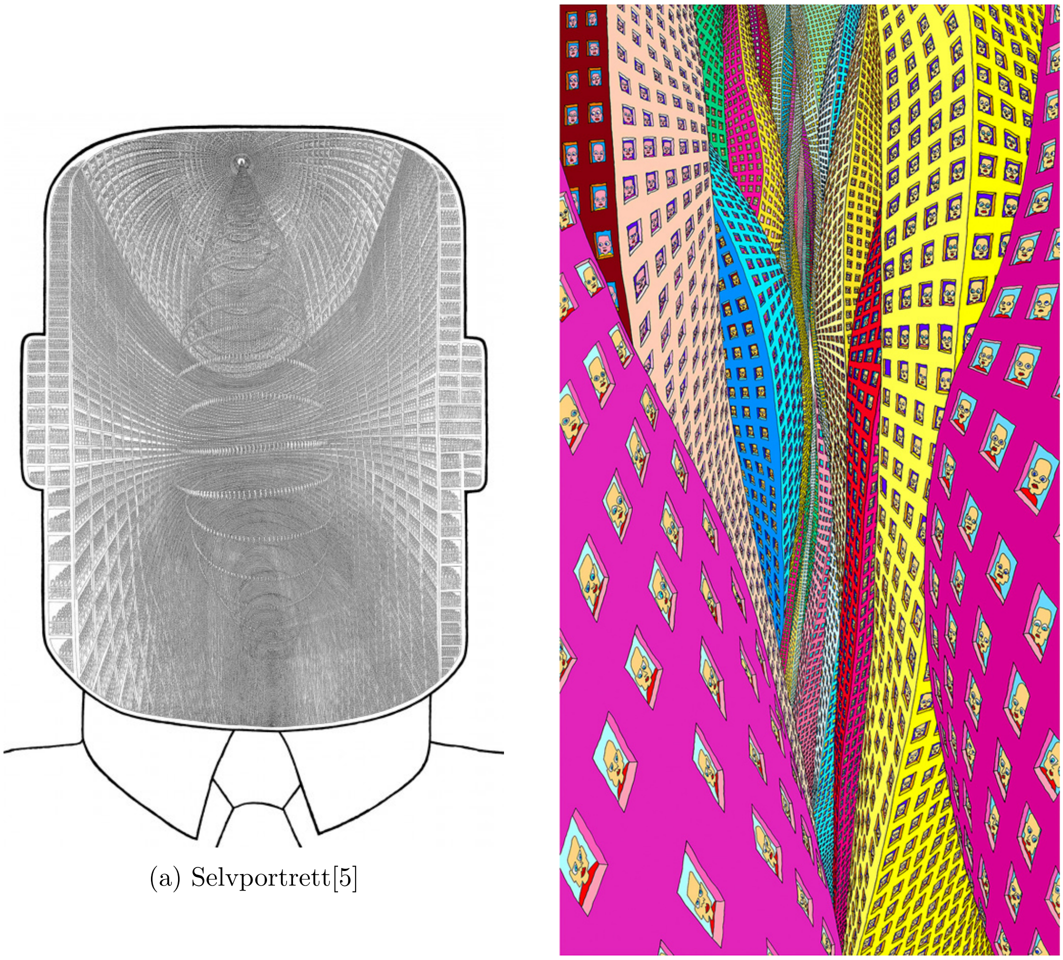
Hariton Pushwagners real name was Terje Brofos[2], and was born May 2, 1940 in Norway and died April 24, 2018. His art is known as Pop art[4]. Pop art grew as an art around 1950, and take inspiration from everyday items and consumerist goods that could be found at the time. The style can be described as bold, with strong colors. Hariton Pushwagners works of art often have strong colors, and portray surreal (not to be confused with surrealism) scenes that often contain repetitive elements on a large scale. Examples of his art can be seen in Figure:1.1a and Figure:1.1b.

1.5.2 Salvador Dali

The full name of Salvador Dali is Salvador Felipe Jacinto Dali y Domenech. He was born May 11, 1904 and died January 23, 1989. Salvador Dali was known as a surrealist artist, and is viewed as an important contributor to this artistic movement. His artworks often depict scenes where items or objects are deformed in ways that seem illogical, while still keeping a large amount of realistic detail in his artworks.

1.5.3 Artworks Used in Thesis

To demonstrate the capabilities of our solution we need to choose a small set of traditional artworks. These works of art will be the reference when attempting to re-create art in 3D. For this purpose two artworks are chosen: Selvportrett[5] (see Figure:1.1a) and Manhattan[6] (see Figure:1.1b). These works of art were chosen because of their structure and visual style. Both images are very spatial, in that they portray a large expansive space. This portrayal of a large space will most likely translate pretty well into VR, as one of the largest improvements that VR has over traditional media, is the sense of space and scale it provides. The artworks also feature heavily repeated elements. If an image or work of art is more uniform, it simplifies the process of describing the artwork, and re-creating it as a scene in VR. An original scene will also be created, after re-creating these two works of art.



(a) Selvportrett[5]

(b) Manhattan[6]

Figure 1.1: Two works of art by Hariton Pushwagner displayed side by side. (a): Selvportrett depicts a scene of many people spread over several floors, with a spiral of humans stretching from the bottom to the top of the image. The top is domed and covered with images of faces. (b): Manhattan depicts several buildings in bright colors, that seem to bend and sway in un-natural ways.

For the secondary method, a classic work of art by Salvador Dali named The Persistence of Memory[7] will be used. The three works of art (Figures: 1.1a , 1.1b, 1.2) are also included as larger versions in appendix D.

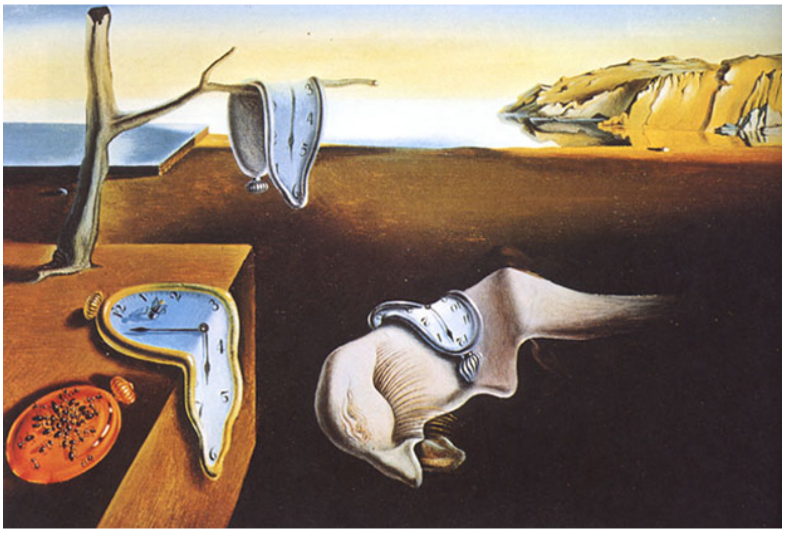


Figure 1.2: Persistence of Memory by Salvador Dali.[7] The image depicts a scene where an everyday objects (clocks) seem to melt as if they where soft, while in reality these objects would be rigid and stiff.

1.6 Related Work

Art outlets have shown interest in VR since it's early days. It is no surprise that projects have been developed before, with the intent of testing virtual reality as an artistic tool.

1.6.1 Before the Computer Age

The ideas behind taking traditional art beyond the common two-dimensional canvas has existed for years before the arrival of modern computer power. Back in 1792, Robert Barker painted the worlds first panorama painting[8], and created the term by naming the work "The Panorama". The artwork depicted a full 360°view of London. A year after, Robert Barker moved his paintings to Leicester Square, where a purpose-built panorama building would house his paintings (see Figure:1.3). The building would hold two large panoramas, both housed in separate compartments illuminated from the roof. These circular housings would allow the viewer to stand on a platform in the middle, and look around at an uninterrupted 360°view of the painted surroundings. The entire idea of this panoramic scene was to present a believable illusion to the viewer. Anyone

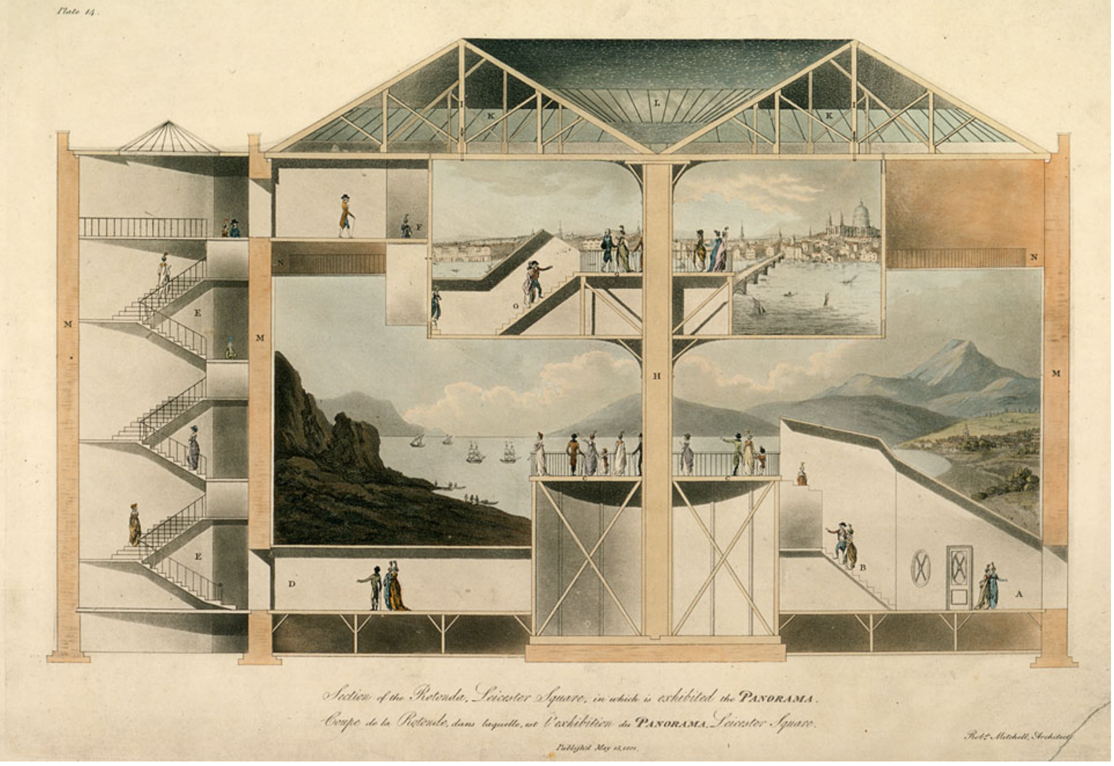


Plate 14.
L
K
K
I
I
N
N
M
M
M
E
E
F
G
H
D
B
A
Section of the Rotunda, Leicester Square, in which is exhibited the PANORAMA.
Coupe de la Rotonde, dans laquelle, est l'exhibition du PANORAMA, Leicester Square.
Published May 13, 1801.
Robt. Mitchell, Architect

Figure 1.3: Cross section of the planned Panorama.[9]

standing in the middle would view the art, as if they were standing inside the depicted scene.

Attempts in Motion Picture

A heightened sense of immersion has been sought after in the film industry for years. 3D cinema using stereoscopic image projection and higher quality surround sound systems are just some examples. A more obscure piece of history is the Sensorama [10] invented by Heilig Morton in 1962 (see Figure:1.4). The sensorama attempted to give viewers a more immersive movie experience by stimulating as many senses as possible. Electric fans provided gusts of wind, and it even employed aromas and vibrating motion in its attempt to bring the user a heightened sense of presence. The sensorama never became a very popular motion picture system, but it's attempt inspired later attempts of similar 4D cinema[11] experiences.

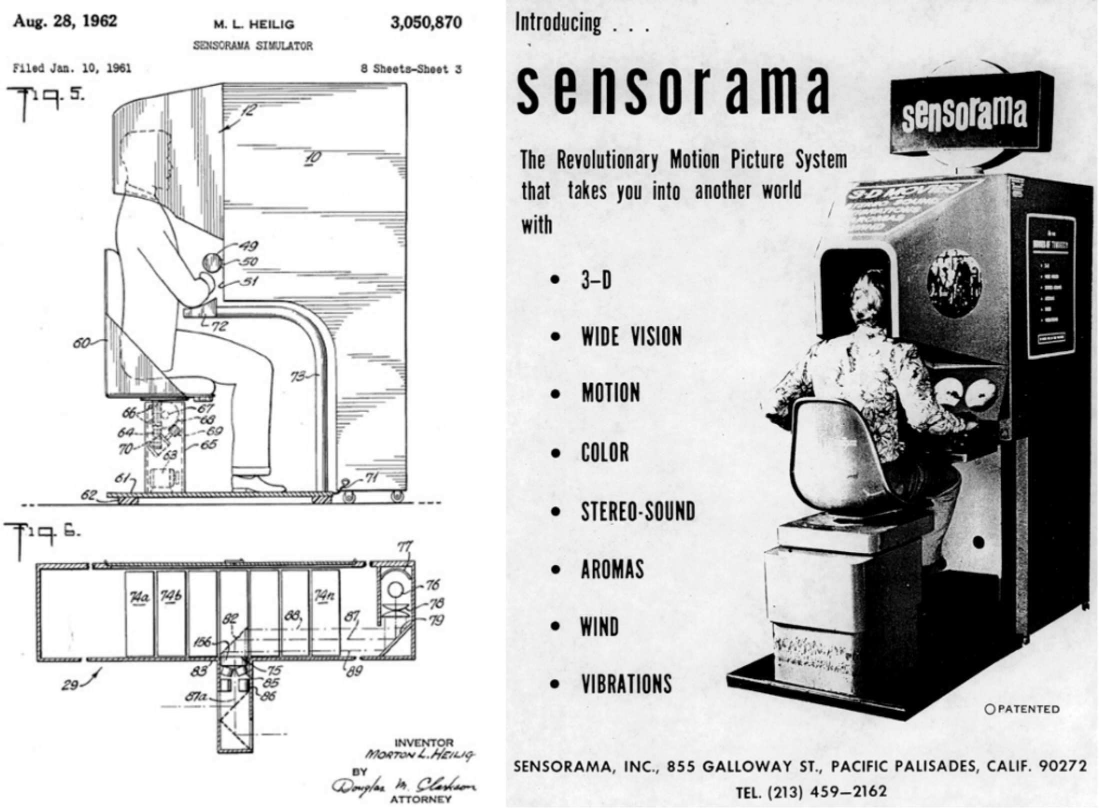


Figure 1.4: Commercial for the sensorama.[12]

1.6.2 The Sword of Damocles

Ivan Sutherland[13] is regarded as a pioneer within computer graphics and VR. His work of creating the first VR system in 1968, is considered to be the first Head Mounted Display (HMD) system ever attempted. This system, called The Sword of Damocles[14] by Ivan, incorporated many of the same techniques seen in modern HMD systems, like head-tracking and stereoscopic display. This early system however, was so heavy and mechanical, that the HMD unit required to be suspended in the air through a ceiling mounted mechanical arm.

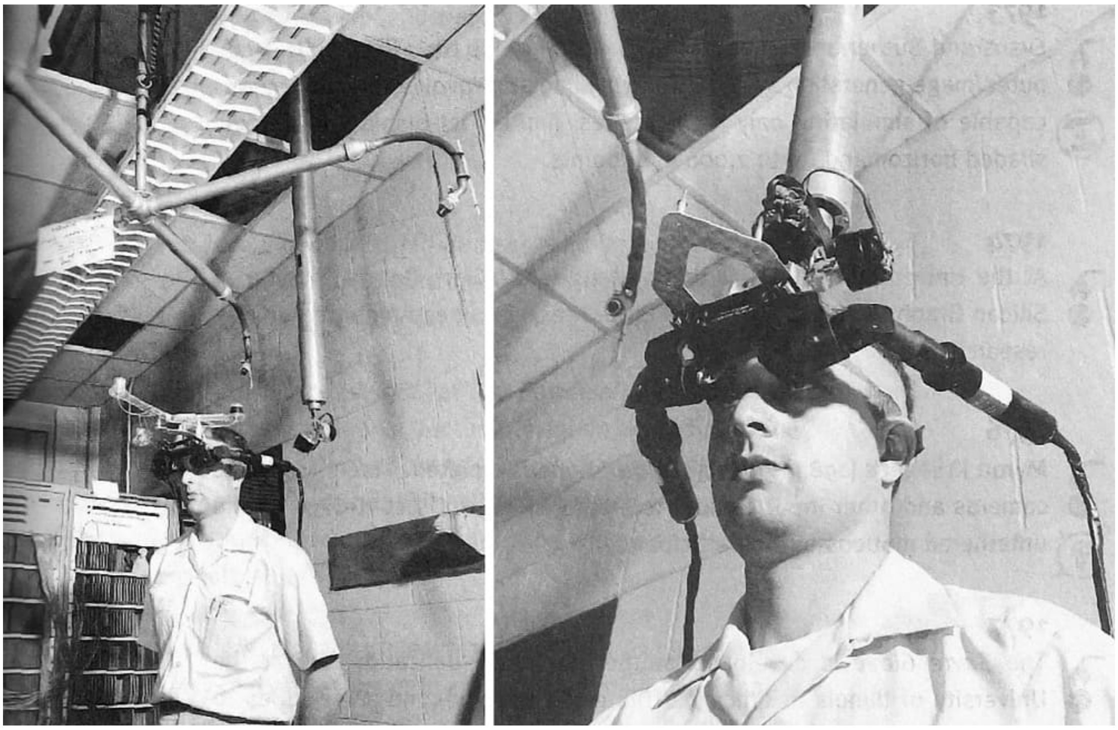


Figure 1.5: Image of the head mounted display HMD, Sword of Damocles. Invented in 1968 by Ivan Sutherland.[15]

1.6.3 CAVE

The CAVE project is a later attempt at virtual reality. The intent of CAVE was to create an immersive 3D environment. The CAVE concept is not based upon head mounted displays, but instead uses the surrounding walls of a square room combined with several projectors, to create the illusion of a 3D environment. The CAVE project requires extensive calibration at setup, but when calibrated, is capable of tracking the position of the user through electromagnetic, or optical sensors. The CAVE project has mostly been applied in an engineering environment, used to enhance product design and development. Figure 1.6 demonstrates a CAVE system, being used to assist the planning of new oil wells.

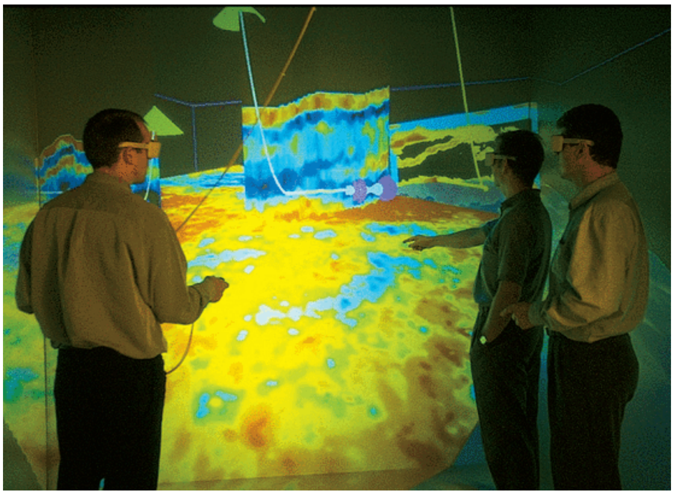


Figure 1.6: Example of CAVE in use by Hydro VR.[16]

1.6.4 Virtual Reality And Modern Art

Since the release of the two HMD units, Oculus Rift and HTC VIVE in 2016, the respective companies have started an initiative to bring better VR experiences to the general public. Some modern museums hold free and open VR art installations. These installations provide a new way to experience the museums content. An example of such an art installation is the VR Museum of Fine Art[17]. This VR application presents a fully explorable 3-dimensional environment, meant to represent a museum. The virtual museum contains real life scale digital replicas of works of art found in the Museum of Fine Arts Boston. These museums or institutions utilize VR as a means to expand the availability of their art collection, making it available to an audience that might not have the ability to travel to the museums location. A great example of art where virtual reality was a key part, is the BMW Art Car from 2017.[18] These art projects are initiated by BMW, and entail painting a work of art on the exterior of a BMW car model. The newest art car from 2017, is painted by chinese multimedia artist Cao Fei[19]. This art car is special, as it is painted fully in VR. By utilizing VR as a tool, Cao Fei was allowed to paint on both the exterior surface of the car, and in the nearby space surrounding the car.

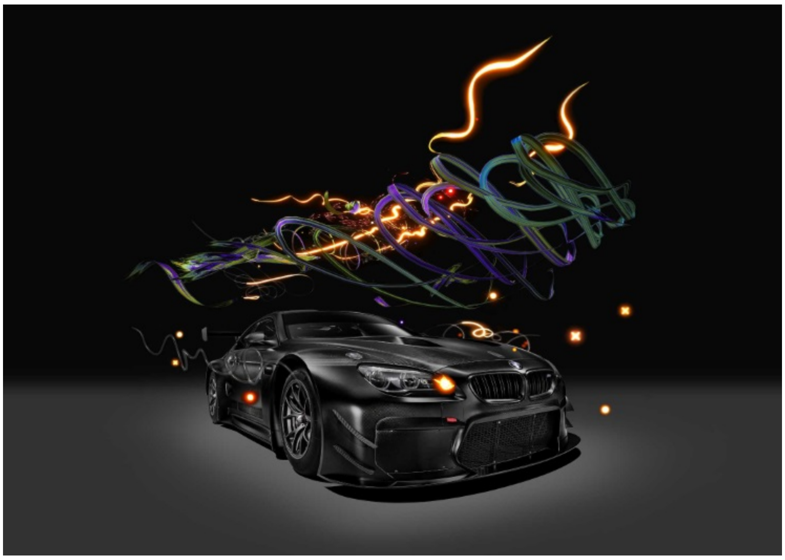


Figure 1.7: Image of the BMW 2017 art car produced by Cao Fei. By using VR as her tool, she was able to produce a unique art car, that stands out among the other BWM art car entries.[18]

1.6.5 Use of Virtual Reality Tools for Artists

There are already tools available for artists that want to work in a fully virtual environment. Tools like Tilt Brush[20], and Oculus Medium[21], give users the ability to paint and sculpt models in three dimensions. Figure 1.8 shows Tilt Brush in use. The users hands hold the tracked hand controllers, and use them as if they were digital brushes. While painting, the user also views their work inside VR using the Vive HMD. The Vive hand-controllers are multi functional, in that their function can be changed through interactive menus attached to the controllers inside VR. Settings like brush color, size, and type can be changed through menu interactions. For this thesis, a search was done for art tools or projects that are closely related to the goals and questions posed in section 1.3. Nothing beyond what is mentioned by this thesis was found, regarding parametrization of art combined with virtual reality. The most relevant examples are ”The Starry Night VR”[23] by Motion Magic VR[24], and ”Dreams of Dali”[25] by GS&P[26]. These VR applications both attempt to transport the viewer into known artworks by their respective artists (see Figure:1.9, and Figure: 1.10). It’s difficult to find any reliable information on how these solutions were implemented, but they appear to be hand made, and are therefore not relevant to our work.

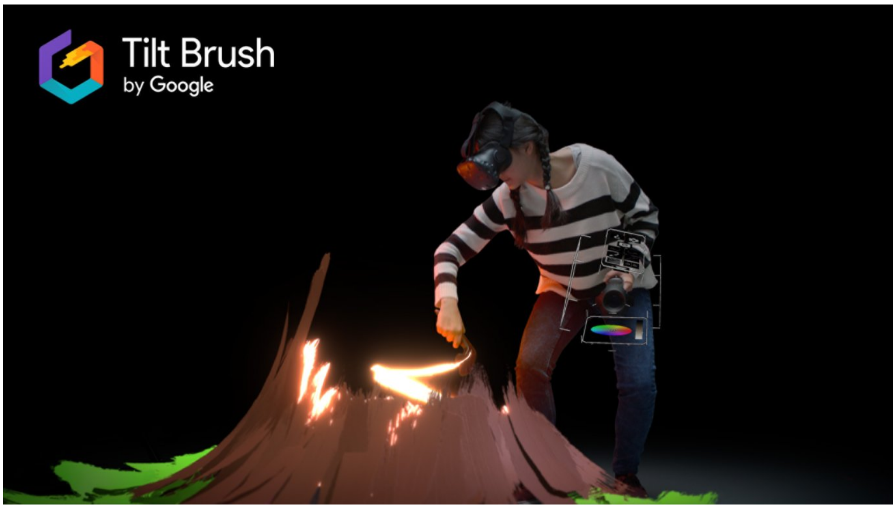


Figure 1.8: Example image taken from the HTC Vive homepages. The image displays a woman painting a sculpture using the positionally tracked controllers, while also viewing it through the Vive HMD unit.[22]

Figure 1.9: Still image from the video ”Dreams of Dali: 360 Video”[25]

Figure 1.10: Still image from the video ”The Starry Night VR”[23]

1.6.6 Generative Art

Generative art is described as works of art that emerge from a system that has some degree of autonomy, hence the name *generative*. In most cases the art is generated not by the artist, but by the system itself. The artists job in this case is setting the rules and constraints for the system. Generative art is a broad term,[27] and can be difficult to describe accurately. Over the years, many systems have been used that could be described as generative art, wherever it be procedural maps inside a video game, or images produced by an algorithm. Generative art is not required to have a random element in its production, meaning that our method of generating 3D scenes, also can be described as a form of generative art.

Processing

Casey Reas[28], the creator of Processing[29], is a popular generative artist. In addition to creating his own works of art (see Figure:1.11), Casey Reas developed Processing. Processing was initially a programming language and learning tool for teaching code. Focusing on visual output. As Processing became more popular it became a popular language for experimenting with computer generated visuals. Processing allows a programmer to experiment with producing computer generated visuals through an algorithmic approach. Processing is not used in this thesis, as it is designed for 2D visual output, and can not be utilized to produce realistic 3D interactive experiences.

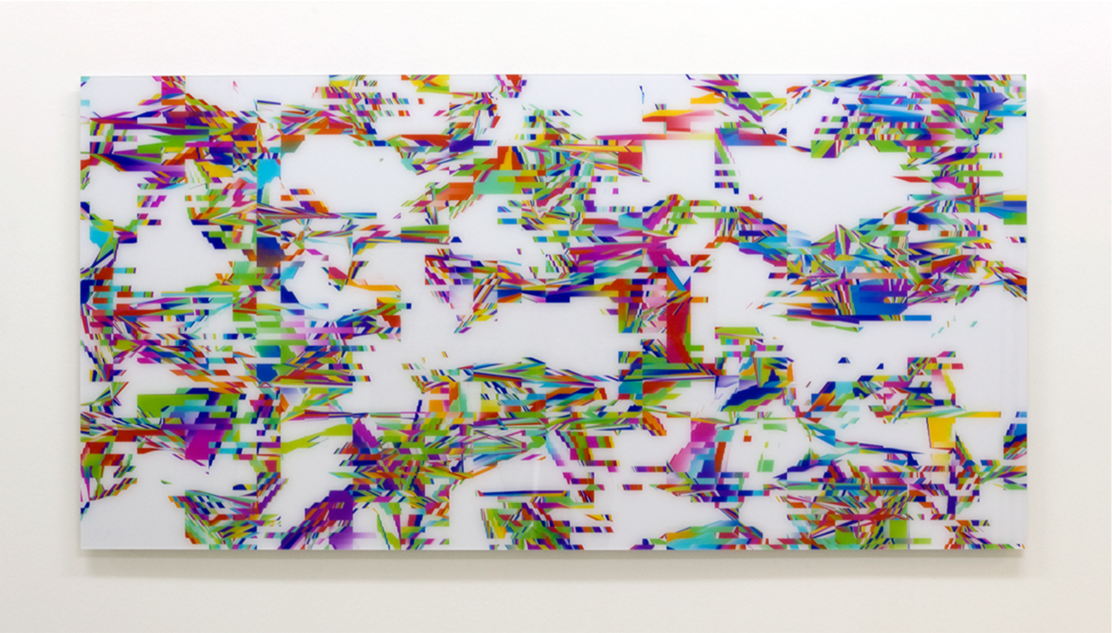


Figure 1.11: Example of artwork produced by artist and developer Casey Reas. Network D created in 2012.[30]

1.7 Node Based Parametric Modeling using Archimatix

In section 3.3.3 it is detailed how a parametric node graph can work as a solution towards producing artistic 3D scenes using parametrization. A third party library for Unity 5.0 (2.3.2) called Archimatix[31] already implements this method. Archimatix is maintained by Roaring Tide Productions, and was released, March 2017. It is a third party tool developed towards producing 3D mesh assets for use in Unity. Archimatix focuses on enabling an iterative design process through parametric modeling, and achieves this through integrating a visual node graph editor into its core workflow. A designer can start creating 3D meshes by dragging and dropping nodes from a node library, into the graph editor. Nodes are connected through wires, that function as input and output controllers between nodes. Archimatix is licensed for purchase on the Unity store. Archimatix was considered as a possible tool that could be useful for the thesis. In the end a custom solution inside Unreal Engine was chosen as the desired choice. Archimatix is a mature and powerful tool, but is built exclusively as a tool for Unity. In section 2.3.4 the reasons for choosing Unreal Engine, as opposed to Unity has been described in detail. A change over to Archimatix would entail moving over to Unity 5.0, as opposed to Unreal Engine 4.0. In addition to this, creating a custom solution inside Unreal Engine gives us full control over the development of the solution. The source code of Archimatix is hidden, and is not possible to extend upon. In the event that a custom editor solution for use inside VR were to be created, Archimatix might not be usable, as it is not designed with VR in mind. Making a VR application utilizing Archimatixes functionality would then not be possible, and is therefore a bad fit for our project. Lastly, it is not possible to animate Archimatix scenes after they are constructed. For our thesis, the introduction of a temporal dimension into the resulting scenes is one of our goals, hence being able to animate the scene during runtime is important.

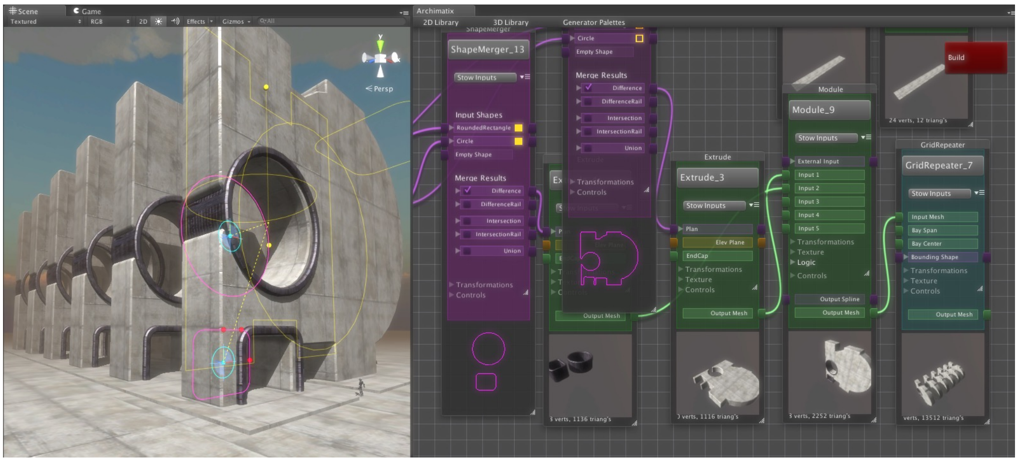


Figure 1.12: Example image of Archimatix in use. The node graph on the right describes the resulting output mesh on the left. All nodes and parameters are editable both in the editor and during runtime.[32]

1.8 Appearance Transfer

Successful attempts have been made to create algorithms powerful enough to analyze the visual style of an image, copying it over to target images that previously did not have that style. By doing this a third image is produced were the result of the appearance transfer from source to target is a new image that is recognizably similar to the old target, but with the added visual style of the source (see Figure 1.13). This effect is possible through a class of neural networks called Convolutional Neural Networks. According to the paper published on this subject, Convolutional Neural Networks "consist of layers of small computational units that process visual information hierarchically in a feed-forward manner".[33] These computational networks take the source and target images as input, outputting the resulting third image.

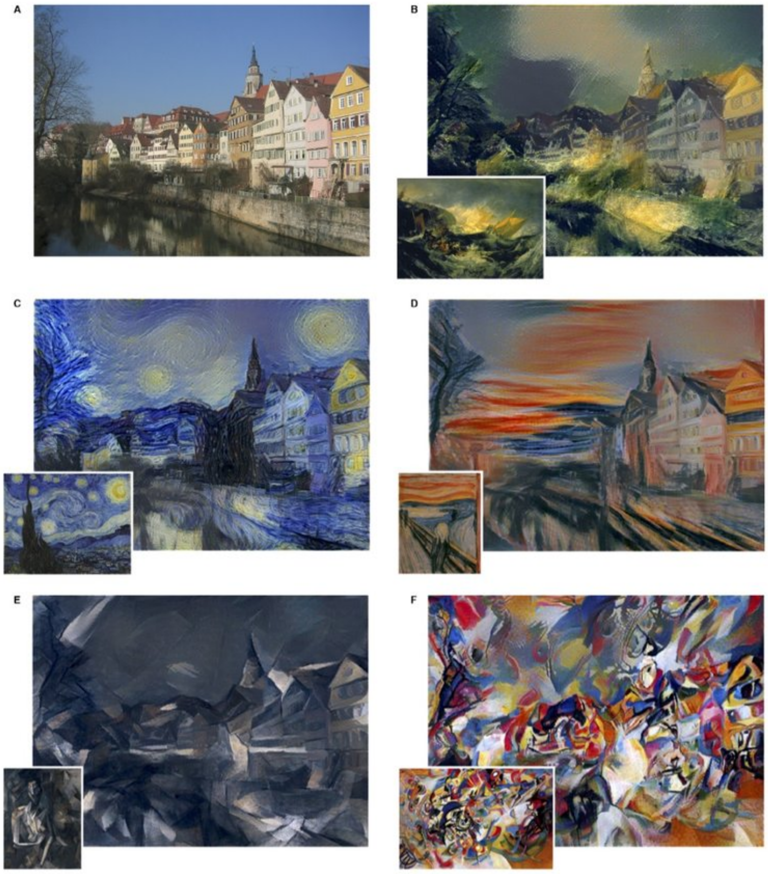


Figure 1.13: Results from applying different artistic styles from a source image, and applying them to a target image (image A). [33](Page 5.)

Chapter 2

Background

Central for being able to display 3D objects is how to store them inside computer memory, and how we use these representations to render modern graphical scenes. The work in this thesis relies heavily on the tools and environment provided by a game engine, therefore an explanation as to why these tools are used is provided. Virtual reality is at the center of our thesis, therefore the basics of virtual reality development and hardware components are explained.

2.1 Meshes

When looking at any objects in the real world, our eyes usually see them as solid objects with texture and color. If we were to look at an apple as an example, our eyes would see its round shape, and green color. As the apple moves closer and the object grows we start noticing more detail. An apple might look smooth from a distance, but if we close in on the object, we reveal complex detail such as bumps and texture. In theory an apple has near infinite detail, depending on the proximity of observation. If we were to make an attempt at representing this apple inside a finite array of numeric memory we would face a challenge. Infinite detail inside finite memory is virtually impossible, so instead we are forced to construct an approximation of the object. This approximation will represent the real world object at a certain level of detail. There are many ways to represent three dimensional objects, but we want a method that is memory efficient and quick to render. The triangle mesh method of representing objects is by far the most efficient and widely used method in modern day graphical computing. This comes down to the properties of triangles, and how these properties makes them easy for computers to render efficiently.

2.1.1 Triangles

A very useful property of triangles are their coplanarity. Coplanarity simply explained is the fact that a triangles three vertices always exist in the same plane. The instant you add a vertex, you do not have guaranteed coplanarity, which is unwanted when rendering computer graphics. This coplanarity is wanted because it defines an enclosed space inside a plane, in contrast to a non planar polygon, that can exist in multiple planes. This single plane enclosed space is used when attempting to render and sort the geometry.

Surface Definition

If we want to draw the surface of a triangle, we need to first define it's surface area. This can be achieved by using the barycentric coordinate system[34]. In this coordinate system, any point inside a triangles area can be defined by placing weights on the vertices of the triangle. The center of gravity of these weights then decide the position of the point. By changing the weights, the positions of the points inside the area are shifted. Shifting these weights along a certain range allows us to parametrically define the entire surface area of the triangle. This makes it possible to quickly rasterize a triangle using built in Graphics Processing Unit (GPU) support. (or to calculate ray-triangle intersections for the case of ray tracing)

2.1.2 Triangle Mesh Representation

The way meshes are defined in game engines is as a set of indexed positions that all connect together to form a number of triangles. These points are called vertices, and together form a set of triangles that form the mesh shape. (see Figure:2.1). Each of these triangles then, via their barycentric parameterization, define a surface representation. By defining enough triangles in the mesh, any shape in three dimensional space can be approximated. Most modern graphical applications require more than just positional and triangle data. Normals, and texture coordinates are required for mesh texturing and light calculations. This means a proper mesh used in games and media requires normal, and texture coordinates in addition to position.

2.2 Rendering a Graphical Scene

The basics for how a graphical scene is rendered has remained mostly unchanged for many years. This is most likely due to how effective the method initially was, and how well optimized it has become over the years. Except from physically based rendering e.g. ray tracing, which is outside of our scope, most 3D graphics rendering is done through a standardized rendering pipeline (see Figure:2.2). A computer, having the

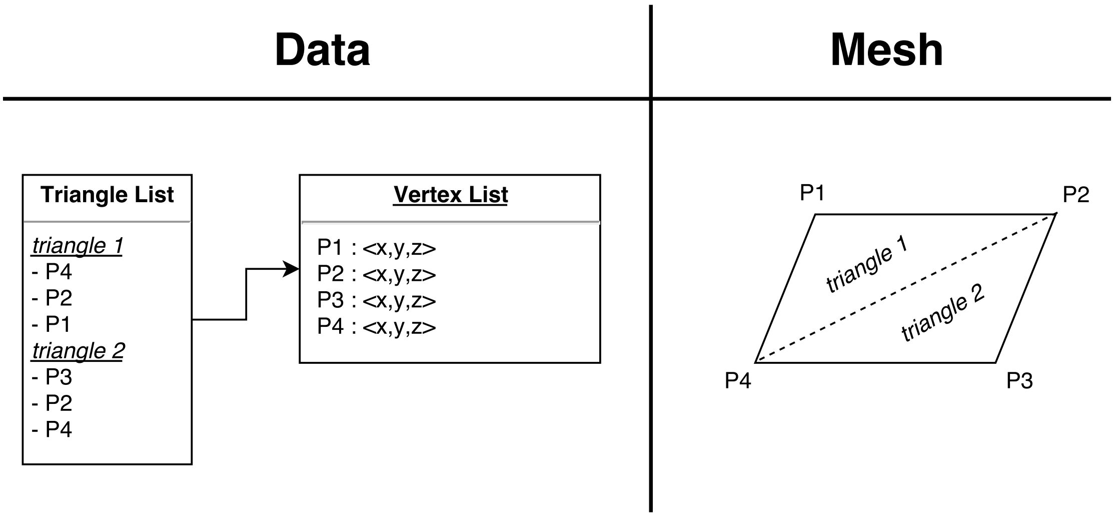


Figure 2.1: Representing a simple rectangle using points and a triangle list.

triangle mesh in memory, needs a way to convert the mesh representation into pixels on the screen. This job is performed by a computer GPU component, instead of the more general purpose computing unit, the Central Processing Unity (CPU). The GPU is designed first and foremost as a massively parallel computing unit targeted towards processing triangle mesh data and shader fragments. The data will go through several processing steps inside the GPU, each step utilizing the output of the previous step. This way of rendering meshes can be visualized as an 'assembly line' of processes, where each process is either programmable or implemented on a hardware level. The instructions of the programmable steps are provided through smaller programs called shaders, that are each compiled from source, on the CPU, and then executed inside the GPU cores. The individual steps are unaffected by each-other, allowing for the steps to be run in hundreds of parallel threads. This parallelization is key to the efficiency of modern graphical rendering.

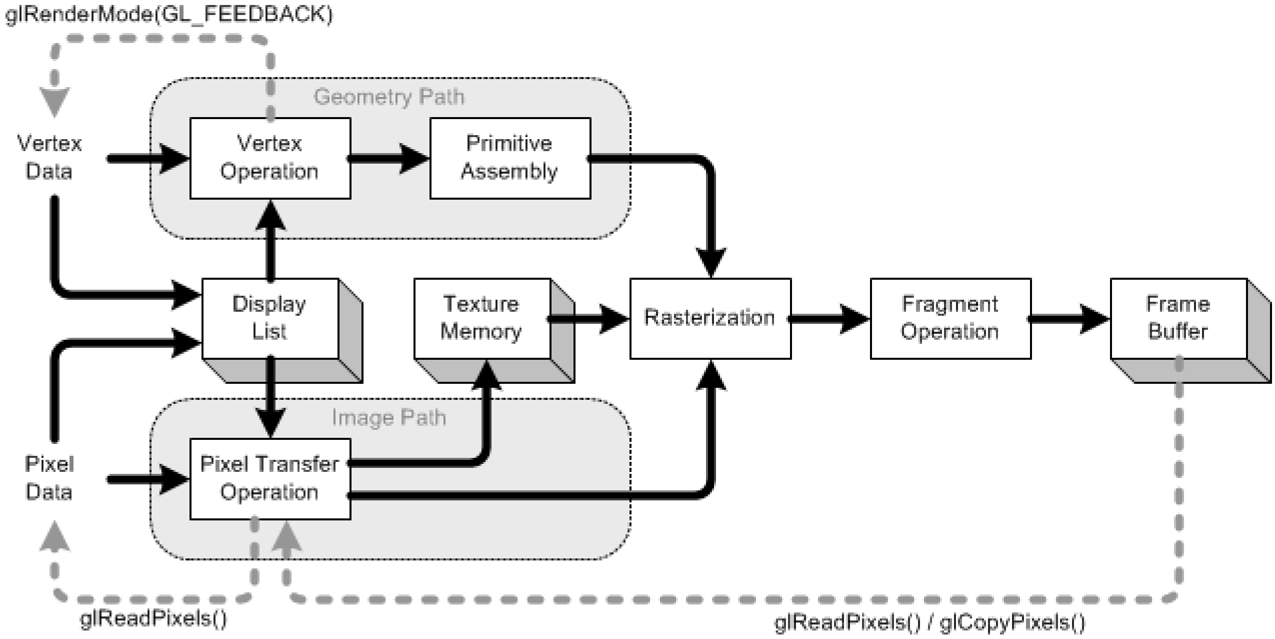


Figure 2.2: Diagram of a standard OpenGL rendering pipeline, containing both programmable and non-programmable steps.[35]

2.2.1 Geometry Instancing

One of the bigger bottlenecks developers meet when trying to render a large amount of meshes at once, is the number of draw calls involved in the process. When a mesh is to be rendered by the GPU, several operations are involved, such as loading the mesh data, sending data parameters to programmable shaders, and actually making a call to the GPU to start rendering our mesh. This is called a ”draw call”, and is sendt every time we want the computer to render a new object. Every draw call sent to the GPU consumes time on the CPU, if we want to draw many thousand instances of the same mesh on the screen, we would have to send several thousand draw calls, this would quickly form a bottleneck in our application, slowing down the entire program. Instead of sending a draw call for each mesh, we send the mesh data once, and supply only the data that changes between instances (e.g position, color, texture). This way several thousand copies of a single mesh can be rendered using only a single draw call. Geometry instancing is optimized towards drawing multiple instances of the same mesh, and as such is suitable when attempting to render a large amount of uniform objects in a scene. Examples of this could be an asteroid field in space, or foliage in a densely packed forest.

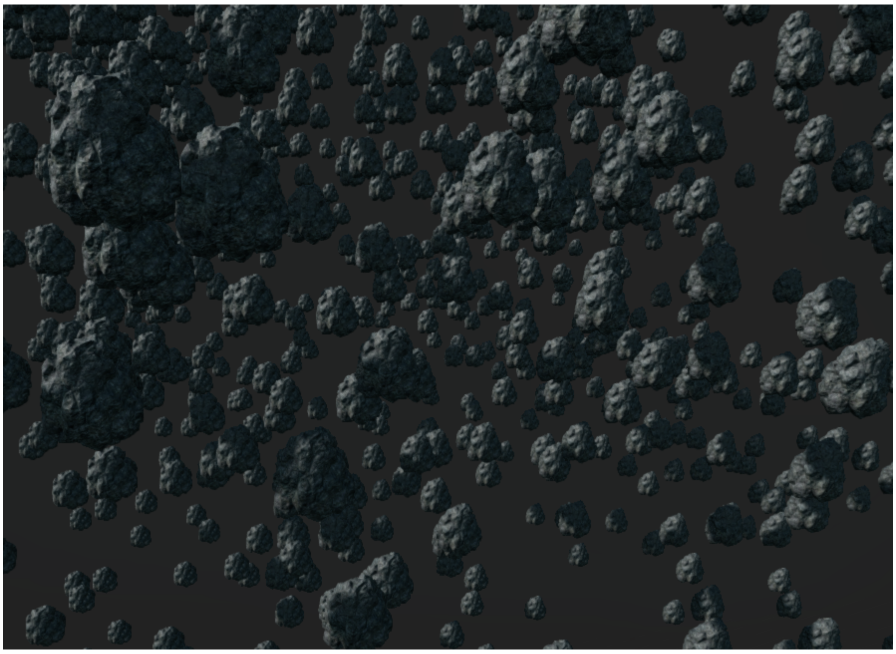


Figure 2.3: Example of geometry instancing used to render a large amount of objects. The performance impact of rendering this amount of object is drastically reduced by utilizing this method. The scene was created using the solution developed in this thesis.

2.2.2 Level of Detail

Some meshes, like meshes used in games or visualizations are very detailed, with a high triangle count. These meshes require more computational power for the GPU to render, but in return have a high level of fidelity. The time it takes to render a mesh, is not affected by the distance of the mesh from the camera. As a mesh is rendered further away, because of perspective projection, it also grows smaller on the screen. This effectively wastes computational power, as a detailed mesh is rendered far away, and detail is lost. To improve rendering efficiency, a mesh can have multiple levels of detail associated with it. When a mesh is far enough away from a camera, a different version with lower detail is rendered instead. With this method, less computational power is wasted on rendering meshes that are far away, while keeping the more detailed versions for times when the mesh is viewed closer to the camera. This technique of loading in different meshes of different detail, while still representing the same object, is called Level of Detail (LOD). LOD is used commonly in graphical applications where runtime performance is important, e.g video games. Figure: 2.4 shows an example of how a high polygon mesh can be represented through different levels of detail. The model itself is used in a resulting scene further into the thesis, where the model is instantiated many thousand times into the scene (see Figure:4.2). The LOD method is crucial for that scene to perform well. The model was created and exported from an open source

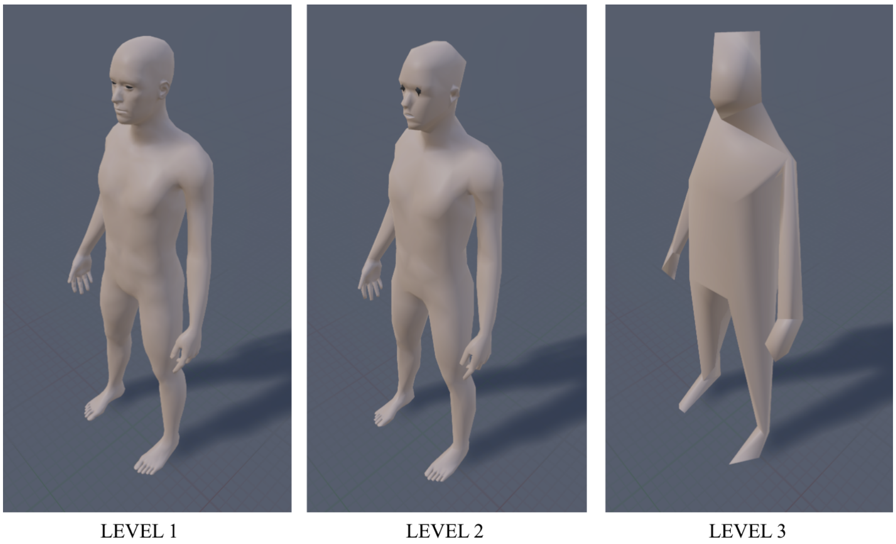


Figure 2.4: A mesh depicting a human, at different levels of detail.

software program, as described in section 3.5.1.

2.3 Game Engines

The modern game industry has now become a billion dollar industry [36]. Large companies are spending millions of dollars to release the next big industry title. In addition to these large companies there have been an uprising of smaller developer groups. These independent developers publish titles that sometimes rival the ones published by larger companies. Much of the reason some of these developers are able to develop these high quality titles, while having less manpower or funding, is through the power of game engines. When developing a commercial video game for the computer or standalone console, there are a multiple of elements that have to come together. Physics calculations, advanced graphics rendering, and runtime behavior, are all components needed to create a game. Developing all of this from scratch demands a lot of time and resources from even the largest company. It's this complexity in developing commercial games that have made the idea of a licensed game engine so appealing. A simple way to describe a game engine, is as an environment of tools and Application Programming Interfaces (APIs) that all are focused towards developing video games easier. Just like how a 3D modeling program is a suite of tools to create and edit 3D models, and a word processor helps you write papers, a game engine is fully designed to offload a lot of complexity from the developers side, over to the game engine's tools. It's however important to note, that a game engine is not designed to replace advanced 3D modeling programs, or music sequencing tools. A game engine will only make the process of integrating such produced content into the project easier. A large list of available game engines[37] exist for developers. When it comes to popularity, the top of this list is dominated by two engines, Unity[38], and Unreal Engine[39]. These two are frequently used engines in the modern game development scene, and each have their strengths and weaknesses.

2.3.1 Should a commercial game engine be used?

Game engines are large complex environments designed to create games within their own systems and tools. In certain scenarios, this fact can work against the developers. If a developer wants to create a feature or operation that is non-standard or very complex, the engine might not facilitate this as well as one might wish. This sacrifice of absolute flexibility and control is a trade-off one must pay to use a game engine. Some developers choose to create their own custom engines, usually to gain full control over the game's logic and behavior. These developers then build tools and features from scratch, usually starting with basic OpenGL or similar low-level APIs. These developers often spend significant time and resources developing these solutions, but in return has an engine that is custom made for their purposes that also costs nothing in licensing fees. Payment and licensing is another point some have against using commercial game engines. The payment and licensing type is different from engine to engine. Some engines require on only a one time payment, some are monthly subscriptions, and some even require percentile of revenue payments that can quickly scale to thousands of dollars. The commercial game engine Unreal Engine[1] was chosen as the engine for this project. Epic Games take a cut of the revenue earned through applications built using their software. This project is not monetized, and as such, Epic Games monetization model does not incur any cost.

2.3.2 Unity

Unity Technologies develops and maintains the Unity Engine. Unity has become a major part of modern independent game development. Unity focuses on making applications scalable and efficient. This focus has made the Unity engine popular among many mobile developers. Unity, like most modern game engines, allow for a multi platform release. This means that a developer can develop once, and then release towards multiple target platforms. Unity utilizes CSharp[40] and JavaScript[41] as a scripting languages. These scripts are used in Unity's modular entity system, enabling fast prototyping and development. Unity also allows aspiring developers to reuse pre-made modules and templates, included in Unity. This makes Unity an excellent platform for developers with less experience in software development.

2.3.3 Unreal Engine

The Unreal Engine is a well known engine that has been around since the 90s. The engine has been utilized by many successful games. For years Unreal Engine (UE) was recognized as the more graphically capable game engine. This reputation came from it's feature rich rendering system. The implementations of deferred shading, global illumination, translucency and post-processing were all superior to it's competitors. Recently with the release of Unity 5.0[42], many of these differences have been evened out.

The Blueprint System

The blueprint system is widely utilized within the UE development community for quick prototyping or general game behavior scripting. Epic Games presents the Blueprint system as a fully fledged visual scripting language. The tool is made for the purpose of defining Object Oriented (OO) classes or objects for use inside the engine. Blueprint is designed to be flexible and powerful enough that it can give game designers access to tools and concepts that usually is only available to programmers. Blueprint bases itself on connecting nodes using wires (see Figure:2.5). These nodes that are connected can represent a wide variety of features, anything from events, functions, in-game objects, or variables. This blueprint system can also exist alongside regular code written in C. A programmer can, using C, develop frameworks (e.g functions and objects) for others to then use inside blueprint. Blueprints are made inside Unreal Engine's own

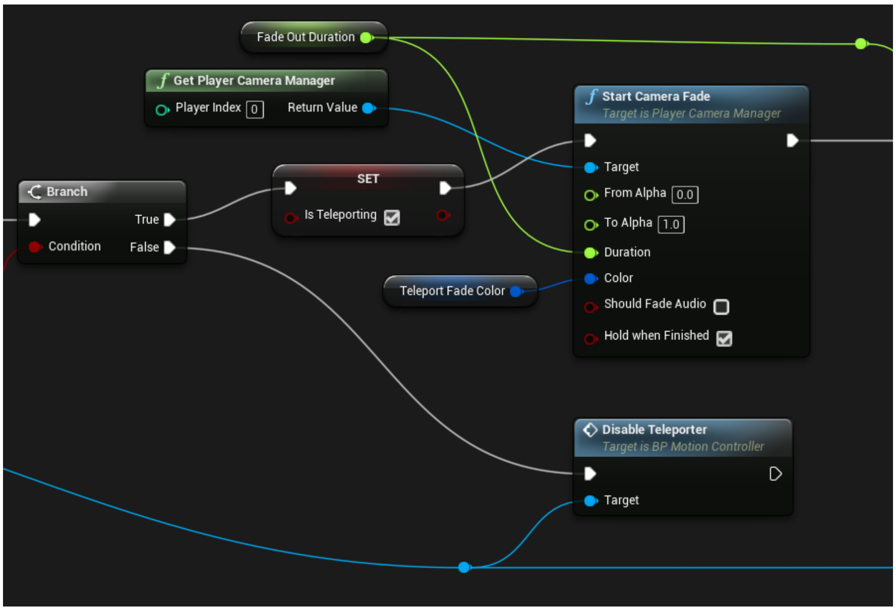


Figure 2.5: A small snippet of a blueprint graph showcasing how nodes are connected together with wires. Differently colored wires represent different variable types

blueprint editor. In the editor all nodes and wires are visualized as seen in figure 2.5. Differently colored wires represent different data structures being sent from node to node. Nodes are multi-purpose, in that they can represent different functionality, e.g conditional operations, functions, even other blueprint graphs can be condensed together into a node. Nodes have slots for attaching wires on either side, the left accepts input, the right connects to output. A Blueprint script will run at different times during the applications lifetime, either during application start, or as triggered responses from events, e.g a collision event or player death. The Blueprints execution flow is decided by the top wire that is always present for all function nodes. This gray colored wire decides which function in the node network is to be executed first, and which functions comes after. Nodes for branching execution, for-loops, and while-loops exist to help with program flow control. In addition to the Blueprint graph, each Blueprint can define any number of local or public variables for use in the graph.

2.3.4 Choosing Unreal Engine

Because both engines have a similar feature set in most areas, when choosing between the two game engines the small details is what makes the difference. Our project does not require very advanced and realistic light calculation. Re-creating artistic styles inside 3D is one of our goals in this thesis. Unity have no way of producing shaders outside of coding it manually, without paying money for third-party extensions. Unreal Engine comes packaged with their own visual node based shader building tool[43], that makes producing advanced shaders easier and faster than Unity. Unreal Engine has Blueprint (see section:2.3.3), this visual language makes prototyping ideas fast, while retaining the power of coding with C++ if desired. Unity only has C# scripts available for producing in-game code. It is possible that Blueprint in UE will also make it easier for other individuals with less technical knowledge to produce their own scenes. Weighing Unreal Engine and Unity against each-other, Unreal Engine 4.0 was chosen as the game engine for this thesis.

2.4 Virtual Reality

The 'virtual' in virtual reality comes from the 14th century, and entails something "being something in essence, though not actually".[44] This makes a virtual reality something that in essence is very close to reality, but not actually really there. If a digital display produces an image of something that looks to be there, but in reality is just a 2D image projected in such a way to convey 'realism', then that could be virtual reality. This idea of tricking a viewer into believing a constructed environment is actually 'real', has been around for decades. Modern technology, through digital displays and high-definition audio, has made constructed virtual realities more believable than before. Attempts at creating virtual realities using computers and displays started as early as the 70s (see section:1.6.2), In these early stages, the technology surrounding virtual reality was crude, and did a poor job of providing a believable immersive view to the user. As technology advanced the industry was mainly focused on training simulations such as military flight simulators. It is only now in modern times that technology has been able to produce results that are convincing enough, due to realistic renderings, high frame-rates and wide field of views. More than before, large companies within health and product design are picking up virtual reality as a possible tool that can supplement or enhance their products.

2.4.1 Virtual Reality Headsets

Providing the VR experience through a multi functional headset is a popular solution. Companies like Valve[45] and Oculus VR[46] have succeeded in distributing their respective HMD units to many countries in the world. These headset solutions vary in both size and available functionality, some are expensive units requiring a powerful dedicated computer, and some are cheaper, but rely on weaker mobile processing power. The tracking, and freedom of movement also changes significantly from unit to unit. The relatively inexpensive hardware requirements and higher portability has made VR headsets more commercially widespread than the older CAVE systems.

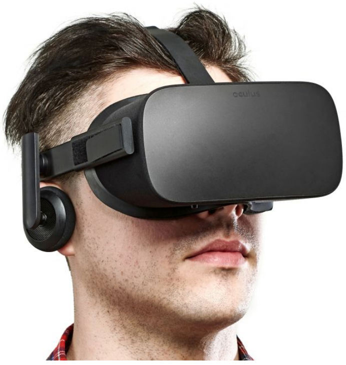


Figure 2.6: The Oculus Rift head-mounted display in use.[47]

High End

A high end configuration is popular due to the better quality VR experience such a unit is supposed to deliver. Such units usually provide two high resolution monitors, each monitor set behind a Fresnel lens. The headset is mounted on the front of the head, usually with straps (see Figure:2.6). Audio is provided through either headphones or earplugs that are plugged into the HMD. When the head position (3 values) and orientation (3 values) are tracked, this is called 6 Degrees Of Freedom (6-DOF) tracking and is used to update the rendering according to the viewpoint. Some tracking systems only track the viewing angles. The most significant difference between configurations, is their tracking capabilities. Arrays of usually two to four Infrared Radiation (IR) trackers, track the positions of multiple light emitting diodes (see Figure:2.7), mounted on the headset. These trackers provides ultra-low latency on responses to movement. In addition, they allow for full 6-DOF within the tracking area. Extra controllers designed for hand tracking (see Figure:2.8) are also usually included. IR tracked hand controllers with finger triggers and buttons allow for better interaction with the virtual world. It is also possible to buy leg and waist trackers, and integrate them into their setup.

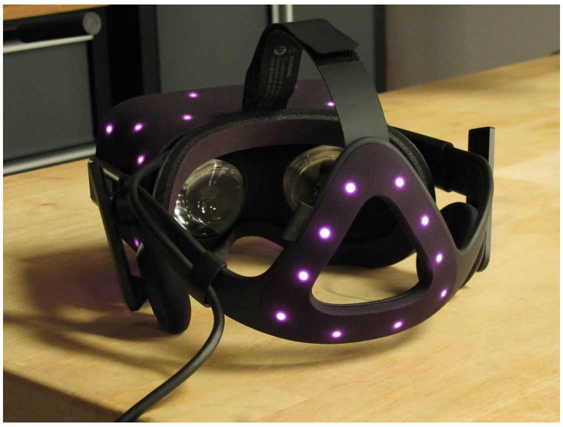


Figure 2.7: The Oculus Rift head-mounted display, with light emitting diodes showing, due to infrared sensitive camera imaging.[48]

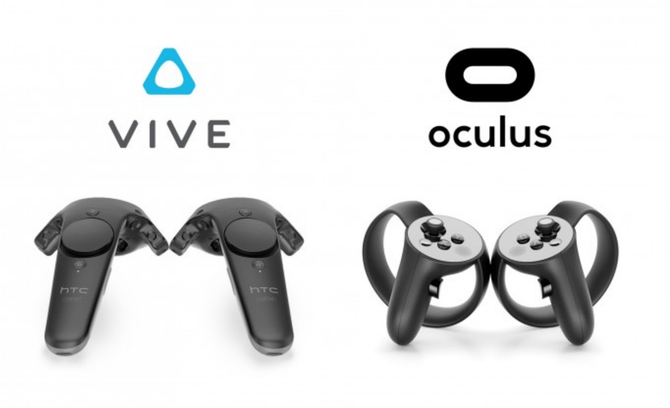


Figure 2.8: Positionally tracked hand controllers for the Oculus Rift(right), and HTC Vive (left).[49]

Mobile and Standalone

A high end HMD unit is costly, many VR developers focus on providing an affordable VR experience to as many people as possible. For this reason, mobile VR was developed, since many consumers on the market already own a high end smart-phone. These smartphones are usually also capable of playing lighter games with both 2D and 3D graphics. There is then nothing preventing us from running optimized VR applications on mobile units, as long as the developers make sure the application performs well enough on the restricted hardware. Mobile VR devices are usually non-expensive products, that are designed to utilize the users phone to display visuals, sound, and perform view tracking. In most cases the phone is fitted into the unit in a front compartment. The stereoscopic view is achieved by splitting the phone monitor into two parts. Each part is then viewed through a lens, much like in the high-end versions of VR. Assuming the consumer already has a high-end mobile unit, these VR configurations can be quite inexpensive. A great example is the Google cardboard.[50] Made out of normal cardboard (see Figure:2.9), the unit provides two lenses, and a space for the mobile unit to rest while in use. Tracking and movement in mobile VR is usually done through the sensor package provided by the phone. Magnetic sensors and accelerometers in the mobile phone take care of tracking viewing angles. This dependence of phone sensors means that most mobile VR applications only allow for 360 degree rotational movement, and does not support positional tracking.

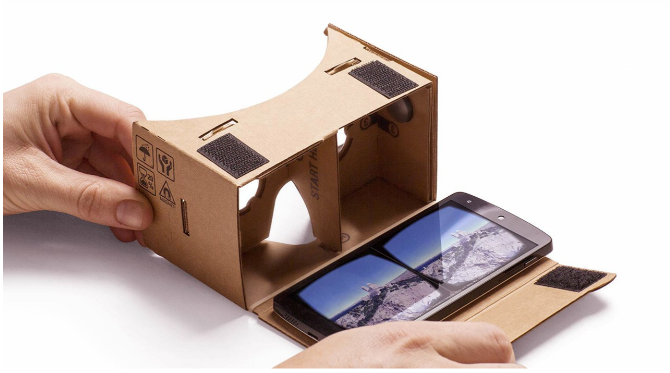


Figure 2.9: Showing an assembled Google cardboard unit. The Google cardboard is designed for the phone to be fitted into the front compartment.[51]

2.4.2 Virtual Reality Challenges

The major challenge with VR is presenting a believable world to the user, in a way that does not incur any discomfort or nausea. The brain is very easily confused by mismatching sensory input. If the users body is physically at a standstill, but the scene is moving, the brain receives contradictory sensory input which can result in nausea.

This is the same effect that causes car sickness, more commonly referred to as motion sickness, or kinetosis[52]. This motion induced sickness is a large challenge in VR. To reduce the discomfort using a virtual reality headset, the system needs to fulfill a set of minimum requirements. These were first set by Oculus at the release of their prototype HMD. The headset is now considered a guideline for how to provide a passable VR experience. These hardware specifications are listed as such.

- 2160x1200 pixels with 90Hz display refresh rate
- Horizontal field of view above 90 degrees
- A one millisecond maximum display latency

The specifications listed above come from an official Oculus blog[53], and a teardown of the HMD specifications[54]. As stated on the oculus website, a Oculus Rift Consumer Version 1 (CV1) headset will require roughly three times the power of more conventional 1080p rendering. The higher performance requirements needed to run VR is a challenge for developers new to VR. Refresh rate and motion-to-photon latency are important when attempting to develop a well performing VR application. These two concepts represent the visual update frequency of the images displayed to the user, and how quickly a users real world movement is represented visually inside the application.

Refresh Rate The refresh rate of a monitor dictates how many new images it can show every second. This becomes very important for virtual reality, as the human eye is very close to the monitor. Since the monitor is so close, even minor faults in the image rate will appear harsh to the user. A virtual reality user that uses an application that has a very varying frame rate, or too low frame rate all together, will most likely experience discomfort.

Motion-to-Photon Latency The motion-to-photon latency reefers to the time it takes for the computer to recognize that the user has moved. That means the time it takes from when the user makes a motion in reality, to when the application has finished recognizing the movement, and represents it with a new frame[55].

2.5 Parametric Design

In this thesis, works of art are described through parametrization, both in geometry and in visual style. The motivation behind this is the compact descriptions of artworks that can be made by parameterizing them. Instead of storing the color-value of every pixel (assuming the art is digitized), the art could be described through parametric values, defining lines, shapes, and color for the entire work. From a parametric description. A re-production of the art work could then be created through a process, reading the parametric description. This form of describing an artistic image or scene with parameters works better when the work has a higher degree of regularity. Regularity comes in the form of repeated elements that are present in the artwork. Repeating patterns, like a checkerboard pattern, are good examples of something with high regularity. A work of art with high regularity is easier to parametrize, as the regular part is described once, and then repeated when needed. If the artwork has low regularity, with more complex details that don't repeat often, then parametrization becomes more difficult. The reasons behind choosing some of Pushwagners works described in section 1.5, are partly due to their high degree of regularity. Instead of relying on a existing work of art, we can also produce our own art. If we create these works of art from scratch, we can introduce regular elements that are more suited for parametrization.

2.5.1 Parametric Equation

The simple explanation of a parametric equation, is that of a function of one or more independent variables. These variables are called parameters. Parametric equations are mostly used to express geometric surfaces and curves. Parametrization is the representation of such an object or surface described by one or multiple functions. An example of a simple parametric function is a sinusoidal function (see Figure:2.10) with some input parameters:

$$f(x, a, b) = a * sin(x) + x/b$$

Depending on the parameters a and b, the curve changes. The interval of x is also part of defining the curve. In this example, the interval is [-5,5]

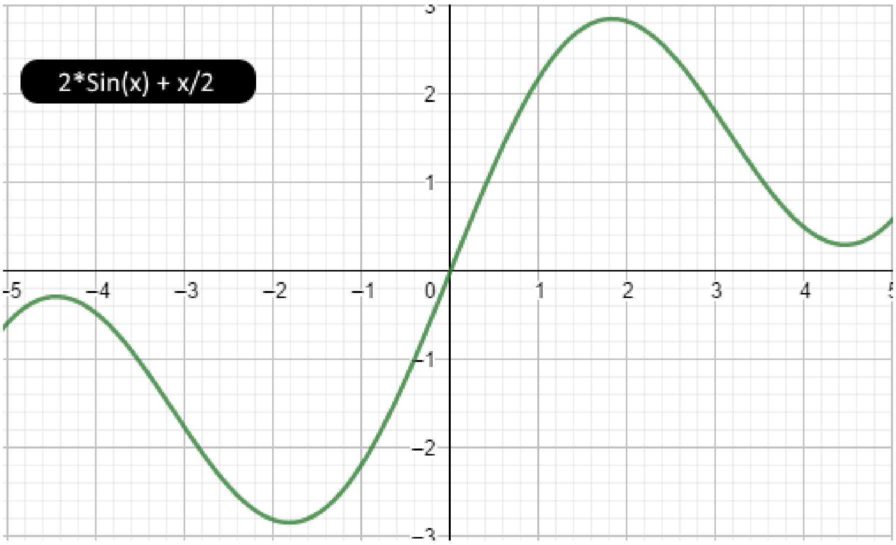


Figure 2.10: A sinus curve plotted from a parametric sinus function. Here the function is on the form f(x,a,b)=a*sin(x)+ x/b where a = b = 2.

Chapter 3

Solution

To attempt to answer the research questions, a working solution is needed. The solution should generate scenes in a way that is relatively quick and not too cumbersome to work with. The intent is to see if a scene can be produced, that has a likeness to the original art, and then create an original scene without reference material. The method should ideally have a parametric aspect to it, this way if the parametric variables are changed, that change will be reflected in real-time. Re-reacting a work of art for 3D is split into two separate parts for this thesis: The shape of the scene(geometries), and the visual style(shaders).

3.1 Project Setup

This thesis focuses specifically on the development of VR applications inside the UE framework. (UE) already comes pre-packaged with a project template for VR development. The template will customize the settings for the development environment towards VR, to enhance stability and performance. To realize the idea of generating artistic scenes through visual scripting, we started by testing the capabilities of Unreal Engine. There is a minimum of functionality that needs to exist in Unreal Engine for us to be able to succeed in our implementation:

1. Implementation of custom runtime-editable mesh structures.
2. Support for low-level shader customization.
3. Enough flexibility in the engine to implement our descriptive meta-language.

The reasons behind these demands are connected to the goal of the thesis. Run-time editable meshes allow for animation and movement, while also allowing the application to generate the scene from a description (meta-language). Modern shaders program the graphical rendering pipeline of meshes to the screen (see Section:2.2). A way to program these shaders in a quick and detailed manner is needed if the visual styles are to be replicated inside a 3D scene. Finally, a way to implement the meta-language is required, as to make it possible to describe the scene both in shape and visual style.

3.1.1 Program Flow

Some quick context is given as to what states the UE program is in, to avoid any confusion. There are two main stages of our application: the editor stage, and the runtime stage. The editor stage is where all code production, scene description, and scene editing is done. In this stage the UE application can be simulated inside the editor. The code and blueprints (see Section:2.3.3) are editable, and also available for simulation individually. When a simulation is started the state of the application transfers over to run-time, where all the code and Blueprint functionality is run continuously. In the run-time stage, it is not possible to edit any code, or descriptions of any scenes that have been made, however procedural meshes can still be edited and animated. When creating a new 3D scene, the scene itself is designed and edited during the editing stage, but can only be experienced inside VR during run-time. The scene descriptions can not be edited while the user is inside VR. It is possible to achieve this capability, but a different approach would have to be implemented (see Section:7.2).

3.2 Procedural Mesh Support

Most meshes and model data in UE are represented using the static mesh component. A static mesh component stores a models polygon data. Since the data is static, the model can be cached in GPU memory, allowing for more efficient rendering. This thesis requires the production of custom meshes from parameters at runtime. The static mesh component does not support run-time mesh modification. This makes it inadequate for this project, since we will produce meshes and edit them when the application starts, and while it is running. Unreal Engine 4 includes an alternative to the static mesh, called procedural mesh. By using a procedural mesh, all aspects of the mesh can be manually defined, even during run-time. The procedural mesh is empty at first. Specific access functions allows us to extract and add sections of mesh during runtime. In addition to building meshes while the application is running, the polygon vertices can also be animated, this will become useful later.

3.2.1 Runtime Mesh Component

The procedural mesh component provided by Unreal Engine fits well with our requirements (generate and edit meshes at runtime), but still poses a problem in that the component is still under development, and as such lacks efficiency. If large meshes with several hundred thousand vertices are to be edited, the procedural component might not be able to achieve this at an acceptable rate of 60 to 90 frames per second. This performance weakness is addressed by a third-party component designed to replace the procedural mesh component. This component is named The Runtime Mesh Component (RMC)[56] and is created by a third party component developer called Conway[57]. The RMC claims to have higher performance than the original procedural mesh component, while keeping full compatibility with existing code. This component is downloaded and compiled together with the rest of the project.

3.3 Building Scenes

Since the project now has support for custom runtime editable meshes, a way to create scenes for the application needs to be found. The method that the scene is built through needs a certain level of flexibility and customization for us to be able to create scenes that are complex enough. Some options have been tested in how geometry is defined, and then displayed inside the engine.

3.3.1 Using External Modeling Tools

A way to construct a scene is to first utilize modeling software to create the desired geometry. In the modeling software, meshes can be created and edited easily, because of the access to complex and powerful meshing tools and mesh-editing features. An export-import pipeline could then be formed, from the model tool, to unreal engine. This is easily done, as most modern modeling tools support exporting to Filmbox or Wavefront .obj formats. These formats are supported in UE and importing them is straight forward. After importing, the scene is assembled using these custom models. If the scene requires complex shapes that are difficult to produce through parametric or procedural means, then this would be a good solution. One problem with this approach is the rigidity of the creative pipeline. If we want to change a part of the scene, we would have to go all the way back to the modeling program. The asset would have to be re-modeled, exported, and then imported again. This strides against our original intentions of creating art-like graphical scenes through parametrization. It also makes it impossible to have dynamically changing scenes during runtime. Much of the work to produce a scene is done in the modeling program. By using this method the idea of utilize a visual language to compose the scenes is undermined.

3.3.2 Scene as a Function of Input

The second way that the thesis goal could be achieved, is more dynamic and easy to edit than the previous one. Instead of importing scene objects as assets, the scene is defined procedurally. We attempt to carefully construct the graphical scene as a function of several inputs. Altering the input parameters will alter the scene accordingly. By doing it this way a higher level of flexibility is gained, since everything is constructed at runtime. If we want to change something in the scene, it is fast and easy to change of the parameters. With this method every scene is generated by a function call, where the only possible change to the scene after implementing it, is through changing the parameters. The negative side of this solution lies in the fact that every new scene has to be implemented from scratch. This promotes very little code re-use, and re-implementing code for each scene can be very time consuming. In addition, if a developer wanted to change a part of a scene that is not covered by the parameters (defined in the function), the entire function would have to be changed in the code. This somewhat defeats the purpose of implementing graphical scenes through parametric input, as this solution is still cumbersome and time consuming. To help this fact one could create a library or set of functions to re-use cross scenes.

3.3.3 Parametric Function Nodes

The third solution and final solution is based on implementing a library of individual nodes, in the Unreal visual programming environment. This creates an expressive language for producing "parametric" scenes. And makes it possible for non-programmers to make scenes too. Each node represents an action or statement made of the scene. For example, to create a circular path of red cubes, only a few nodes would be needed. (see Figure:3.1). Each of the nodes are implemented by short C++ functions that take one

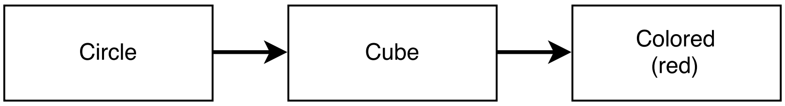


Figure 3.1: An example of a very simple scenes described by a node flowchart.

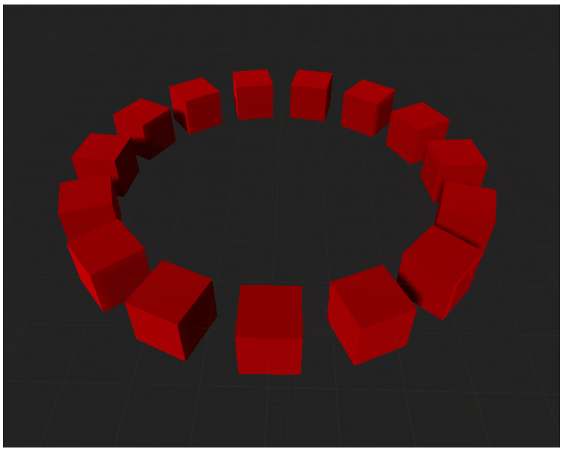


Figure 3.2: An example of the circle of cubes that would be produced by the configuration depicted in Figure 3.1

or several parametric inputs. In addition to parameters some nodes takes mesh data as input and outputs modified data. This way we can chain together nodes and their effects. An advantage of this solution is the re-usability of the code. Very different scenes can be produced simply by altering the composition of nodes, or changing the input parameters such as the circle-paths radius or positioning. Adding and removing nodes and changing their parameters is done in the editor. The editor detects any changes, and automatically re-compiles the scene descriptions, built through blueprint. This way, changes can almost instantly be reflected in the in-editor scene view. The function library of nodes can easily be expanded after need. The more functionality is added to the library, the more powerful it becomes.

3.4 Using Unreal Blueprints

Unreal Engines Blueprint system is important for our work. Blueprint allows us to represent our function library as nodes. Unreal Engine uses a dynamic component based actor system to represents the in-game objects and their behavior. Every actor derives said features from a template. These templates, called blueprints, serve as a description of a possible in-game entity. The blueprint can be used to spawn this entity as many times as required. A very important part of every blueprint, are the graphs that define runtime behavior. These graphs are composed of a set of connected nodes. The nodes and how the are connected, decide the blueprint behavior. This visual scripting language will be used to execute our function library.

3.4.1 Creating our Blueprint

The first step is to create the blueprint itself. Unreal Engine Blueprints and how they work are explained in section2.3.3. An empty blueprint starts as a blank canvas. After creating the class, the developer is free to add pre-made components, like static mesh components, or collision components. In the event that third-party components have been imported and compiled, these can be included as well. In our case two specific components are required, The RMC, and the Hierarchical Instanced Static Mesh Component (HISMC). Without these components we will not be able to display any meshes, or edit any of our mesh data. The HISMC is specifically needed to apply instancing for heavily repeated meshes. With the components ready and set up, we can create our code for generating scenes.

3.4.2 Creating Graph Nodes

As mentioned before in this thesis, the blueprint graphs are used as a way to describe the scenes. To do this we need to expand the pre-existing set of nodes with a custom node-set. Creating new functionality for an in-game object is as easy as creating a C++ class, and have it inherit from unreal engines *AActor* class (see Listing:3.1). After creating the class UE allows us to expose the C++ functions using annotations.

```cpp
UCLASS()
class PARAMETRICART18_API AComposer_Scene : public AActor
{
   GENERATED_BODY()
   public:
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   URuntimeMeshComponent* RMC;

   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   UHierarchicalInstancedStaticMeshComponent* HISMC;

   UFUNCTION(BlueprintCallable, Category = "Composer-Mesh")
   void Colored(FMeshData mesh, FColor color, FMeshData& outMeshData);

   UFUNCTION(BlueprintCallable, Category = "Composer-Mesh")
   void Snap(FMeshData target, FMeshData mesh, EFacingEnum facing,
    FMeshData& outMesh);
}
```

Listing 3.1: Example of a basic class for use within the Unreal Engine. Written in C++ it also makes use of UE specific annotations such as UCLASS and GENERATED_BODY, to mark the class and its variables with metadata. The metadata is used by the engine during compilation. It contains a sample of our function library.

By letting our blueprint inherit from the C++ class, it will automatically gain access to all functionality that has been annotated for use within the Blueprint editor. This means every *UFUNCTION* will show up as callable nodes inside the blueprint graphs. An example: the function definition *Colored*(see Listing:3.2) will be available as a node when compiled with the UE annotations. (see Figure:3.3)

```
UFUNCTION(BlueprintCallable, Category = "Composer-Mesh")
void Colored(FMeshData mesh, FColor color, FMeshData& outMeshData);
```

Listing 3.2: The colored function, with the UFUNCTION annotation

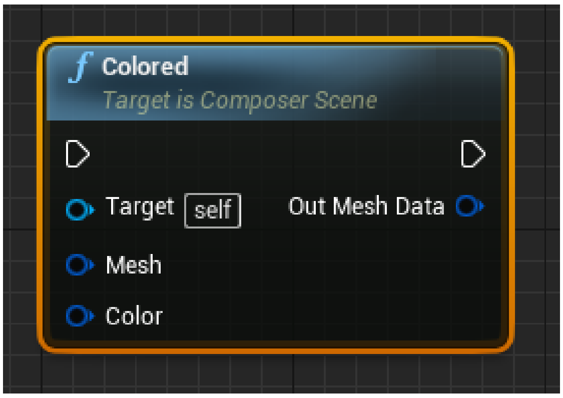


Figure 3.3: The *UFUNCTION* from lst:3.2, exposed inside the blueprint editor

3.4.3 Generating Meshes in Code

The RMC allows us complete control over the mesh data at runtime. This can be considered both positive and negative in our case. The cost of full control, is that some support functionality is lost. We get very little help for actually creating the shapes and geometries we want to display. Any data that should be put into the RMC needs to be created manually. Unreal Engine 4.0 does not provide any method for generating geometries, any of this functionality must be produced from scratch in code for the solution to work. Producing all of the code manually would take more time than is affordable, so two extra libraries are added into the project.

> **GaussianLib**: A basic linear algebra library for C++. GaussianLib is required as a dependency of GeometronLib. This library is also used when performing linear algebra operations on meshes produced in our code.
>
> **GeometronLib**: Library for generating basic geometric shapes, with some added functionality for collision checking and mesh processing.

These two libraries are open-source and written in C++. The code source files are added to the project directory, and are compiled together with the rest of the code.

Their functionality is accessed by importing any needed header-files. Blueprint nodes that generate geometric shapes for the scenes (see Listing:3.3) can now be created in the code.

```
UFUNCTION(BlueprintCallable, Category = "Composer-Mesh")
void AComposer_Scene::Cuboid(const FVector diagonal, const FVector
    segments, bool winding, FMeshData& outMeshData)
{
  Gm::MeshGenerator::CuboidDescriptor description;
  description.alternateGrid = false;
  description.size = { diagonal.X, diagonal.Y, diagonal.Z };
  description.segments =
    {
      (uint32)segments.X,
      (uint32)segments.Y,
      (uint32)segments.Z
    };

  Gm::TriangleMesh mesh =
      Gm::MeshGenerator::GenerateCuboid(description);

  if (winding) FlipGmMeshWinding(mesh);
  outMeshData.GmMesh = mesh;
  CopyGmTriangleMeshToBuffers(outMeshData);
}
```

Listing 3.3: Code for generating a cuboid using GeometronLib.

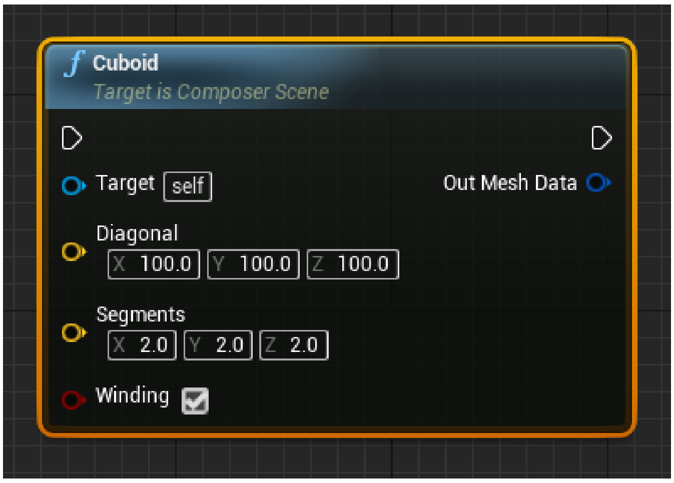
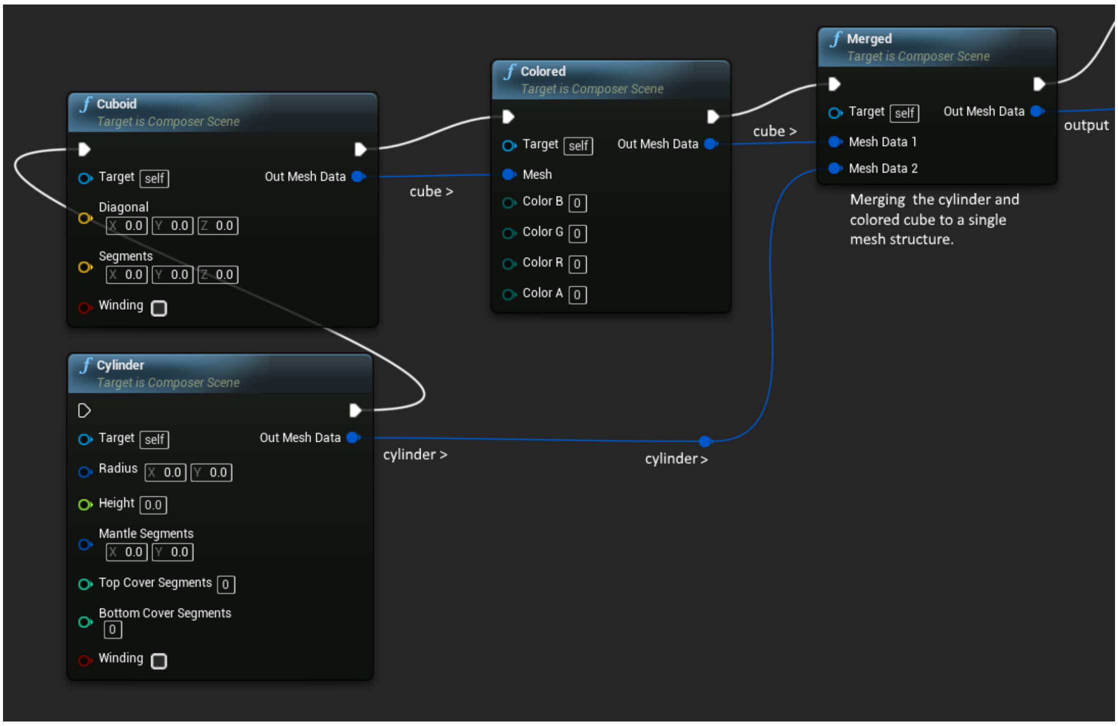


Figure 3.4: The cuboid mesh generator from the code above, as a node.

The FMeshData Structure

Every function in the library that work on, or generates meshes, has the *FMeshData* structure either as input or output. *FMeshData* is exposed to blueprint, which means it can be passed around as a variable from one node to another. Without this structure it would become difficult to keep track of the mesh data. *FMeshData* works as a wrapper around the *Gm::TriangleMesh* structure that is included from GeometronLib. The reason the preexisting structure is wrapped inside an overlaying structure, is to gain compatibility with the RMC, as the component can not process and render the GeometronLib mesh structure directly. Instead, the data is stored in the *FMeshData*, and then converted over to a RMC compatible format at the end of the scene creation process. This FMeshData structure is available inside the Blueprint editor, and can be used to transfer mesh data between function nodes, as seen in Figure:3.5.

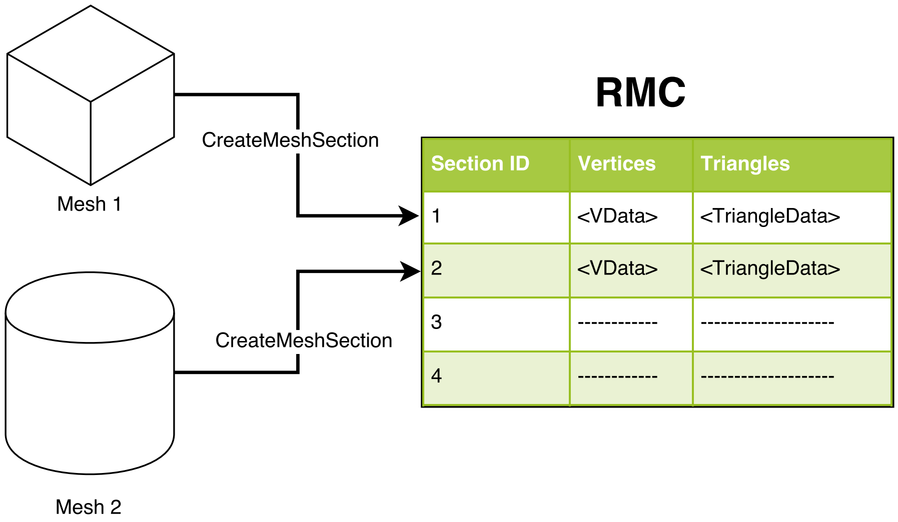


Merging the cylinder and colored cube to a single mesh structure.

Figure 3.5: Example graph of the flow mesh data between nodes. The mesh data is represented by the blue wires connected to the nodes through input and output slots. The example graph produces a colored cube, and an un-colored cylinder, then merges them together into a single mesh.

3.4.4 Using the Runtime Mesh Component

It is not enough to just define and implement functions, we also have to make sure the RMC gets access to the scene data so it can be rendered in the scene. The RMC treats a single mesh as different sections. When loading in new mesh data, the section ID needs to be specified, and the data is either inserted or overwritten in the slot according to the ID (see Figure:3.6). Each section can be configured independently from the others, as if it was a separate mesh.



Section ID	Vertices	Triangles
1	<VData>	<TriangleData>
2	<VData>	<TriangleData>
3	------------	--------------------
4	------------	--------------------

Figure 3.6: Meshes are sectioned off into specified section slots. Sections are referenced through a numeric ID, and can be individually retrieved or overwritten.

3.5 Creating a Scene

How one of the resulting scenes were built is explained, as to give an example of how our function library is used in practice. The reference for this scene is Pushwagners Selvportrett (see Figure: 1.1a). How a scene is constructed can be described with 5 steps:

1. Find parameters that define the scene.
2. Identify the geometries that could be used to represent our scene.
3. Construct description of the scene using meta-language (function library).
4. Apply desired visual style.

There is more to creating a scene than just these five steps, but they provide a general overview of how a scene, with or without reference material, is constructed. The first

Table 3.1: My caption

Variable	Detail
Radius	The total radius of the entire scene geometry
Floor Height	Height of the floor that is repeated on every level of our scene
Pillar Width	Width of the pillars that are repeated.
Floors	How many floors the structure will have in total
Sections	How many circle sections the scene will have

step is to decide the parameters for the scene (see Figure:3.7). The parameters that are identified could be any part of the scene found to be suitable for parametrization. Suitable parameters are often scalar, such as length, height, "amount of", or frequency (if the scene has a random element). The number of parameters identified could be any, but for this scene five specific parameters are identified. The parameters from figure 3.7 are described in table 3.1. After the parameters are identified, a clean blueprint is created. All of the scenes start out as clean Blueprints. Newly created Blueprints have no pre-existing functions, graphs, or variables defined, and only have access to a default set of nodes from the engine. The blueprint is then set to inherit functions and variables defined in our C++ class. When a blueprint inherits from a C++ class it gains access to all functions and variables that have been annotated (see Section:3.4.2). The design of a scene starts with adding parameters in the form of variables inside the blueprint (see

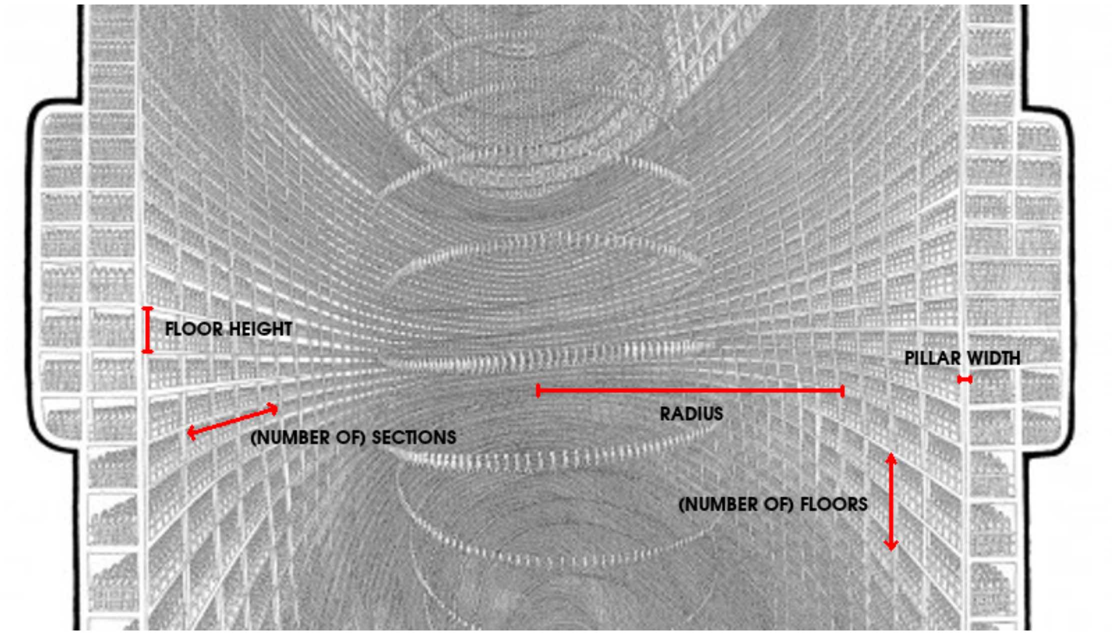


Figure 3.7: Selvportrett with an overlay of parameters that can be utilized to define the scene desired 3D scene.

Figure:3.8). Variables can be made public or private by the creator. A public variable can be changed from outside the editor when the Blueprint is instantiated inside a scene. A private variable is locked, and hidden from outside of the editor. In this scene private variables only store calculations that are done while generating the scene, and are only there to reduce visual clutter inside the blueprint graph. The next step is to identify the geometry that can best fit as ”building blocks” for the scene. If a viewer ignores the many humanoid shapes in the artwork, the viewer will see that the artwork consists mostly of straight angular shapes and large surfaces. This makes rectangular cubes a good fit for attempting to approximate the scenes shape. In addition, the curved nature of the scene can be approximated through curved pipes. The third step is to design the blueprint graph that will describe our desired scene. The Blueprint graph in it’s entirety is shown in appendix C. The graph is sectioned into different areas, that each construct specific parts of our finished scene, step by step. After the graph is finished the resulting scene is visible inside the editor. The parameters set from before can be changed, and the scene is rebuilt accordingly.

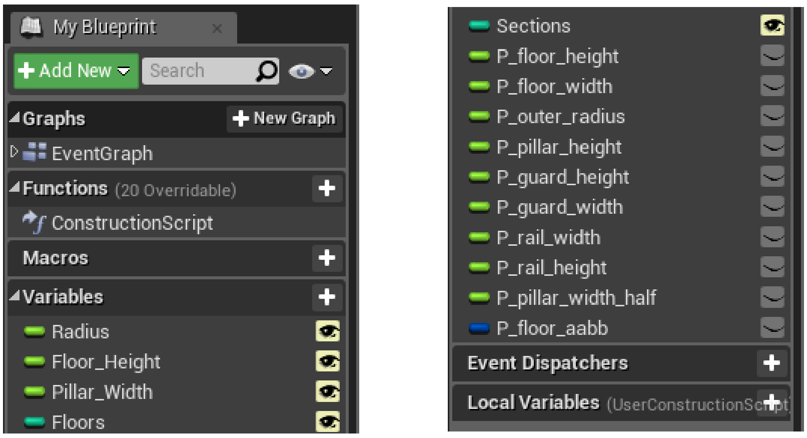


Figure 3.8: Variables for our scene. Some variables are public, and allow us to change parts of the scene during runtime. Variables with P preceding the name are private variables that help us store calculations while generating the scene.

3.5.1 Populating the Scene

Selvportrett depicts what looks like several thousand people populating the scene. To approximate this inside the 3D scene, a human model was first created, and then copied into the scene through geometry instancing. The model itself was created inside an open source software, called MakeHuman[58]. MakeHuman consists of an editor where a human model can be generated from different parameters (see Figure:3.10). An example of some parameters that affect the final outcome are age, height, gender, muscularity, etc. By changing these parameters the model is rebuilt according to the changes.

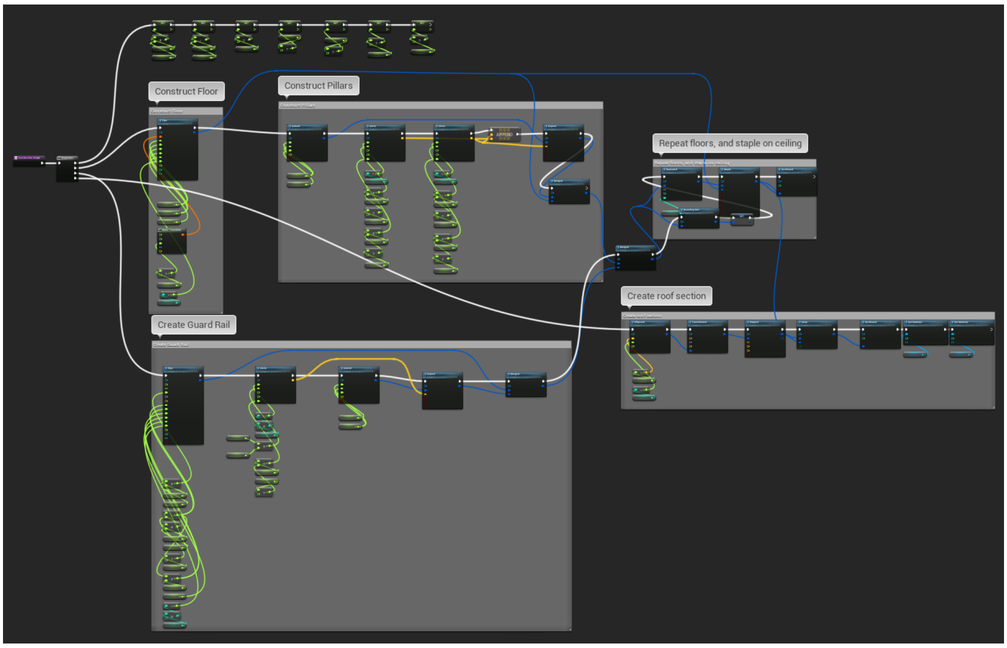
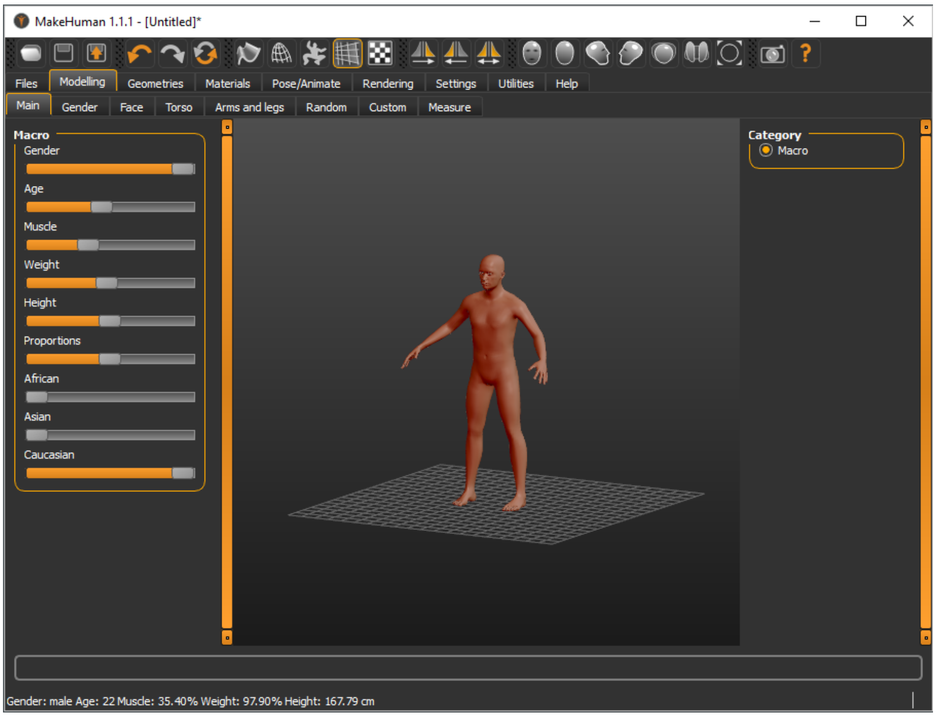


Figure 3.9: The Blueprint graph describing Selvportrett. The graph is zoomed out to get an overview of the complexity of the graph.



Gender: male Age: 22 Muscle: 35.40% Weight: 97.90% Height: 167.79 cm

Figure 3.10: MakeHuman model editor, where an array of parameters are used to define a human model. A larger version can be viewed in Appendix C

3.5.2 Recreating the Visual Style

Generating the visual style of the 3D scene is done through Unreal Engine's own Material Editor, where the styles of different meshes can be designed. The original artwork has been studied, to identify how best to replicate the visual style inside a 3D scene. The original artwork only utilizes different tones of black and white to color the lines of the work. It is likely that a graphite pen or a similar tool was used to produce the style. To replicate this style in 3D, the geometry would have to be painted an even white color, with only the edges highlighted in black. This poses a problem, as detecting geometry edges inside a standard shader, just from reading geometry data is not feasible. The reason for this is the fact that each potential pixel of the resulting image painted on our screen, is processed in isolation, without knowledge of surrounding pixels. When no information is available from neighboring pixels, it becomes impossible to determine wherever the pixel represents an edge or not.

A workaround to this problem is through adding a post-processing step to our rendering technique. To post-process is to add additional processing of an image, after it has been fully rendered by the graphics pipeline. This post-process step will instead of geometry (vertices), take a fully rendered image (pixels) as input to work on. When working on an image or texture in a shader, every pixel is accessible at any time, making it possible to determine if a pixel in the image is an edge or not. There are several methods that can be used to determine edges in an image. In this thesis two of these methods are used in tandem to increase the accuracy of the edge detection. After the scene is rendered through a basic renderer (no lighting, special effects etc.) the depth texture and normal texture (see Figure:3.11) is sent to the post-processing step. In this step edges are detected based on the changes in color from pixel to pixel. The resulting edge pixels identified by analyzing the two images is multiplied together and then used to color the scene either white or black (see Figure:3.12). The specific UE material implementation of the edge detection algorithm is detailed on the official Unreal Engine forum[59]. The method and material code described in the forum was adapted for use in this thesis.

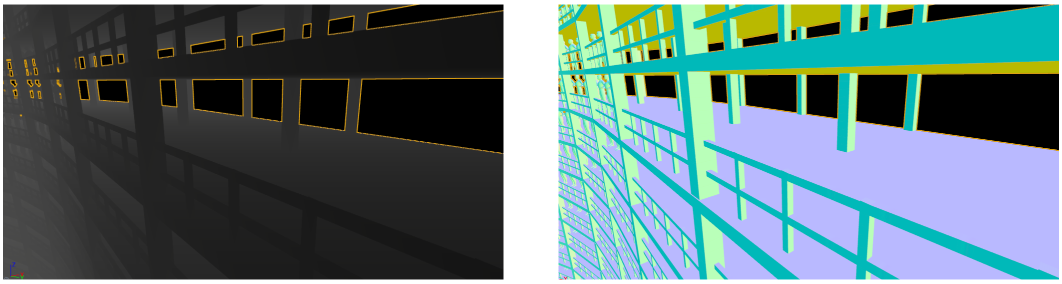


Figure 3.11: Left: The depth information of the rendered scene (the orange highlight is a selection highlight from the editor, and can be ignored). Right: The surface orientation(world normals) information of geometry.)

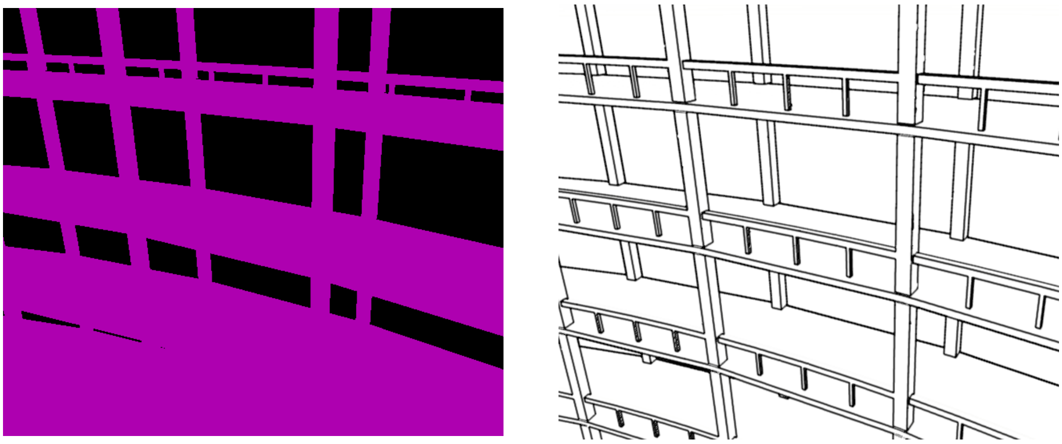


Figure 3.12: Left: Unshaded view of the scene before post-processing. Right: After using depth and normal textures to highlight edges on the scene.

Often when an artists draws by hand, small imperfections in the lines drawn are present from either the tool creating uneven lines, or the artists hands being unsteady. Humans have trouble drawing perfectly straight lines, while computers do this trivially. To introduce an extra layer of realism to the artistic style, and to further replicate the style of Pushwagner, some noise is added to the lines output by the post-processing step (see Figure:3.13). This noise disturbs the lines slightly both vertically and horizontally, creating lines that seem slightly imperfect. The noise is sampled from a high resolution

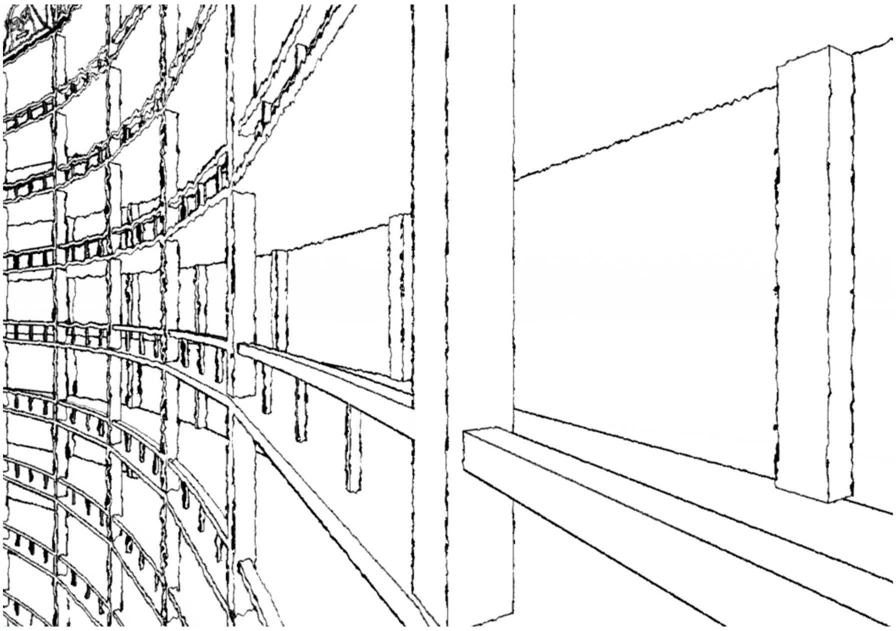


Figure 3.13: Disturbing the lines slightly based on a deterministic source of noise.

image colored with random values. This image serves the purpose of a deterministic source of noise. Since the source of the random values never change, the perturbing of the lines will stay constant from image to image. If true random values were used, then the lines would change between images, producing lines that move erratically, which is unwanted. It is worth noting that appearance transfer (see Section:1.8) could be used to replicate the visual style, as opposed to a manually created shader material. This would require implementing the appearance transfer method inside a shader, which is outside the scope of this thesis. Applying appearance transfer would also cause the lines of the image to change erratically, as the appearance transfer algorithm is calculated for every single (image) frame of the running application, and the algorithm output is not guaranteed to be consistent. Applying appearance transfer on a scene inside VR would most likely result in inconsistent visuals, and as such is not used in this thesis.

3.6 Re-creating The Persistence of Memory

In section 1.3 it was mentioned that a secondary solution should be attempted towards projecting Dali's painting "The Persistence of Memory" into VR. The art-work "The Persistence of Memory" (see Figure:1.2) is geometrically and stylistically different from the previous artworks from Pushwagner. Instead of utilizing the previously described method in this chapter, a completely different approach is taken in attempting to re-create this art inside VR. The Persistence of Memory is a surrealist[60] piece of art. Surrealism is a movement within the art world, where artists attempted to create illogical and disconcerting scenes that still had a sense of photorealism. Dalis art-work introduces soft or melting objects, when in reality, they should be hard and un-yielding. We attempt to recreate part of the artwork in VR. As Dali does not have the structural regularity as Pushwagner, we instead try to reinterpret the work and add "parametric" behaviour by realtime physics based simulation of the melted watches. The simulation technique developed by NVIDIA, called NVIDIA FleX[61] is incorporated into the Unreal Engine. To be able to utilize FleX, a completely separate version of Unreal Engine has to be utilized, as the FleX framework is complex, and needs integration into a system before it's available for use. NVIDIA corporation has already provided a version of the Unreal Engine, where FleX is integrated into the system, allowing users to make use of the simulation technique together with the other functionality given by the engine.[62] A scene is built, and populated with geometry to mimic the work of art by Salvador Dali. First a very simple model of a wall clock was produced inside a 3D modeling application (see Figure:3.14).

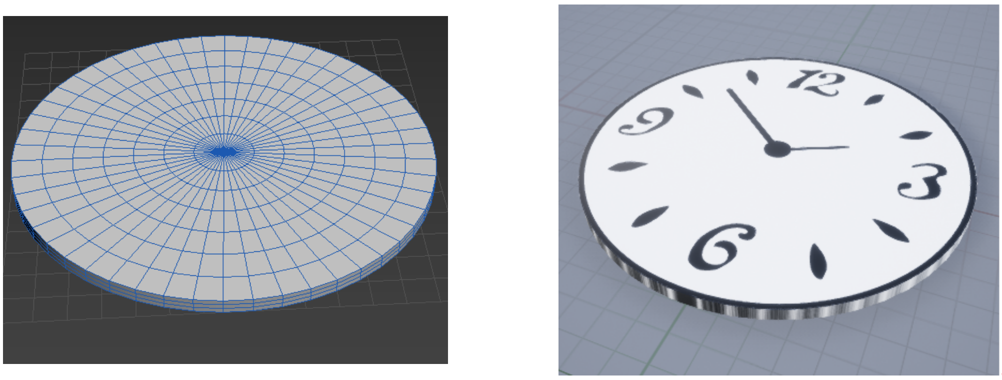


Figure 3.14: Simple mesh of a clock. Left: without texture inside modeling application. Right: textured mesh inside Unreal Engine.

After the model is imported and textured, work was put into configuring the FleX framework to "attach" soft-body physics to the mesh. All physics calculations inside the FleX simulation is done through calculations done on small particles, attached to meshes. The mesh itself is allowed to deform according to the movement and position of the attached particles. A container for these particles is configured and attached to the mesh, shown in Figure 3.15. The physics simulation of the particles is configurable, and are configured in this thesis to be slow-moving, have high friction, and to be non-rigid. Other options include hard and rigid configurations, cloth simulations, and liquid simulations if required. The clock model is brought out into our scene, and the attempt

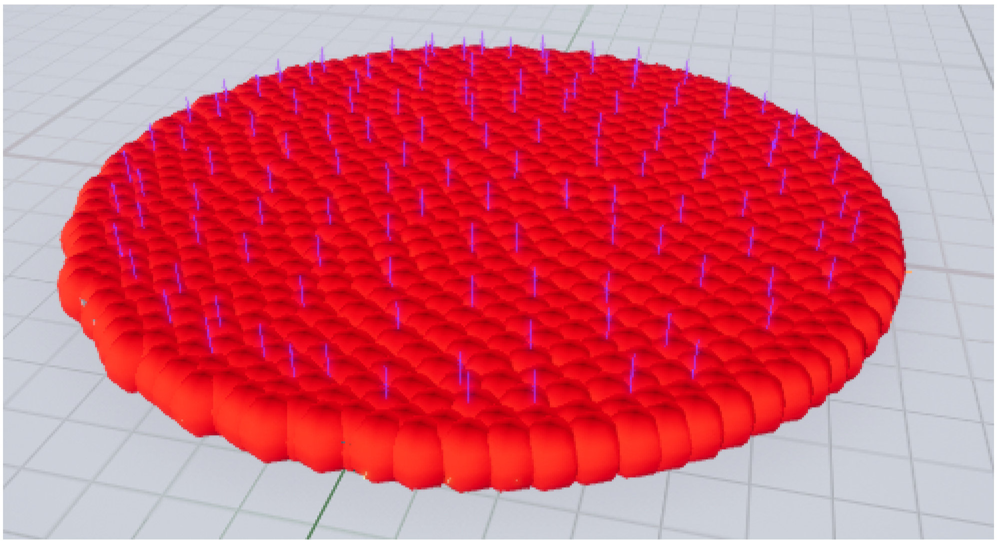


Figure 3.15: The previously solid and non-deformable mesh has hundreds of FleX particles attached to the geometry, allowing it to twist and deform according to calculations performed on the particles.

at projecting The Persistence of Memory into VR is completed by adding support for VR headsets and controllers. The final scene result is seen in Figure: 3.16.

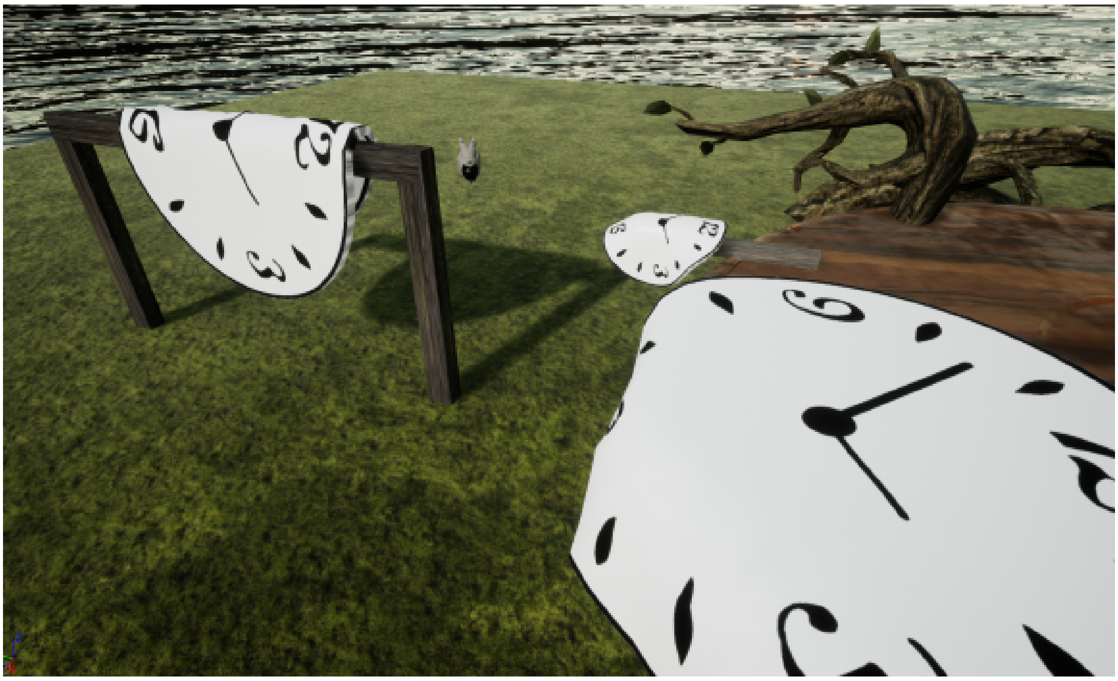


Figure 3.16: Image of the scene inside the Unreal Engine Editor, where some objects are made soft and malleable through physics simulation.

Chapter 4

Results

This thesis has looked at two different parts of creating artistic scenes. One is the geometric aspect, where the artwork is broken down into its constituent geometric shapes. We then try to reproduce this geometry using the final solution from section 3.3.3. The second part is the visual aspect of a work of art. All artworks follow a certain visual style. This style is dictated by both what colors the artist has chosen, and what tools have been used. A graphite pen and a painter brush produce drastically different visual outcomes. In essence we have attempted to imitate the artists style using modern shader language. It's not enough to simply break down a scene into geometries, and replicate visual styles. Most artworks also need a 3D interpretation of the scene. As such some freedom have been taken in interpreting the scenes and how they should be constructed. How this interpretation has affected the end result is detailed for each work. The results of our work are split into three attempted scenes. Two are inspired by the works of Hariton Pushwagner, and one is an original scene without any reference material. The fourth work is Dali's work The Persistence of Memory, developed through a different solution.

4.1 "Selvportrett"(Selfportrait) by Hariton Pushwagner

Likely the most geometrically complex work that we attempt translating into VR is Pushwagners Selvportrett. It's scene includes a huge crowd of humanoid onlookers, all standing in a domed area reminiscent of a stadium. Selvportrett gives off a strong sense of space and scale due to the depicted surroundings. The bottom of the scene seems to stretch downwards endlessly. The result of the generated scene can be seen in the figures, 4.1, 4.2, and 4.3.

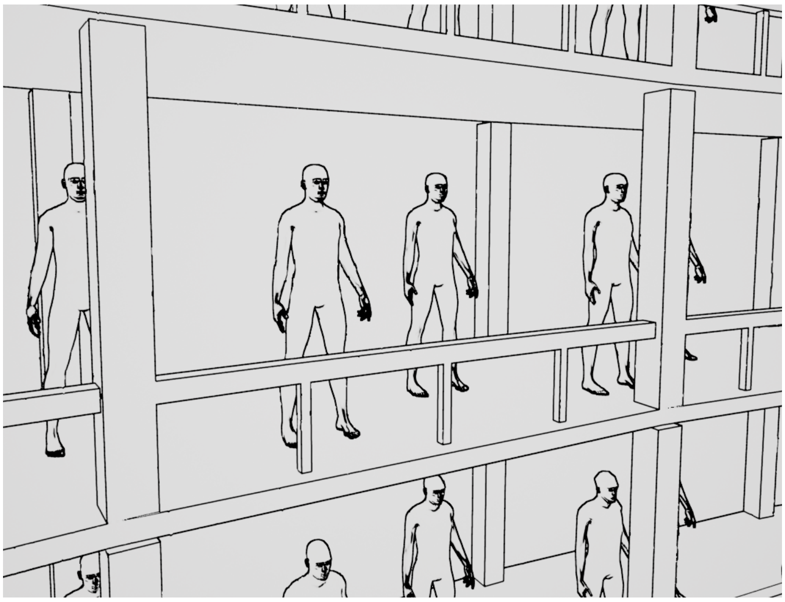


Figure 4.1: Close up of a single floor of the scene

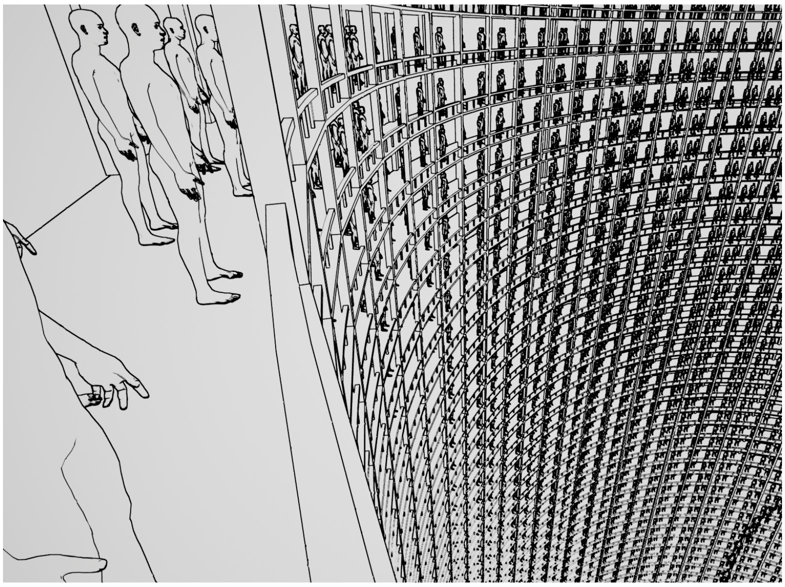


Figure 4.2: Same area seen from the side

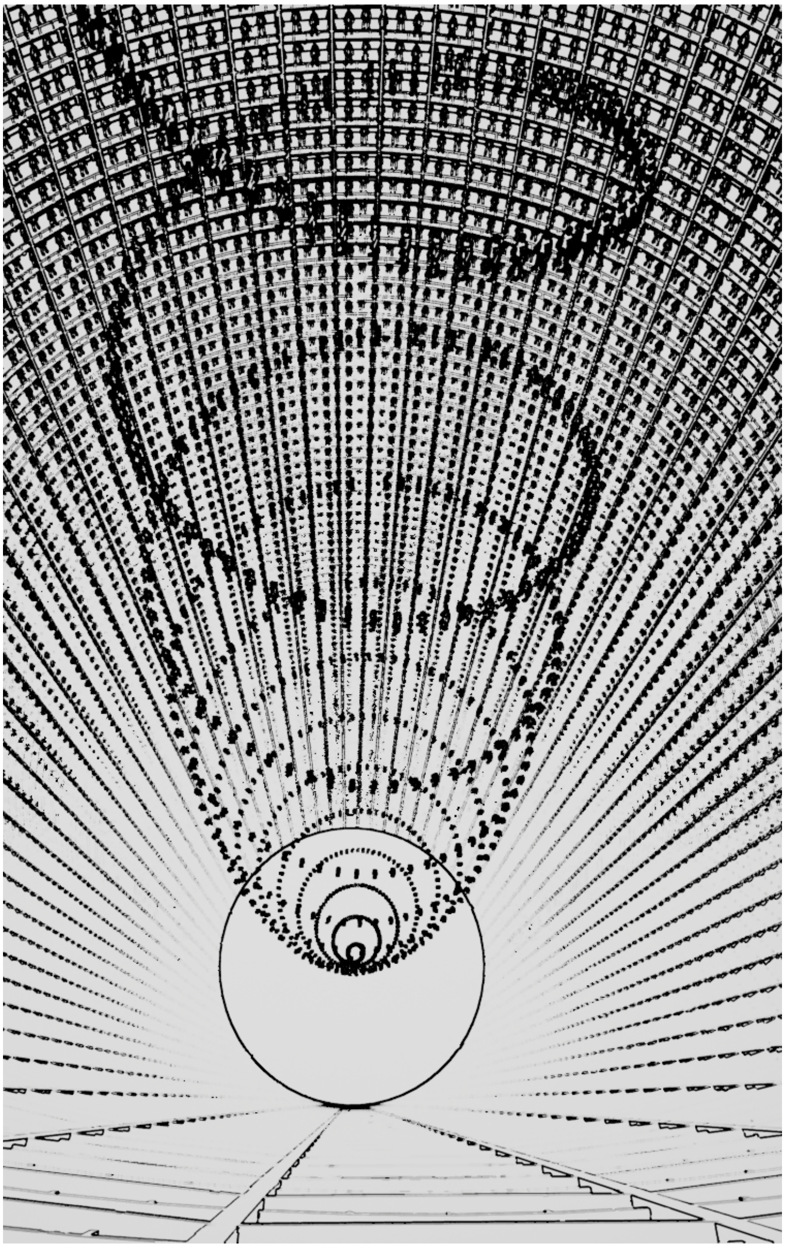


Figure 4.3: Looking down the middle of the scene, with the spiral of humans going down. Note how details get lost in the distance due to the nature of the shading style.

In Figure: 4.3 some of the lines are not drawn near the bottom. Looking far down into the scene, the distance and viewing angle is such, that the depth texture and world normal texture becomes too uniform in depth and color, making it increasingly difficult for the shader to discern edges. It is possible that a completely different approach to recreating the visual style would be needed to eliminate the problem.

4.2 Manhattan by Hariton Pushwagner

The second work Manhattan is simpler in geometry and style. The artwork was interpreted to convey a wavy movement which is incorporated into the scene by dynamically changing the geometry. The scene mostly consists of incredibly tall square shaped buildings. “Manhattan”, much like “Selvportrett” does not depict or indicate any end to the scene, as the entire picture stretches endlessly upwards and downwards and forwards. When translating this to a 3D scene we can not represent this infinity, as it would require us to create an infinite amount of geometry. Despite this problem we still want give the sense of scale and height from the original work. In the artwork Manhattan (see Figure:1.1a) the buildings depicted seem to sway and bend as if they were made of a softer material. This was translated into a swaying motion over time, adding a temporal dimension to the scene. The movement of the buildings was represented by a simple wave function (sinusoidal) based on the time passed inside the application. This movement was applied through accessing the mesh of each building, and moving each vertex slightly over time. The visual style of this scene was produced through standard shading with shadows and light calculations. This means no special post-process effect was applied to achieve this style. It was decided that standard UE rendering was sufficient for the scene, and that most of the style transfer could be achieved by setting the colors on the buildings, and applying shadows.

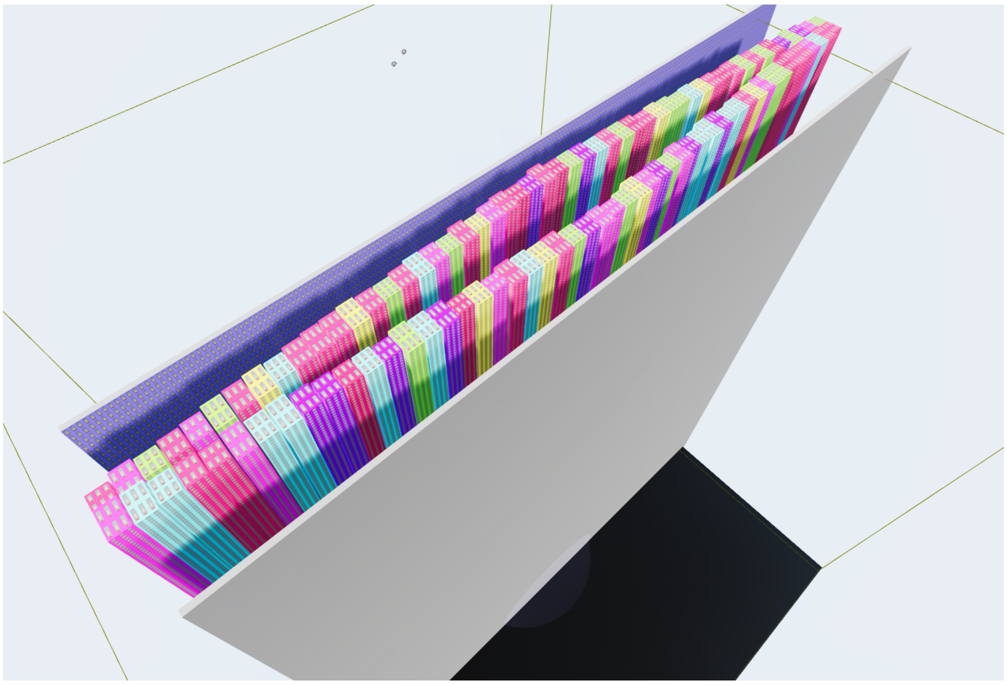


Figure 4.4: A birds eye view of the Manhattan scene, seeing how it is constructed.

Figure 4.4 shows how walls where placed on either side of the scene, this is done to prevent the viewer breaking immersion, by looking between buildings. Without anything blocking the view, the user would look out into a completely empty field, breaking the attempt at conveying the idea of the viewer being enclosed in a maze of buildings.

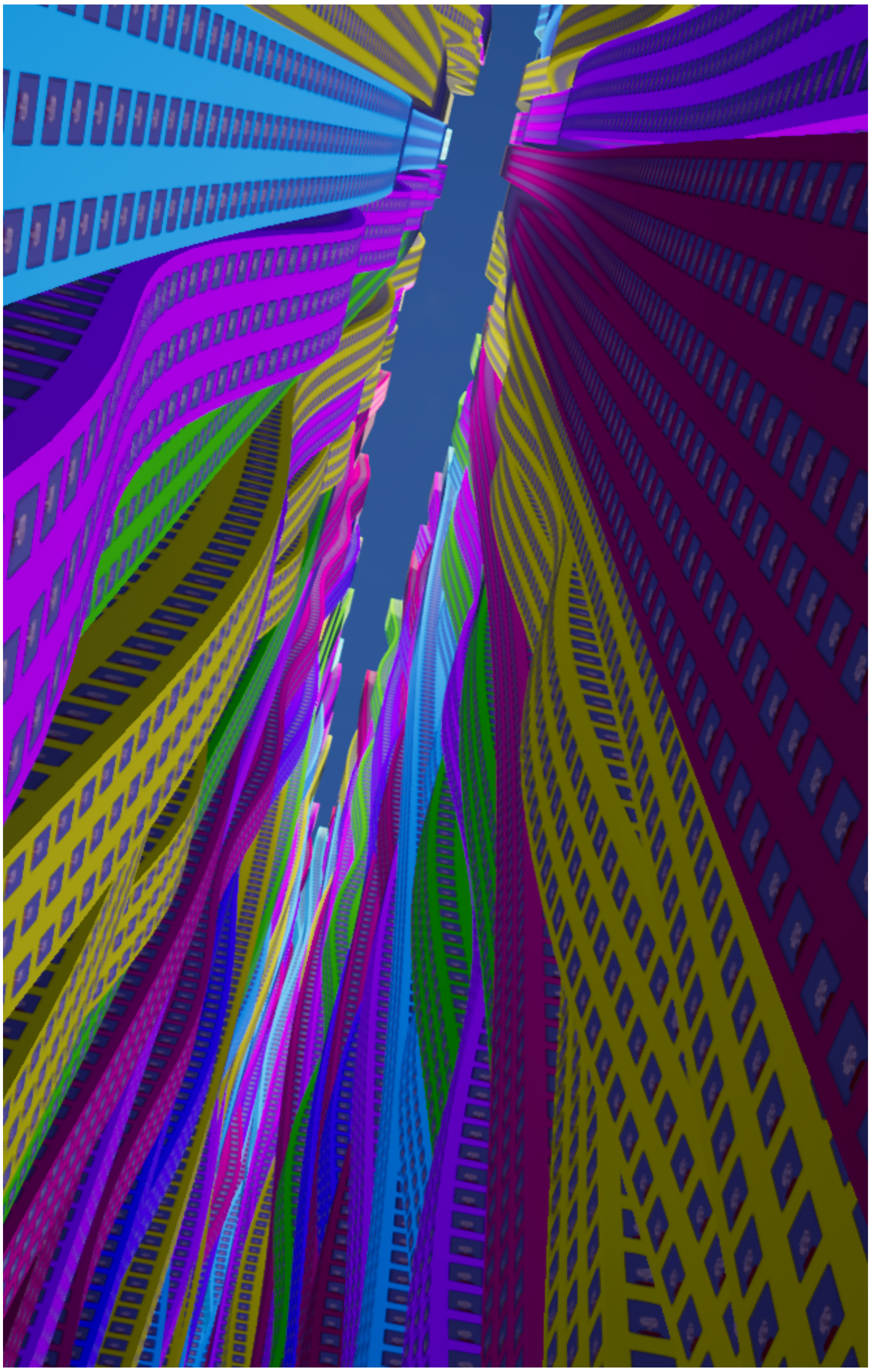


Figure 4.5: Looking up into the sky.

4.3 Original Scene - Reactor

Creating a scene without referring to an external work of art allows us to test the frameworks ability to produce scenes. The created scene was intended to be reminiscent of some futuristic construction. The entire scene rotates around it's own axis, together with the bands of light each rotating along their axis.

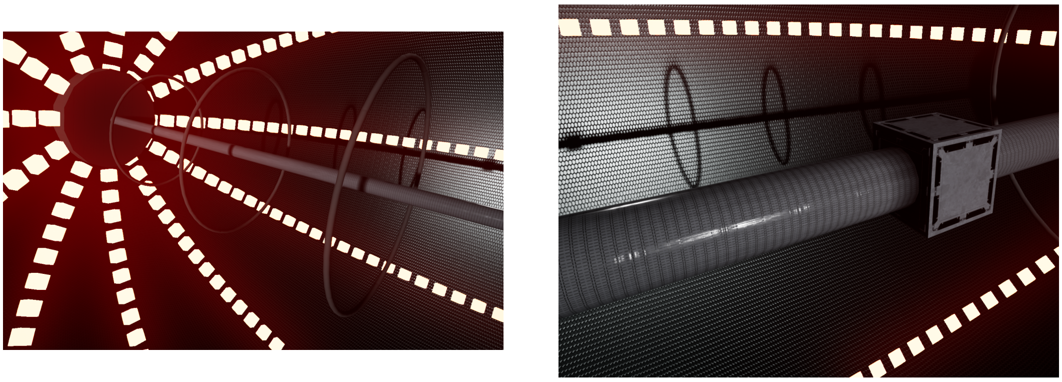


Figure 4.6: Two different viewing angles of our original scene. The smaller cube is where the viewer stands and looks at the surrounding structure.

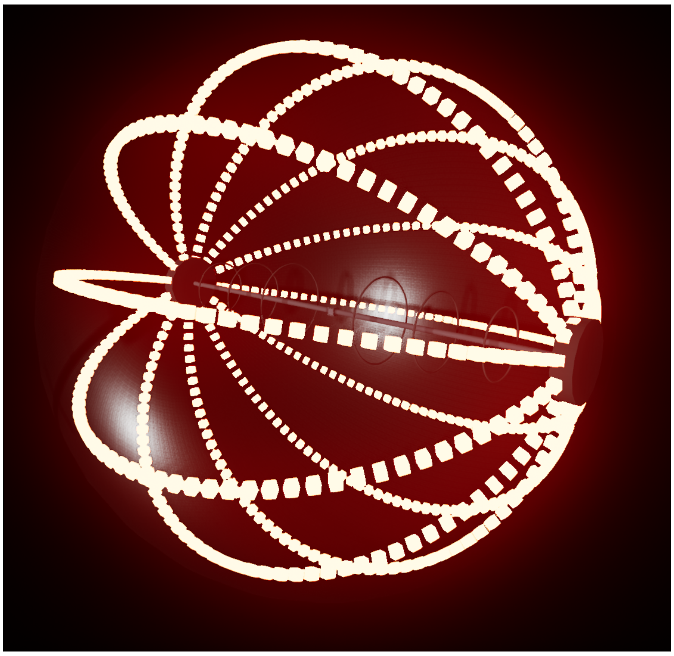
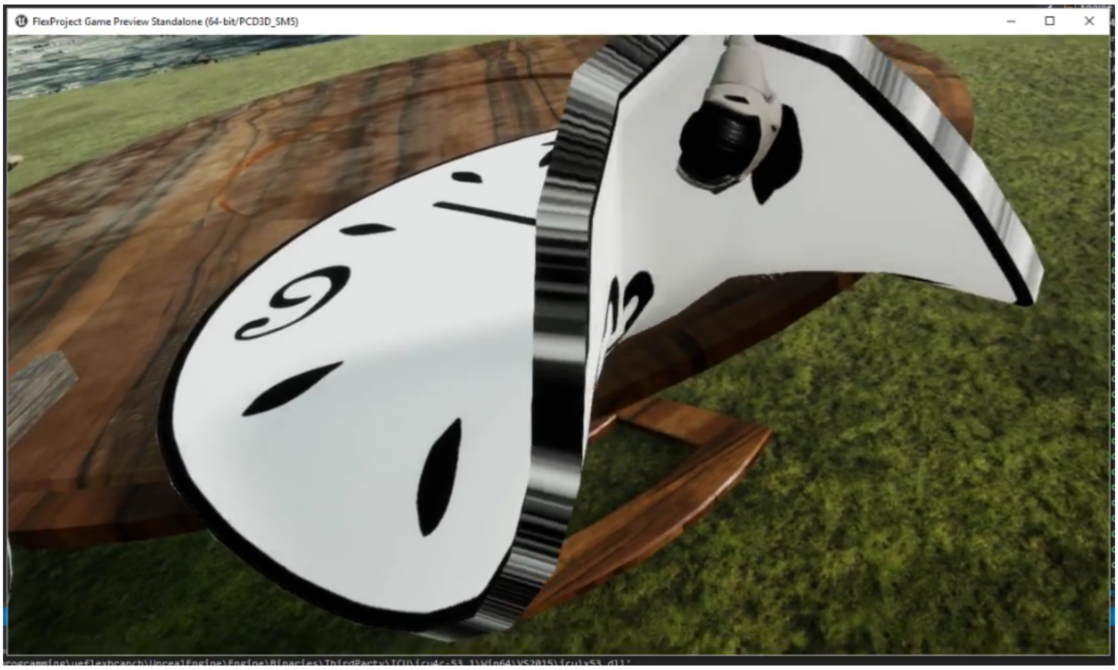


Figure 4.7: An outside view of the scene from further away.

4.4 The Persistence of Memory

For this scene a different method was attempted. The original work of art contains surrealistic and illogical shapes, where clocks, expected to be rigid and hard, appear to be soft and malleable. By using soft-body physics simulation in real-time through NVIDIA FleX, a re-creation is made for use inside VR. The scene performs well in real-time, and the clocks fastened on different surfaces are all intractable through hand controllers. For this scene, no special visual style was incorporated, other than a standard shader with lighting, and textures of grass and water. The original artwork by Salvador Dali has a more simplistic and flat visual style, however this was not pursued in this thesis due to time constraints.



```
rogramming\ueflexbranch\UnrealEngine\Engine\Binaries\ThirdParty\ICU\icu4c-53_1\Win64\VS2015\iculx53.dll'
```

Figure 4.8: The Persistence of Memory scene inside Unreal Engine is intractable using controllers, where the controllers are represented as hands inside the scene.

Chapter 5

Evaluation

A solution used to answer our research questions has been provided through a function library, that can create geometry, this library can be used to compose geometry in various ways to generate a scene that can be viewed in VR. This thesis does not solve a well defined concrete problem, but is more an exploration of VR in new domains. Therefore an split evaluation is done, to get subjective points of view on our work. Our evaluation is divided into two sections. One section is objectively evaluating the method and solution itself, while the other is separated from the solution, focusing on analyzing the feedback provided by a public showing of our results. By evaluating these two parts of the work, an answer to the research questions from section 1.4 is given.

5.1 Evaluating our Method

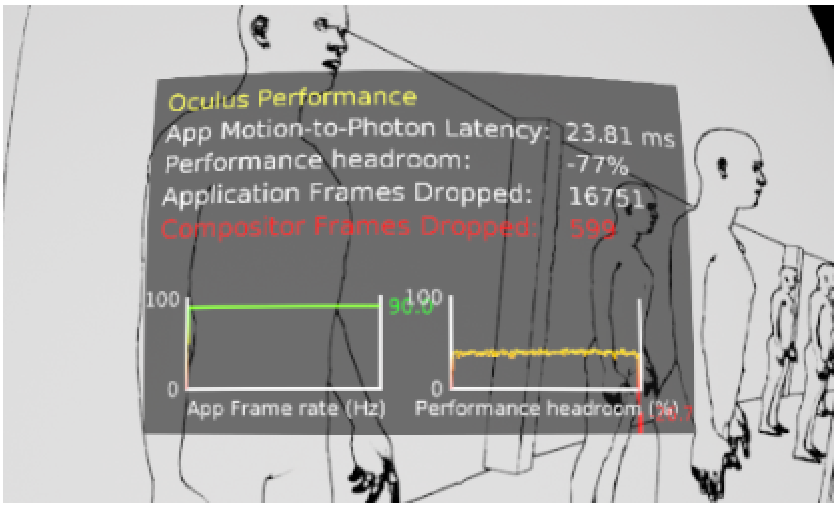
The first part of the evaluation solely focuses on our solution, and how well it serves to reproduce the artistic scenes, together with our own original scene. It might be difficult to assess and evaluate our solution, as we are not solving a specific problem, but instead exploring the possibilities VR has a an artistic medium. The ease of use when a new scene is created is evaluated, and what limits the solution has when it comes to portraying scenes is discussed. The performance of our solution and the produced results is analyzed. We measure how long it takes to create a scene at startup, and how well that scene performs during runtime. The latter is particularly important to achieve a good VR experience.

Table 5.1: Hardware specifications

Hardware	
Processor	Intel Core i7 7700K 4.20GHz
Graphical Processor	NVIDIA GeForce GTX 1080 8GB
RAM	32 GB DDR4 2400 MHz
Motherboard	ASUSTeK PRIME Z270-A
Storage	500 GB NVMe SAMSUNG MZVLW512 SCSI

5.1.1 Measuring Numerical Performance

As mentioned in subsection 2.4.2, the numerical performance of a VR application is important for the experience of the user. This means if our scenes do not perform well enough, the quality of the experience might diminish as a result. The computer hardware that was used to both develop and run the solution is noted in Table:5.1. To help us measure the performance of our scenes we use an external tool called Performance Head-Up Display (HUD) [63]. This tool is part of the Oculus Debug Tool (IDT)[64] package.



Oculus Performance
App Motion-to-Photon Latency: 23.81 ms
Performance headroom: -77%
Application Frames Dropped: 16751
Compositor Frames Dropped: 599

Figure 5.1: The performance head-up display showing performance numbers inside our Selvportrett scene. The runtime performance in this instance stays around a steady 90 frames-per-second.

While we are running the scenes in real-time, the Head-Up display gives us a visual summary of different performance numbers (see Figure:5.1). We care mostly about two specific numbers: Frames Per Second (FPS), and Motion to Photon Latency (MTPL). The meaning behind these numbers are detailed in 2.4.2. The other data in the HMD HUD that is not FPS or MTPL are discarded, as they do not directly affect the user experience. The performance values for each scene is noted down.

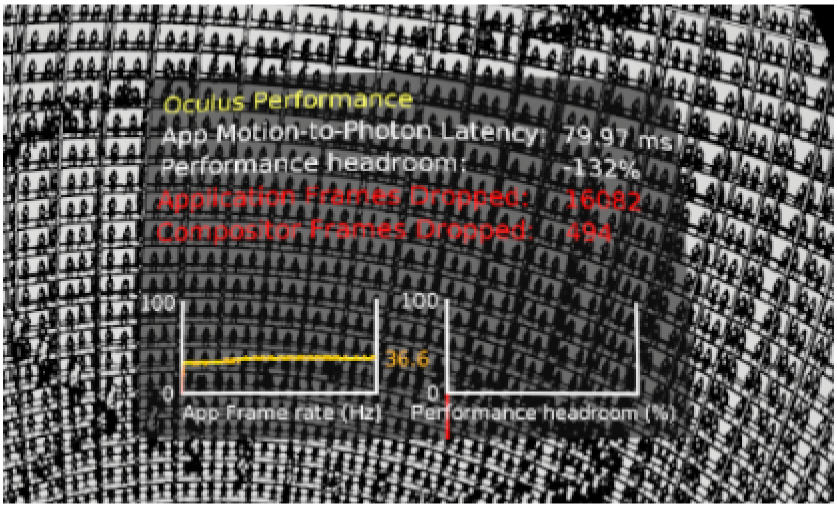
Table 5.2: Runtime Performance

Scene	FPS	MTP Latency
Selvportrett	90fps - 35fps	23ms - 80ms
Manhattan	90fps - 89fps	29ms - 33ms
Original scene	90fps - 87fps	22ms - 23ms

The numbers in table 5.2 represent the frame-per-second and motion-to-photon latency of each scene, as recorded while manually testing the scenes. As a basis of reference we know that the official recommended FPS for VR is 90 FPS (see section 2.4.2), anything below this should be considered non-optimal, but is still passable. When we go below 60 FPS the low frame-rate becomes very noticeable, and can quickly lead to discomfort and nausea. MTPL is different, in that lower numbers are better. A optimal MTPL number for good VR presence is stated to be below 20ms[65]. Anything above this target up to 60ms is considered passable. If the MTPL rises above the upper bound, it might lead to discomfort or sense of presence being lost for the user. The data in table 5.2 show that both Manhattan and our original scene have optimal FPS, with a steady frame-rate of approximately 90. The MTPL is also well within acceptable range, with Manhattan having the high latency most likely due to it's heavier CPU usage from the animated motion. This was expected, as neither Manhattan or our own scene are very complex or demanding on the hardware. The numbers in Table:5.3 show the triangle count for each scene attempted. Despite Selvportrett having a very high triangle count, it performs well, most likely due to the use of LOD and geometry instancing. In total for Selvportrett, 5400 individual human meshes were instanced around the scene. Note that with our parametrized scenes we can easily change our scenes to be much more taxing on the hardware, bringing the performance numbers down drastically. We have however chosen the scenes (in this case only Manhattan) to be as close to the original, without sacrificing performance. Selvportrett as a scene is very different, as the heavy use of instancing brings our performance down drastically. Instantiating thousands of models in the scene makes it difficult for the hardware to keep up with the 90 FPS target number. Depending on where the user looks, the amount of instances in view changes, making the frame-rate vary wildly from 90 to 35 FPS. The drop in performance depending on viewing angle is noticeable within VR, but has not incurred any great amount of discomfort or nausea. The only scene where the MTPL exceeds 60ms is Selvportrett. This increase

Table 5.3: Triangle count for each scene

Scene	Triangle Count
Selvportrett	51 651 304
Manhattan	186 472
Original Scene	17 276



```
Oculus Performance
App Motion-to-Photon Latency:  79.97 ms
Performance headroom:          -132%
Application Frames Dropped:    16082
Compositor Frames Dropped:     494
```

Figure 5.2: When looking out into the center of the scene, the frame-rate drops significantly, down to roughly 35 frames-per-second. The MTPL latency also increases drastically.

in lag is not very noticeable, likely due to the low amount of hand interactions in the scene.

5.1.2 Creating and Altering Scenes

Creating 3D scenes using parametric function nodes is fast and allows for the creator to be experimental as the scene is being created. The ability to view a scene as it is constructed, helps the creator evaluate changes made to the scene, as any change is represented immediately in the editor. By constructing the scene through a node-graph, it is possible to change any part of the creation process, making possibly drastic changes to the scene in a short span of time. As a practical example, the resulting scenes were re-designed many times during this thesis. The process of creating a new scene became shorter and more efficient as the function library grew more powerful. The final resulting scenes shown in chapter 4 did not take more than a few hours to create each. The negative part of function nodes is the fact that all scenes must be constructed with the provided nodes and geometries as building blocks for the scene. This is potentially limiting, as it is difficult to cover every possible user scenario with a limited function library. This problem can be fixed partly by giving users the ability to define their own custom nodes, but this solution does not eliminate the problem entirely.

As the complexity of a scene grows, the amount of nodes and wires needed to represent it grows with it. This increase of nodes produces visual clutter where it becomes increasingly difficult to maintain the graph.

Careful structuring of the graph and use of sub-graphs is effective in reducing visual clutter, as seen in Figure: 5.4, and Figure: 5.5.

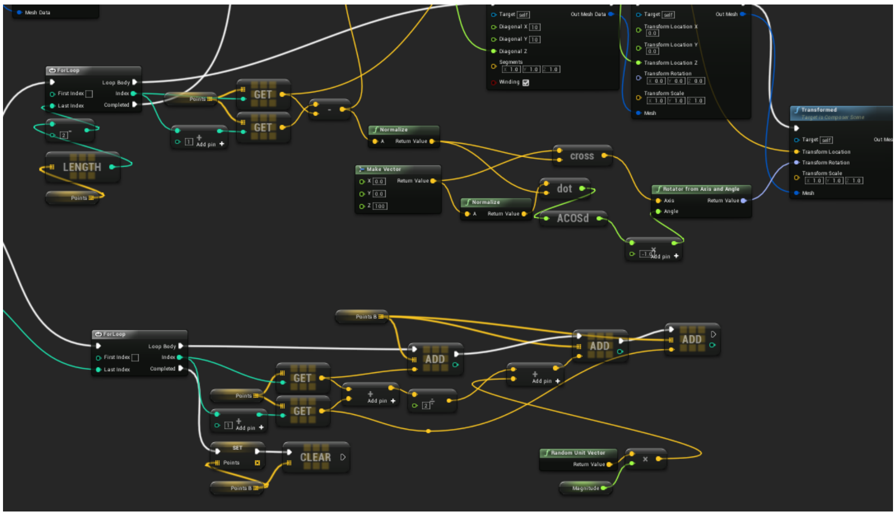


Figure 5.3: A large amount of nodes, making it more difficult to get an overview of the graph.

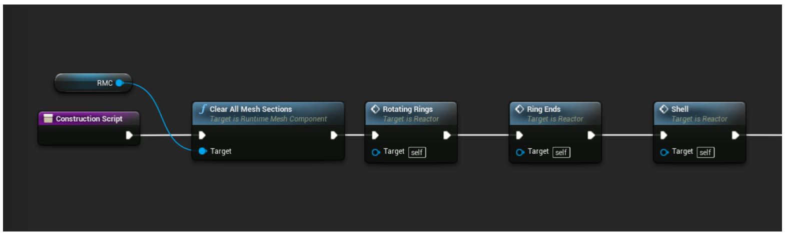


Figure 5.4: Using sub-graphs to reduce clutter in the scene. Each node represents multiple nodes organized into a sub-graph.

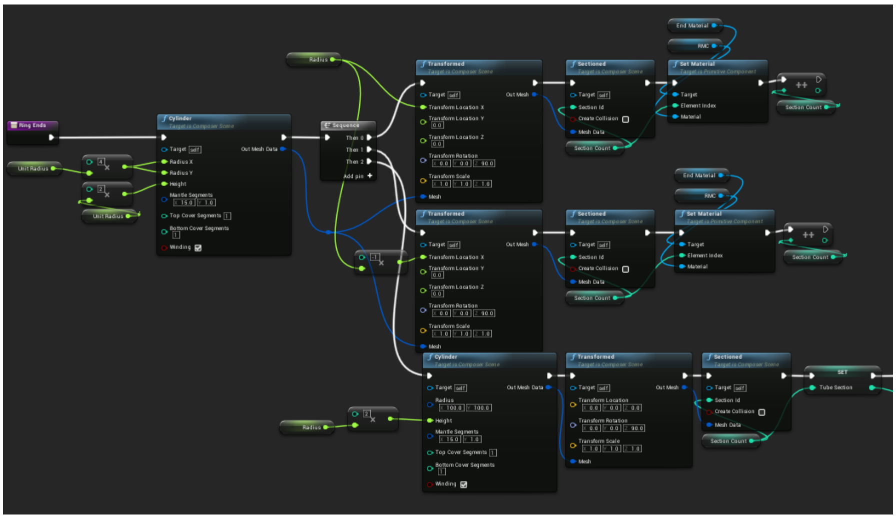


Figure 5.5: Expanded view of one of the sub-graphs from Figure 5.4.

5.2 Evaluating our Feedback

We do not focus solely on the numerical evaluation of this solution. Our goals and questions are also focused on the perceived experiences that users have when viewing our scenes compared to the original. To acquire some usable data on this matter we conducted a public demo session, where any person interested could come and test our scenes inside VR (see Figure:5.6). The demo was located at the Faculty of Fine Art, Music and Design [66]. The reason for choosing this location was wanting to have qualified respondents from the local art community. Since our thesis is partly art related, gaining some extra insight from people that study or teach art related subjects would be valuable.

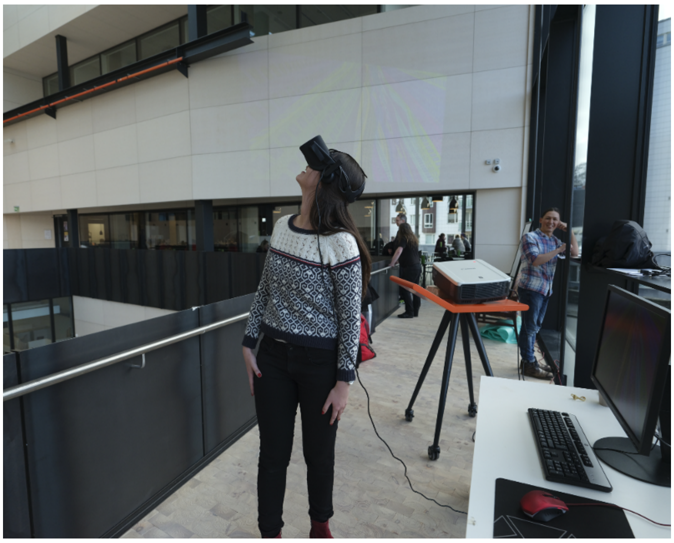


Figure 5.6: Art student viewing one of our scenes from our demo booth. Behind the projector is the original Selvportrett image, and on the wall behind is projected the scene that the user is looking at.

5.3 Public Demo

A suitable location at the faculty was allocated, and a small booth was erected. Anyone among the faculty could come over, put on the headset, and would be walked through the two scenes that were available at the time. Some context surrounding our work would be provided, including showing a original print of selvportrett so they could directly compare the artwork with the VR version. After being presented with the original work they would be allowed to view the scenes. First the selvportrett (see Figure:1.1a) scene, then the *Manhattan* (see Figure:1.1b) scene. After the demo the participant would be prompted to answer a sheet of seven questions. The questions are listed as such:

1. Are you a student or do you have higher academic rank?
2. What are your initial impressions of the scenes?
3. How do you think the scenes work, compared to the original art?
4. Do you think the scenes capture the spatial experience of the original scenes?
5. What do you think my role is in this production, am I an artist?
6. Do you think virtual reality changes the possibilities in artistic expression?
7. Do you use, or do you think you will use VR tools to create art? If so, why?

The seven questions were prepared to be related to our research questions posed at the start of our thesis. This way, by analyzing the response, we could start to answer some of our research questions. The reason we ask for initial impressions is to create a picture of their more immediate reactions. If they respond that the scene is underwhelming or fail to impress them in any-way, then we would have failed in trying to create a passable VR experience. Here the users have the opportunity to voice any possible issues they could have with the experience. We also ask them to compare the VR scene to the original artworks, giving us some insight into how viewing the work of art in 3D changes the experience, as opposed to viewing it on a 2D surface.

5.3.1 Response

Most of the response from the demo session was positive, and we acquired some insight into what opinions people from the art-community have surrounding VR as an artistic medium. Since our questions were answered in free-text and not on a numerical scale, it becomes difficult to present the results with graphs our figures. Instead an analysis of the response was made, and then condensed down into a trend for each question asked.

5.3.2 Oral Feedback

In addition to the written question response, we also take note of the immediate response of the users. Some of the most genuine and un-altered response one can gather from these demos are the initial impressions viewers have, in the instant they see the scene inside VR. Some of these oral responses have been noted down.

- ”Yes, now I really am inside it.”
- ”Woa, I don’t dare look down.”
- ”With the movement (in the scene) I could look at this for a long time.”
- ”Are there only two scenes?”

- ”Freaky! So you’ve made this in 3D? That must have taken a lot of time!”

It’s clear from the oral response that our scenes invoked a feeling of immersion and presence in the scene immediately. This is positive, mostly because many artists look after ways to deliver a message, or invoke a feeling in the viewer. If our VR manage to invoke a stronger reaction, then it would indicate that our solution have been successful in translating these traditional works of art over to VR.

5.3.3 Written Feedback

The question answers are summarized, and a trend for each question asked is provided. When examples or quotes are given from the list of answers, these will be abridged and translated. All written feedback is also provided un-abridged and un-altered, in appendix A.

Question 1: Are you a student or do you have higher academic rank?

Overall 4 out of 17 answers to this question stated they had higher academic rank. 12 out of 18 respondents stated they were students at the faculty.

Question 2: What are your initial impressions of the scenes?

This question got a generally positive response. Eight of the fourteen responders mentioned the immense sense of scale and height they got from the scenes. Comments were made on the detail and execution of the scene, and how it is impressively constructed. One respondent commented how the scene appeared out of focus or slightly blurred out. This is likely due to the loss of detail that happens at distant parts of the scene as a result of our method of replicating the originals visual style. (See section:) Three respondents mentioned how they initially felt disconcerted emotionally by the scenes, as they had a scary or eerie atmosphere. Three respondents reported they felt small or insignificant when viewing the scenes.

Trend Impressed by the sense of large scale and the detail of the scene. Certain respondents felt disconcerted or insignificant.

Examples

- ”Huge, large, space”.
- ”Well thought out and executed”
- ”Spacious interesting, spooky”

Question 3: How do you think the scenes work, compared to the original art?

Here we get a sense of how the artistic scene changes when translated from a 2D canvas over to a 3D scene in VR. Most response talks of how it enhances what the artist probably tried to convey, that of scale in both space and number. Both Selvportrett and Manhattan have massively repeating elements, and this element is preserved in our digital scene. According to the response, the viewer is transported from a passive viewing positing from outside the work, to a more active position, where they can move and see the work from different angles. Many stated that this effect heightens the experience of the work. Both question one and three had a respondent mention that the VR scenes appeared blurry, this could either result from the monitor resolution from our HMD not being high enough, or the visual effect placed on our scenes.

Trend What was portrayed by the original 2D artwork is transferred and for many enhanced beyond the original work.

Examples

- ”It's much more vivid than it's original work.”
- ”It definitely adds a dimension to the original art that should have been there all along.”
- ”You immediately realize what artwork it is, and you feel you have truly entered the artworks.”

Question 4: Do you think the scenes capture the spatial experience of the original scenes?

The response to this question was almost uniformly positive. 11 out of the 13 answered the question with some form of ”yes”. Four out of 15 respondents expressed that the VR scene enhanced the experience, beyond the original. One answer argued the perspective of the scene was not captured due to the original having a ”fish eye perspective”. By changing the viewers perspective inside the application this effect could be replicated.

Trend Keeps the experience, and for many viewers, enhances it.

Examples

- ”Yes definitively.”

- ”Yes - feel even larger”
- ”Not quite. In the original it seems like the viewer has a lot more in a fish eye perspective. To get this experience I think the camera ”lens” should have a lower focal length.”

Question 5: What do you think my role is in this production, am I an artist?

Five out of Fifteen answers expressed disbelief towards me, the creator of the scenes, being an artist. Translating a work into a different medium is not enough to make oneself an artist. According to some of response, we are people who have enough knowledge about this technology to perform this translation, however, that is not to say that one is an artist. In the event that something original was created in VR, then we could allow ourselves to be the artists. Three out of Fifteen meant that i am an artist, and one answer in particular, from a respondent with higher academic rank, argues towards me having the role of an artist. Art is about imitation, and ”giving form to an shape where there was none”. This is in contrast to many other answers, that argued simply imitating a style and translating it over to a different medium is not enough to set my role as an artist. Other respondents argued that instead of an artists role, i had the role of a translator (from medium to medium), or a facilitator of translating the art. The divide between answers makes it difficult to state with confidence that i am an artist, however a discussion around the topic can be made.

Trend Arguments for and against me being an artist is made. Translating a work from 2D to 3D might not be enough for me to be considered an artist. Different roles are suggested, such as translator or facilitator.

Examples

- ”In this context, adapter.”
- ”No, you are facilitator.”
- ”Well yeah, is not all art about imitation? To give form to an shape where there was none? To create is to be a creator and that is to be an artist, coding or not.”

Question 6: Do you think virtual reality changes the possibilities in artistic expression?

Mostly answers in the positive. A few comment how, while VR gives new experiences, it also makes users dependent on more hardware than before. The biggest effect of VR is the ability to transport the viewer into a different role in the scene, from static to active. VR as a medium will not replace any other medium according to respondents, but will stand as a new medium with entirely new possibilities.

Trend Expands the possibility of what is possible for artistic expression.

Examples

- "Yes. & new mode of expression."
- "Yes. Definitively. It opens up new worlds of opportunities never ever imagined before."
- "Definitively, but also makes you depend on more tools like cables, glasses, pc etc."

Question 7: Do you use, or do you think you will use VR tools to create art? If so, why?

This question gave a more mixed response. Many would like to attempt using VR tools, but not on a full time basis. Most mention VRs ability to give the viewer a increased feeling of presence in the scene. This makes VR a good tool to leave more lasting impressions. Others who did not want to utilize VR in their work mentioned how they prefer their traditional tools.

Trend Both yes and no. VR is still under development, and needs to be advanced further for it to become more viable for users that are used to more traditional tools.

Examples

- "It will depend on the context, but it's a possibility."
- "No. I don't like making digital art. And i got dizzy."
- "Yes. Architecture projects."

Question Summary

We have shown our result to qualified respondents and they provided answers to our questions. Their initial impressions of our scenes were varied, with many stating they were impressed by the detail, and answered the experience was immersive. Most of the respondents answered that our scenes captured the original artworks well, and that the feeling of massive scale and size that the original art attempted to portray was kept intact, if not enhanced. Some respondents commented that the active viewing angle provided by VR gave them a feeling of their own insignificance or uneasiness. This suggests that our scenes might have something added to them that was not present in the original 2D works. Many respondents meant that VR is a new medium suitable for artistic expression. The ability to much more strongly involve the viewer inside a work of art or virtual scene makes VR into a valuable tool for future artists. The willingness of the respondents to use VR as an artistic tool in the future were mixed. Some state their preference for non-digital art as their reason for not choosing VR as their medium. Others felt that VR as a technology was very interesting, and could see it as a tool for expressing art and design. When asked if I, the producer of the scene was an artists, the response was mostly "No". Several arguments were made towards us not being artists because we merely translated the work over to a different medium. In this sense I was adapting art, not creating it. One answer argued that simply creating the scenes put me in a creator role, and this would qualify me as artists.

5.4 Feedback Related to Research Questions

The research questions posed in section 1.4, are re-iterated here:

1. How can a two dimensional work of art be interpreted through virtual reality?
2. What happens when a two dimensional work of art is transformed through these processes?

An attempt at answering these questions can be made, as we have evaluated our solution and the public response from our demo.

How can a two dimensional work of art be interpreted through virtual reality? We have iterated through three solutions (see section: 3.3) towards creating artistic scenes inside VR. This directly relates to the first research question, of how we can interpret from two dimensions to three dimensions. Our first solution described in section 3.3.1, is closer to other previous attempts at interpreting a know work of art, an example of this is (...) detailed in section: (...). Here modeling tools are used to create the scene in a manual process. Our third solution from section: 3.3.3 produced our results, and utilizes parametrization with a visual language to describe the produced scenes. Our solution distinguishes itself from other attempts, as they are different in their methods and tools used in the production. The evaluation of the performance, comfort, and ease-of-use of our solution stated that the scenes produced are fully viewable in VR, and represent the original artworks to a sufficient degree. The analysis of collected response from the public demo (described in 5.3) supports this statement. The solution is also generic enough that it is not limited to a small set of artworks with a specific style. This solution could be applied to other works of art, and would still be able to reproduce a scene of sufficient quality.

What happens when a two dimensional work of art is transformed through these processes? How a known work of art might change depending on the medium it is represented in, could be particularly interesting for artists. By asking questions and gathering answers from qualified respondents, we have gained some insight into how our scenes might have changed when adapted to a 3D medium, viewed inside virtual reality. It is evident from both immediate oral response, and written response, that a viewer of our scene in VR is transported to an active viewing position inside the artwork. This active viewing position brings a different perspective on the art, and can help an artist invoke emotions that might have been more difficult to achieve on a traditional 2D medium. Feelings of space, scale, and movement are transmitted well in particular through VR as is suggested by the response from users that viewed our scenes.

Chapter 6

Conclusion

In my work I have successfully attempted to find a solution towards translating 2D works of art into a temporal and navigable 3D space, viewable in VR. In addition to this goal, I wanted the solution to be powerful and flexible enough to create my own original scenes, without using reference artwork. I wanted to reconstruct the artworks by parametrization, which would allow us to describe our scenes with function parameters, and have the computer interpret these descriptions and construct the scene for us. By altering the description, changes to the scene could be made quickly with minimal effort. The translation from 2D to 3D is split into two parts. One part focused on geometry reconstruction, and the second part focusing on keeping the visual style of the artwork as it was translated over from 2D. A solution that fulfills the goal of creating parametrized artistic scenes in VR is proposed.

Creating a new scene starts with creating a blank blueprint that inherits the functions from our function library. From this blank blueprint we can start inserting and connecting nodes to form our desired geometry. By changing the input to the nodes or their composition, we alter the resulting output. The changes in our description is represented in the model as we make them. This makes prototyping scenes easier, as we can see the output of our description as we construct it.

6.1 Geometry and Visual Style

For this thesis the method focused on splitting a work of art into two separate parts. One part looks at the geometry of the original artwork, and attempts to recreate this geometry in 3D space. The second part focuses on the visual style of the art, and how the art style of something projected from 2D to 3D can be kept intact between mediums. This way of splitting the focus of an artwork into two separate parts, to make it easier to translate it to a different medium, is effective. It is possible that this method could be applied to a wide range of differing art-works with different styles.

6.1.1 Geometry

Our solution depends on visual scripting using nodes and parameter inputs to describe a scene and it's constituent geometry. We used the Blueprint system of Unreal Engine 4.0 to prototype a library of function nodes designed towards constructing geometry and placing them in our scene. Using this method we constructed the geometry of three scenes, viewable in VR.

6.1.2 Visual Style

Utilizing the power of modern shaders we attempted to transfer the visual style of the works of art into a 3D context. The result of this was a graphical style that resembled the original to a high degree. It's clear that modern shader technology is powerful enough, that with some effort we can imitate the style of at-least the stylistically simple (flat-coloring, black-and-white) work of art we chose as reference material.

6.2 Performance and Comfort

Using tools provided by Oculus VR[63], we tested our scenes in VR, and gathered performance data during the tests. Our most complex and demanding scene was the scene translated from Selvportrett (See Figure:1.1a). This scene required several thousand individual humans to be shown in frame at any given time. With the use of hardware instancing and LOD, we managed to populate the scene with 5400 individual human models (90 models per floor times 60 floors total), totaling to an average of 39 million triangles in view at any given time. Depending on the view of the user in VR the FPS would change drastically, but would never reach a level that was un-acceptable (see Section:5.1.1). Even though some scenes were demanding on our hardware, we still achieved a passable VR experience that could be safely displayed to a public audience. We observe that modern hardware instancing and LOD techniques are capable of rendering very geometrically detailed scenes in VR, without diminishing the experience to an un-acceptable level.

6.3 Reception of Results

We held a public demo at University of Bergen, Faculty of Art, Design and Music[66]. Where we asked viewers a set of questions related to our thesis. The reception was positive overall, with more than one respondent wanting to see more scenes produced by our solution. When describing their impression of our scenes compared to the original works they were based on, they described feeling a greater sense of presence and sense of scale. The original works of art contained heavily repeated elements, and large spacious environments. This large scale in both the size of the scene and amount of visible elements was fully present in our 3D scenes, according to respondents. Many had the opinion that the more active viewing position gained from actually viewing the scene from the inside through VR, changed their look on the artistic works. Some respondents described feeling small and insignificant in the scene, or being disconcerted by the works. These are things that were not present when viewing the original works on a 2D surface. VR is an excellent medium for conveying space, distance, and scale. This shows that it's possible when using VR as an artistic medium, for viewers to gain a sense of presence in the artwork, which in turn could give viewers new perspectives on the art.

Chapter 7

Further Work

7.1 Expanding Function Library

We produce the geometry of our scenes from a function library that we developed as part of our solution (see Section:3.3.3). The complexity limit of the scenes we make is directly related to the limits of our function library. Splines, axis-symmetry, and nodes for Constructive Solid Geometry (CSG), are all ideas for ways to expand the power and flexibility of our library. By adding more functions for geometry we help eliminate some of the problems currently present in our solution surrounding the production of scenes with less regularity present.

7.2 Custom Scene Editor

One possible expansion of our solution would be the creation of a custom editor for composing the scenes produced. This editor would be designed so users without previous knowledge of computer graphics or computer science could use it. If VR is to become a suitable platform for artists to express themselves, then the tools created for them also need to be designed with artists in mind. Artists could have different needs than a architect or game designer, these needs have to be identified and accounted for. In our solution we utilize Unreal Engine Blueprints as a node editor, but it is possible that with a custom solution, we could achieve less visual clutter for complex scenes. A much more extensive solution for a custom editor would be an editor that worked inside VR. If a user could put on a HMD, and start our the editor, he/she would be able to view the scene inside VR as it is being constructed. The advantages of viewing a 3D scene inside VR while you construct it, is the ease of creating the scene at a correct scale. When editing a scene on a 2D monitor, it is not immediately apparent to the creator how the scene will look and feel inside VR. By viewing it in VR while we construct the scene, we immediately get a sense of how the scene is going to appear when other users view it. To achieve this we would have to construct a tool that integrates virtual reality from the start, where a user can create, edit, and connect nodes using interactive controls in our application. This would give creators a 3D space to place their nodes inside, which could possibly help eliminate visual clutter, and make the tool more intuitive for users with less experience in 3D design, and VR.

Bibliography

[1] Epic Games. *Unreal Engine Home Page*. 2018. URL: https://www.unrealengine.com/en-US/what-is-unreal-engine-4 (visited on 02/27/2018).

[2] *Pushwagner*. 2018. URL: https://www.pushwagner.no/ (visited on 04/25/2018).

[3] The Editors of Encyclopaedia Britannica. *Salvador Dali*. 2018. URL: https://www.britannica.com/biography/Salvador-Dali (visited on 05/28/2018).

[4] Tilman Osterwold. *Pop art*. Taschen, 2003.

[5] *SELVPORTRETT*. 2018. URL: https://www.pushwagner.no/galleri/kunst/SELVPORTRETT-MONOKROM_1_1_1_1_1_1 (visited on 04/25/2018).

[6] *Manhattan*. 2018. URL: https://www.pushwagner.no/galleri/kunst/MANHATTAN_1_2_1 (visited on 04/25/2018).

[7] Phaidon. *Salvador Dali's The Persistence of Memory explained*. 2018. URL: http://uk.phaidon.com/agenda/art/articles/2016/may/10/salvador-dalis-the-persistence-of-memory-explained/ (visited on 05/22/2018).

[8] Wikisource. *A Compendium of Irish Biography/Barker, Robert — Wikisource*. [Online; accessed 30-January-2018]. 2011. URL: %5Curl%7B%20https://en.wikisource.org/w/index.php?title=A_Compendium_of_Irish_Biography/Barker,_Robert&oldid=2553696%20%7D.

[9] *Panorama Cross Section*. 2018. URL: https://lh4.googleusercontent.com/vjSYy7kzNmhTBID-UaW3mXwTu0TTIbWocMfR9mIzQRV_qakspcEIRWEhg6HWuAgUMpf52jHlTvjspjOKDbeW
ohWzjnFqCY9nGgh-9JmVUEpYr2zjwENaeRPGQx4UNj6xrFGn2tCA (visited on 05/31/2018).

[10] H.M. L. *Sensorama simulator*. US Patent 3,050,870. 1962. URL: https://www.google.com/patents/US3050870.

[11] Eunji Oh, Minkyoung Lee, and Sujin Lee. "How 4D Effects Cause Different Types of Presence Experience?" In: *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*. VRCAI '11. Hong Kong, China: ACM, 2011, pp. 375–378. ISBN: 978-1-4503-1060-4. DOI: 10.1145/2087756.2087819. URL: http://doi.acm.org/10.1145/2087756.2087819.

[12] Nikita Fedorov. *The History of Virtual Reality*. 2018. URL: https://www.avadirect.com/blog/the-history-of-virtual-reality/ (visited on 05/31/2018).

[13] *Ivan E. Sutherland.* 2017. URL: `http://www.invent.org/honor/inductees/inductee-detail/?IID=530` (visited on 05/10/2018).

[14] Ivan E. Sutherland. "A Head-mounted Three Dimensional Display". In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I.* AFIPS '68 (Fall, part I). San Francisco, California: ACM, 1968, pp. 757–764. DOI: `10.1145/1476589.1476686`. URL: `http://doi.acm.org/10.1145/1476589.1476686`.

[15] The CCCU Psychology Programme Blog. *IvanSutherland-sword-of-damocles.* 2018. URL: `http://cccupsychology.com/blog/2017/08/17/virtual-reality-a-brief-history-current-trends-and-future-directions/ivansutherland-sword-of-damocles/` (visited on 05/31/2018).

[16] E. M. Lidal et al. "A Decade of Increased Oil Recovery in Virtual Reality". In: *IEEE Computer Graphics and Applications* 27.6 (Nov. 2007), pp. 94–97. ISSN: 0272-1716. DOI: `10.1109/MCG.2007.141`.

[17] *The VR Museum of Fine Art.* 2016. URL: `http://store.steampowered.com/app/515020/The_VR_Museum_of_Fine_Art/` (visited on 01/16/2018).

[18] Dr. Thomas Girst. *Cao Fei - BMW M6 GT3, 2017.* 2017. URL: `http://www.artcar.bmwgroup.com/en/art-car/text/Cao-Fei-BMW-M6-GT3-2017-10063.html` (visited on 05/22/2018).

[19] Cao Fei. *Cao Fei.* 2018. URL: `http://www.caofei.com/about.aspx` (visited on 05/22/2018).

[20] *Tilt Brush.* 2018. URL: `https://www.oculus.com/experiences/rift/1111640318951750/` (visited on 01/16/2018).

[21] *Oculus Medium.* 2018. URL: `https://www.oculus.com/experiences/rift/1336762299669605/` (visited on 01/16/2018).

[22] Google. *Tilt Brush.* 2018. URL: `https://www.viveport.com/apps/bbbc73fc-b018-42ce-a049-439ab378dbc6` (visited on 05/31/2018).

[23] Motion Magic VR. *The Starry Night VR.* 2018. URL: `https://samsungvr.com/view/4cS0iSadKdm` (visited on 05/23/2018).

[24] Motion Magic VR. *Motion Magic VR.* 2018. URL: `https://samsungvr.com/channel/57de2e01b62eb1001ae12e43` (visited on 05/23/2018).

[25] Goodby Silverstein & Partners. *Dreams of Dali.* 2018. URL: `http://thedali.org/exhibit/dreams-vr/` (visited on 05/23/2018).

[26] Goodby & Silverstein. *We Are GS&P.* 2018. URL: `https://goodbysilverstein.com/about/we-are-gsandp-3` (visited on 05/23/2018).

[27] Philip Galanter. "What is generative art? Complexity theory as a context for art theory". In: *In GA2003–6th Generative Art Conference.* Citeseer. 2003.

[28] Casey Edwin Barker Reas. *Information.* 2018. URL: `http://reas.com/information` (visited on 05/28/2018).

[29] *Processing.* 2018. URL: https://processing.org/ (visited on 02/13/2018).

[30] Casey Reas. *Network D.* 2018. URL: http://reas.com/network%5C_d%5C_p1/ (visited on 05/31/2018).

[31] Roaring Tide Productions. *What is Archimatix?* 2018. URL: http://www.archimatix.com/ (visited on 05/21/2018).

[32] Roaring Tide Productions. *Gallery.* 2018. URL: http://www.archimatix.com/gallery (visited on 05/21/2018).

[33] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *CoRR* abs/1508.06576 (2015). arXiv: 1508.06576. URL: http://arxiv.org/abs/1508.06576.

[34] *Barycentric Coordinates.* 2018. URL: http://mathworld.wolfram.com/BarycentricCoordinates.html (visited on 04/24/2018).

[35] Song Ho Ahn. *OpenGL Rendering Pipeline.* 2018. URL: http://www.songho.ca/opengl/gl_pipeline.html (visited on 05/31/2018).

[36] Emma McDonald. *The Global Games Market Will Reach 108.9 Billion Dollars in 2017 With Mobile Taking 42 Percent.* 2017. URL: https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/ (visited on 10/25/2017).

[37] *List of game engines.* 2018. URL: https://en.wikipedia.org/wiki/List_of_game_engines (visited on 04/20/2018).

[38] *Unity Product Page.* 2018. URL: https://unity3d.com/unity (visited on 04/19/2018).

[39] *What is Unreal Engine 4.* 2018. URL: https://www.unrealengine.com/en-US/what-is-unreal-engine-4 (visited on 04/19/2018).

[40] Microsoft. *Introduction to the CSharp Language and the .NET Framework.* 2017. URL: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework (visited on 12/06/2017).

[41] Mozilla. *JavaScript.* 2018. URL: https://developer.mozilla.org/bm/docs/Web/JavaScript (visited on 05/10/2018).

[42] *Unity 5.0 Release Notes.* 2015. URL: https://unity3d.com/unity/whats-new/unity-5.0 (visited on 01/15/2018).

[43] Epic Games. *Essential Material Concepts.* 2018. URL: https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/IntroductionToMaterials (visited on 05/15/2018).

[44] Online Emythology Dictionary. *virtual (adj.)* 2018. URL: https://www.etymonline.com/word/virtual (visited on 05/15/2018).

[45] Valve Corporation. *Welcome to Valve.* 2018. URL: http://www.valvesoftware.com/company/ (visited on 05/11/2018).

[46] *Oculus Rift.* 2018. URL: https://www.oculus.com/rift (visited on 04/23/2018).

[47] Nick Pino. *Oculus Rift review.* 2018. URL: https://www.techradar.com/reviews/gaming/gaming-accessories/oculus-rift-1123963/review (visited on 05/31/2018).

[48] Qiaozhi (George) Wang. *An Overview of Tracking Technologies for Virtual Reality.* 2018. URL: https://www.linkedin.com/pulse/overview-tracking-technologies-virtual-reality-qiaozhi-george-wang/ (visited on 05/31/2018).

[49] Ben Lang. *Including Controllers, Vive and Rift Could be Evenly Matched on Price.* 2018. URL: https://www.roadtovr.com/including-controllers-htc-vive-and-oculus-rift-could-be-evenly-matched-on-price-touch/ (visited on 05/31/2018).

[50] *Google Cardboard.* 2018. URL: https://vr.google.com/cardboard/ (visited on 04/23/2018).

[51] Aleks Buczkowski. *Google Cardboard-paper Virtual Reality set now supports Street View.* 2018. URL: http://geoawesomeness.com/google-cardboard-paper-virtual-reality-set-now-supports-street-view-app/ (visited on 05/31/2018).

[52] Stephane Bouchard et al. “Exploring new dimensions in the assessment of virtual reality induced side effects”. In: *Journal of computer and information technology* 1.3 (2011), pp. 20–32.

[53] *Powering The Rift.* 2015. URL: https://www.oculus.com/blog/powering-the-rift/ (visited on 01/16/2018).

[54] *Oculus Rift CV1 Teardown.* 2018. URL: https://www.ifixit.com/Teardown/Oculus+Rift+CV1+Teardown/60612 (visited on 01/16/2018).

[55] Jingbo Zhao et al. “Estimating the motion-to-photon latency in head mounted displays”. In: *Virtual Reality (VR), 2017 IEEE.* IEEE. 2017, pp. 313–314.

[56] *Runtime Mesh Component.* Github repository. URL: https://github.com/Koderz/RuntimeMeshComponent.

[57] Conway. *Chris Conway Koderz Github Page.* 2018. URL: https://github.com/Koderz (visited on 03/05/2018).

[58] MakeHuman community. *MakeHuman.* 2018. URL: https://bitbucket.org/MakeHuman/makehuman/overview (visited on 05/23/2018).

[59] Guthmann. *Edge Detection PostProcess Feedbacks.* 2018. URL: https://forums.unrealengine.com/development-discussion/rendering/121539-edge-detection-postprocess-feedbacks (visited on 05/30/2018).

[60] André Breton. “Manifesto of surrealism”. In: *Manifestoes of surrealism* 15 (1924).

[61] NVIDIA Corporation. *NVIDIA FleX.* 2018. URL: https://developer.nvidia.com/flex (visited on 05/22/2018).

[62] NVIDIA Corporation. *NVIDIA GameWorks and UE4.* 2018. URL: https://developer.nvidia.com/nvidia-gameworks-and-ue4 (visited on 05/22/2018).

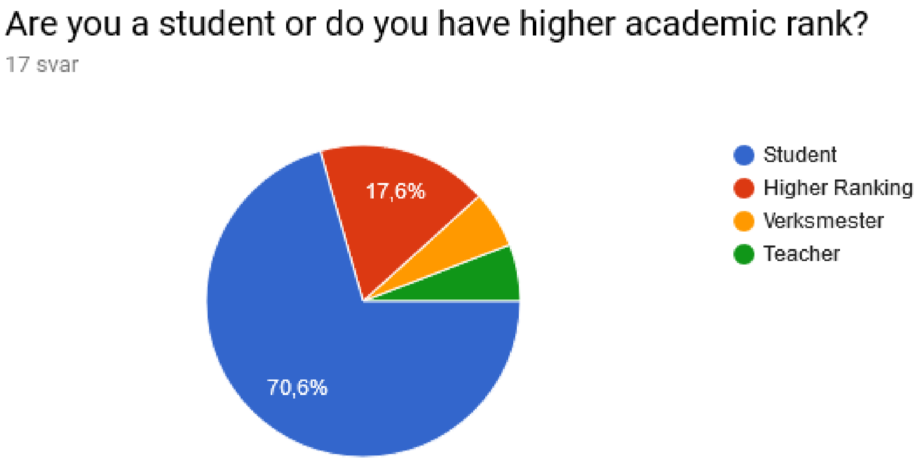
[63] *Performance HeadUp Display.* 2018. URL: https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-hud/ (visited on 05/06/2018).

[64] *Oculus Debug Tool.* 2018. URL: https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-hud/ (visited on 05/06/2018).

[65] J. Zhao et al. "Estimating the motion-to-photon latency in head mounted displays". In: *2017 IEEE Virtual Reality (VR).* Mar. 2017, pp. 313–314. DOI: 10.1109/VR.2017.7892302.

[66] *Faculty of Fine Art, Music and Design.* 2018. URL: https://kmd.uib.no/en/about-the-faculty-of-fine-art-music-and-design (visited on 05/04/2018).

Appendix A



Are you a student or do you have higher academic rank?

17 svar

QUESTION 2: WHAT ARE YOUR INITIAL IMPRESSIONS OF THE SCENES?

1: Interesant og fasinerende

2: That they are very realistic despite the simplicity of the design

3: Huge, large, space

4: Well thought out and executed

5: Proper depth, Sc 1: slightly blurry but fascinating, Sc 2: calming, sharp

6: Vertigo in a good way. Very impressive. I liked it. Just another for expression.

7: Fullført skikkelig. Ble transportert dit med en gang. Bevegelse og følelse.

8: Very vertigo? It was a new thing for me so very exiting.

9: Massive, kinda scary. It feels like I'm being part of a cell.

10: BIG! Environments deep and high

11: The scene's is a bit creepy, makes me feel small but also makes me curious.

12: Space, height and after that i realized I'm one of those people standing. (in first picture). it felt like being in some futuristic movie. I felt small.

13: Begge scenene fokuserer på ein slags evighet og det uendelige. Med høgdeskrekk so var eg litt satt ut, men eg føler at scenene fokuserer på at eg berre er ein liten del i eit uendelig stort univers.

14: Imponerende og beskrivende

15: Spacious, interesting, spooky

QUESTION 3: HOW DO YOU THINK THE SCENES WORK, COMPARED TO THE ORIGINAL ART?

1: Et sterkt utrykk som gir en god merverdi til orginalverket. Denne form for kunstformidling har et stort potensiale i å formidle kunst til et nytt publikum.

2: Yes. But they are something entirely different. I think sound could and should be further explored as an element to enhance the experience.

3: Gives a feeling of hight, space

4: It definitely adds a dimension to the original art that should have been there all along

5: You immediately realize what artwork it is, and you feel you have truly entered the artworks.

6: It gives and brings other dimensions. We are integrated in the work. Maybe another way to enjoy art.

7: 2D ---> 3D, going from reading the art to being apart of it.

8: Added a new layer but kinda blurry

9: It's much more vivid than It's original work

10: Better representation of depth and multiplictron(unreadable)

11: The scene's work good compared to the original. Because it makes you feel small and not important.
You are just a small piece of a big pie.

12: Well you get much better 3D feeling because you look at it from a different perspective (from inside of a scene)

13: Det å dra inn betrakter som ein del av verket forsterker noe det fokuset kunstverket vil vise.

14: Komplimenterer kverandre veldig

15: Compared to the artwork it gave a different perspective. You look upon the scene and the animation, you're part of it, one of the many (people there).

QUESTION 4: DO YOU THINK THE SCENES CAPTURE THE SPATIAL EXPERIENCE OF THE ORIGINAL SCENES?

1: Ja

2: Not quite. In the original it seems like the viewer has a lot more in a fish eye perspective. To get this experience I think the camera "lens" should have a lower focal length.

3: Yes - feel even larger

4: Yes definitely

5: Yes

6: Definitely. Hope to see more in the future.

7: Yes =)

8: Yes

9: Yes!

10: Yes in my opinion "better".

11: Yes I do.

12: Yes and even better. I actually were afraid to fall :)

13: da, og eg føler også at det forsterker det i høg grad!

14: Ja

15: It impacts the body differently. You feel the space by experiencing it compared to just viewing it.

QUESTION 5: WHAT DO YOU THINK MY ROLE IS IN THIS PRODUCTION, AM I AN ARTIST?

1: Din rolle er lener seg på et eksisterende verk og er av kunstnerisk karakter.
Dette kan: 1. sees på å ha en formiddlingsrolle for det eksisterende verk. 2. Eller være en kommentar som refererer til verket.

2: No. You are the tech wizard

3: In this context, adapter

4: (blank)

5: You are not of the art, but the creator of the world, so yes and no.

6: No, you are facilitator.

7: Learning technique skills needed to take a 2D term and translate it to 3D. Agitating atmosphere

8: Kinda the artist re-imagining another one? kinda half way an artist.

9: Not sure. The image cave vividly when you turn it into VR. but it would be nicer if you had more opinion in this scene.

10: Yes. Interesting to use someone else's work. But would be nice to see you "own" visions as well

11: Your role would be to make the people feel the impression pushwagner wants to show.

12: If you make something your own, but now you just copied so you have skills to communicate visually but can you express yourself as an artist?

13: På sett og vis. Detter er jo basert på eit ferdig kunstverk og nesten kopiert, men som betraktar, skapte denne produksjonen noko anna og verket fekk ein anna betydning, derfor meiner eg at du fungerer som ein kunstnar.

14: Ikke når scenene er basert på en annes kunstverk. Hadde du lagd noe "original", så ja!

15: Well yeah, isn't all art about imitation? To give form to an shape where there was none? To create is to be a creator and that is to be an artist, coding or not.

QUESTION 6: DO YOU THINK VIRTUAL REALITY CHANGES THE POSSIBILITIES IN ARTISTIC EXPRESSION?

1: Ja så absolutt. Tror vi går en spennende tid i møte.

2: Yes. Definitely. It opens up new worlds of opportunities never ever imagined before.

3: Yes indeed

4: (blank)

5: Yes

6: Maybe

7: Yes. & new mode of expression

8: Yes

9: Yes. It will come to people directly and it can give us a lot of confusion, for example sensory, etc.

10: Yes.

11: I don't know maybe. It will give a new dimension and experience so that you can find new ways.

12: Definitely, but also makes you depend on more tools like cables, glasses, pc etc.

13: VR gjer nettop dette med å forflytte betraker frå ei statisk stilling til ei nesten aktiv rolle i verket. Dette gjer at ein får ei heilt anna rolle som publikum, så ja! Eg trur VR endrar uttrykka ganske så voldsomt

14: Ja du kan gjøre ting som vanligvis ville vært umulig. (Hadde blitt enda kulere med lyd.)

15: Yep! It will not replace anything but creates a different kind of artistic expression

QUESTION 7: DO YOU USE, OR DO YOU THINK YOU WILL USE VR TOOLS TO CREATE ART? IF SO, WHY?

1: Ja. Med tiden vil dette være verktøy og et medie som fler og fler vil ta i bruk. Ennå er dette relativt nytt og for en spesiellt interesert gruppe. Etter min mening kommer mulighetene og bruken at VR til å eksplodere og bli en naturlig del i hvordan vi kommer til å komunisere og sammhandle med hverandre.

2: Yes. Because it as a medium visualizes presence and challenges scale at a new level. Some people even call VR the empathy machine Look here: https://www.ted.com/talks/chris_milk_how_virtual_reality_can_create_the_ultimate_empathy_machine

3: Not me, but my students try out, present space surfaces / light

4: (blank)

5: Yes. Because it's fun and the future.

6: It will depend on the context. but it's a possibility.

7: Yes. Architecture projects

8: It would add a new layer so definitely an possibility

9: Yes, but it's hard to answer. I can say that it widens possibility. So why shouldn't i try it!

10: No. Maybe, who knows!

11: No. i Don't like making digital art. And i got dizzy.

12: Unfortunately I don't think so.

13: Sjølv studerer eg møbeldesign og romarkitektur, men også her er mulighetene mange for å oppleve rommet rundt oss. Eg kjem nok til å vere innom VR men at eg jobber med det på kvart prosjekt trur eg ikkje.

14: Nei, fordi eg ikke er kunstner, men kansje for å vise frem et designobjekt.

15: Personally, no, I neither have the tools or the knowledge or the motivation to.

Appendix B

GEOMETRIES

CUBOID	Generates cuboid mesh
ELLIPSOID	Generates ellipsoid mesh
CYLINDER	Generates cylinder mesh
PIPE	Generates pipe mesh
CAPSULE	Generates Capsule mesh
TORUS	Generates Torus mesh
TORUS KNOT	Generates Torus Knot mesh

POINT FUNCTIONS

CIRCLE	Points in a circle
RAY	Points along a finite ray
WAVE	Points in a wave pattern
SQUARE	Points in a square shape
SPIRAL	Points along a spiral
NOISE	Points with randomized positions
ROTATED	Rotation of points in 3D space

MESH MODIFIERS

CLIPPED	Clips a mesh into two separate parts along any given plane
TRANSFORMED	Transforms a mesh (translation, rotation, scaling)
STAPLE	Moves target mesh onto a destination mesh surface, and performs a merge
SNAP	Performs a staple but without merging the mesh
INSTANTIATED	Geometry instantiates a mesh at a given list of points.
MERGED	Merges two meshes into a single mesh
REPEATED	Input mesh is repeated a given number along the [X Y Z] axes
SECTIONED	Inserts a mesh into an available RMC section.
COLORED	Set color values for a mesh
MIRRORED	Mirrors a mesh around a given point
CENTERED	Centers a mesh to a given point
COPIED	Copies a mesh to a list of points
ROTATING	Rotates a mesh (Realtime)
WAVING	Applies a wave movement to a mesh.

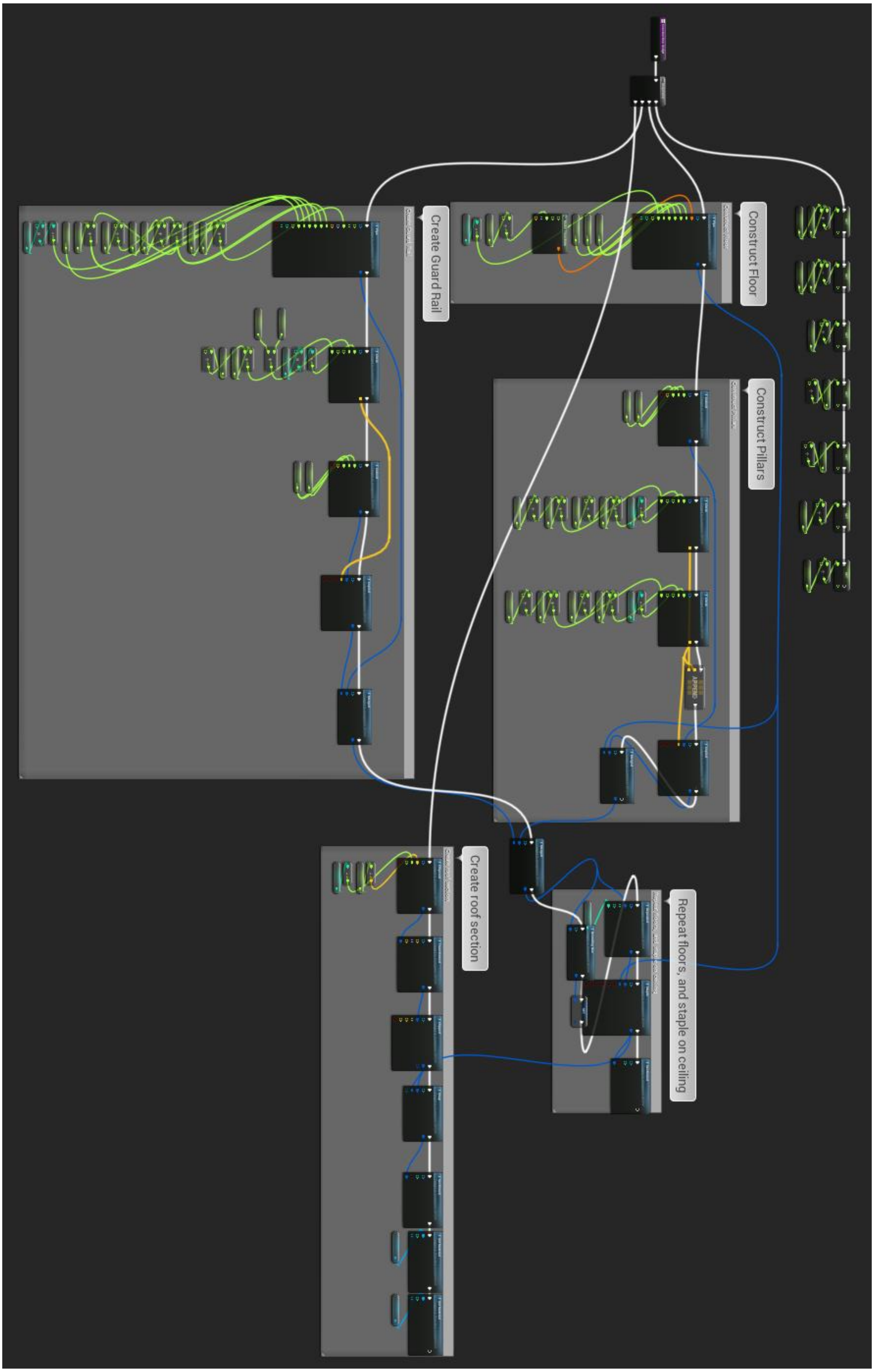
MISCELLANEOUS

MAKE MESH DATA	Creates empty mesh data structure
BOUNDING BOX	Calculates and returns bounding box of mesh
CENTER AND EXTENT	Calculates mesh center and outermost point

SECTION FUNCTIONS

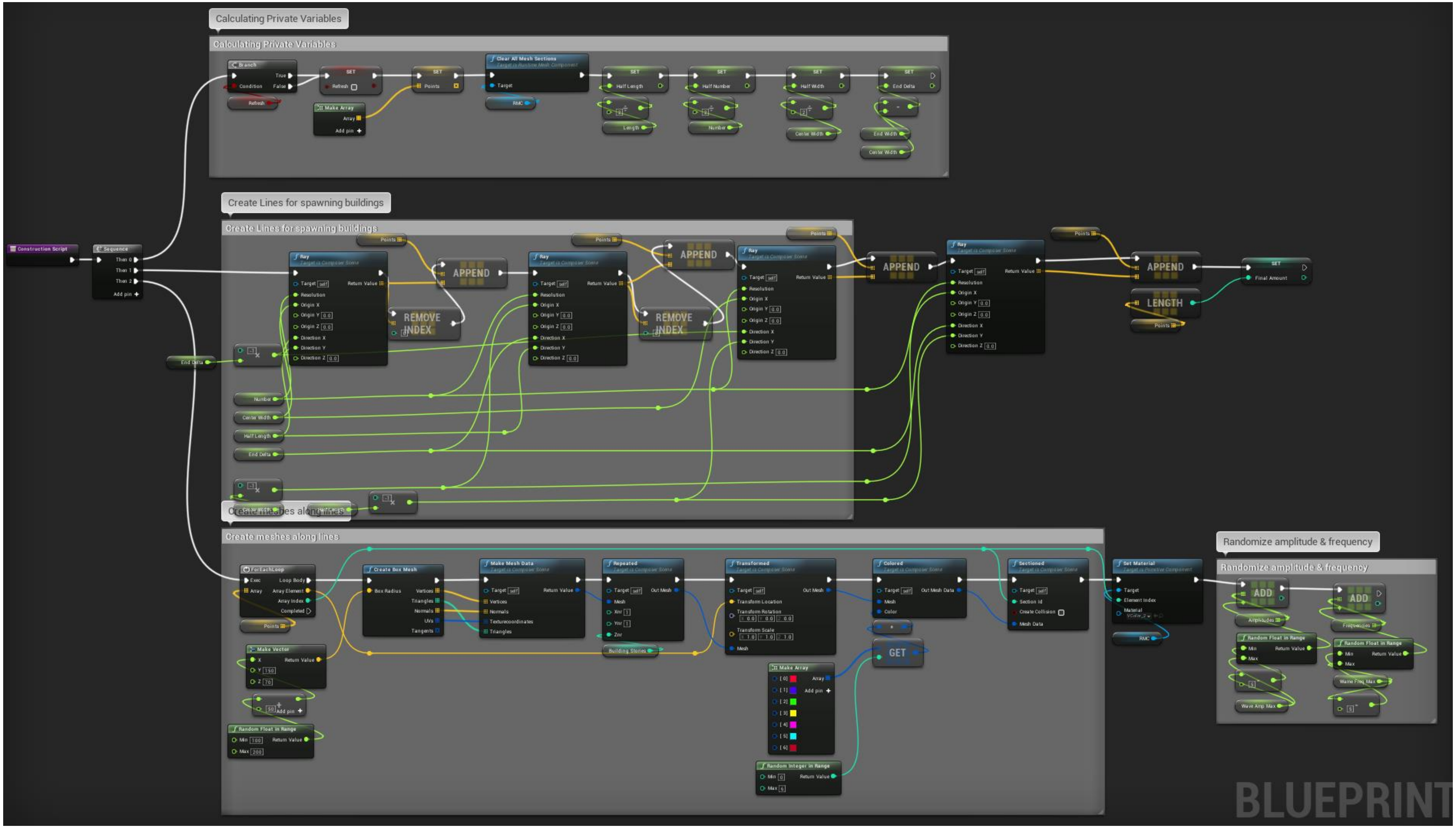
REPEATED SECTION	Repeates a mesh section a given number along x, y ,z axes
SECTION BOUNDING BOX	Calculates and returns bounding box of entire section
SECTION DATA	Returns mesdata for a section

Appendix C

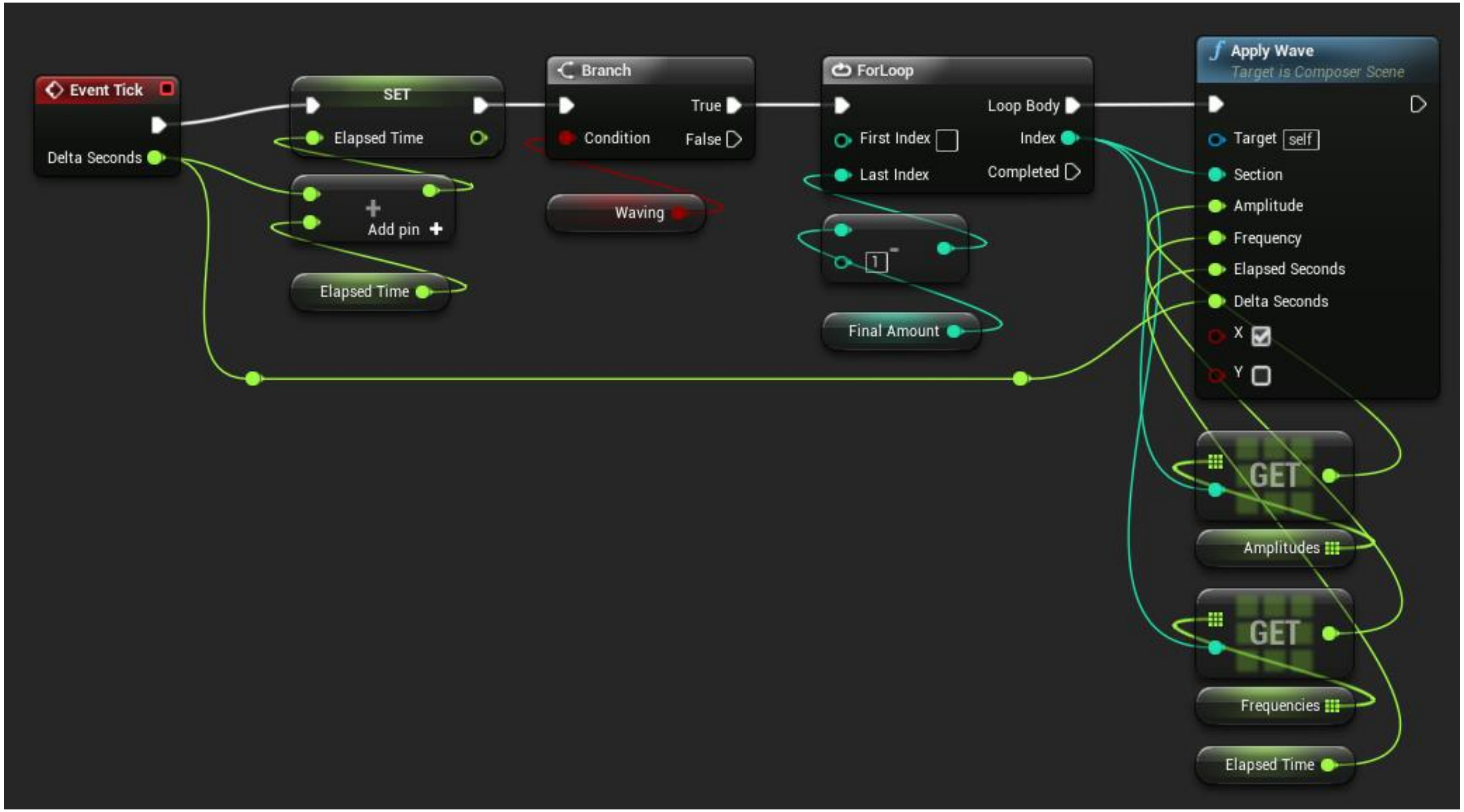


Selvportrett

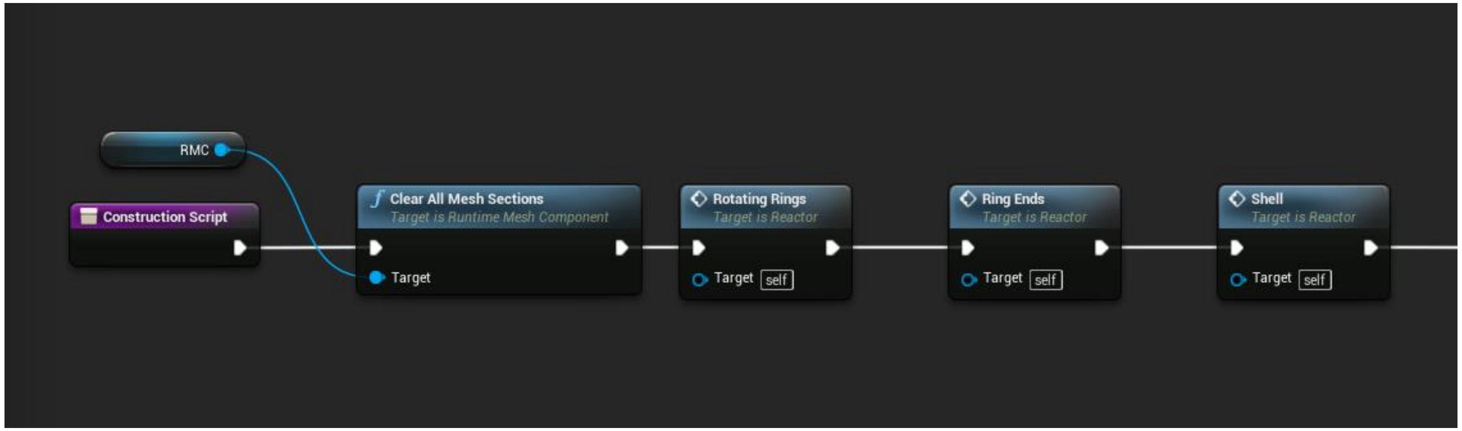
Manhattan

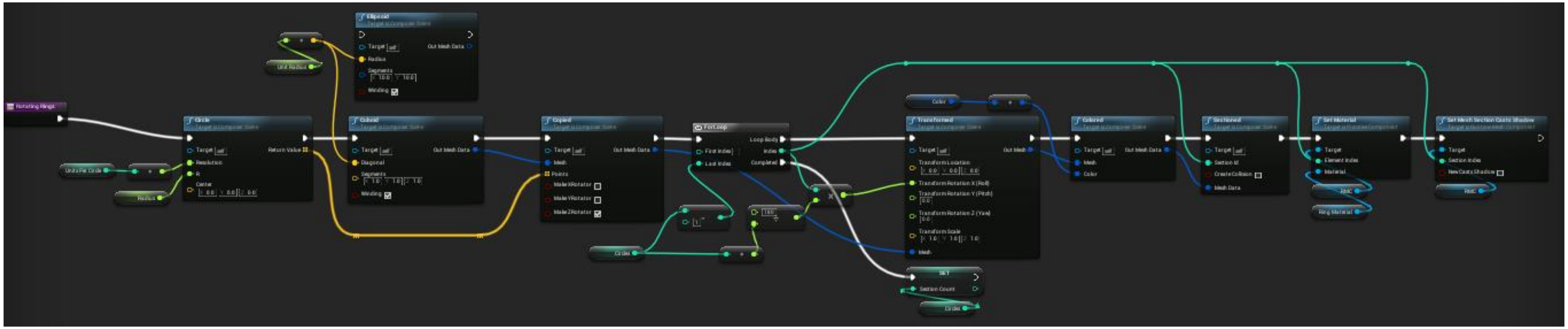

Calculating Private Variables
Create Lines for spawning buildings
Create meshes along lines
Randomize amplitude & frequency
Construction Script
Sequence
Then 0
Then 1
Then 2
Add pin
Branch
Condition
True
False
Refresh
SET
Make Array
Array
Points
Clear All Mesh Sections
Target
RMC
Half Length
Length
Half Number
Number
Half Width
Center Width
End Delta
End Width
Ray
Resolution
Origin X
Origin Y
Origin Z
Direction X
Direction Y
Direction Z
Return Value
APPEND
REMOVE INDEX
LENGTH
Final Amount
ForEachLoop
Exec
Loop Body
Array Element
Array Index
Completed
Make Vector
Random Float in Range
Min
Max
Create Box Mesh
Box Radius
Vertices
Triangles
Normals
UVs
Tangents
Make Mesh Data
Texturecoordinates
Repeated
Mesh
Out Mesh
Building Stories
Transformed
Transform Location
Transform Rotation
Transform Scale
Colored
Color
Out Mesh Data
GET
Random Integer in Range
Sectioned
Section Id
Create Collision
Mesh Data
Set Material
Element Index
Material
ADD
Amplitudes
Frequencies
Wave Amp Max
Wave Freq Max
BLUEPRINT

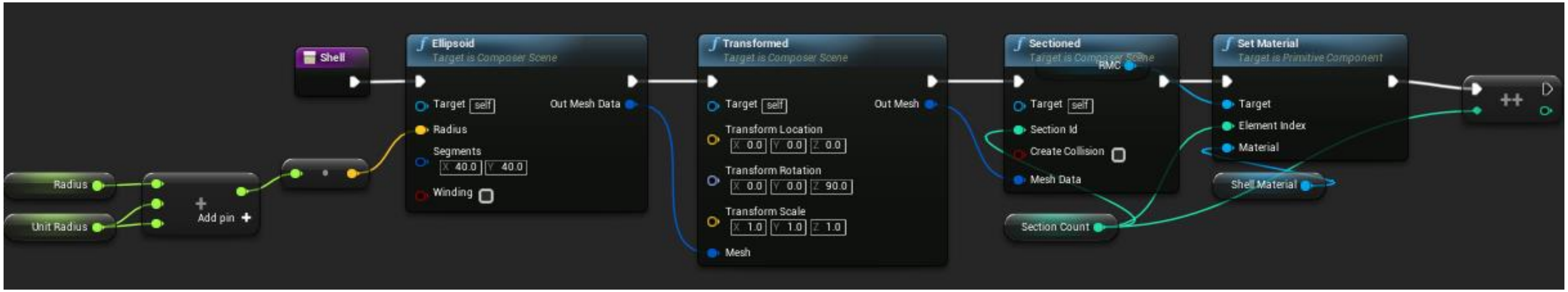
Manhattan Wave Function

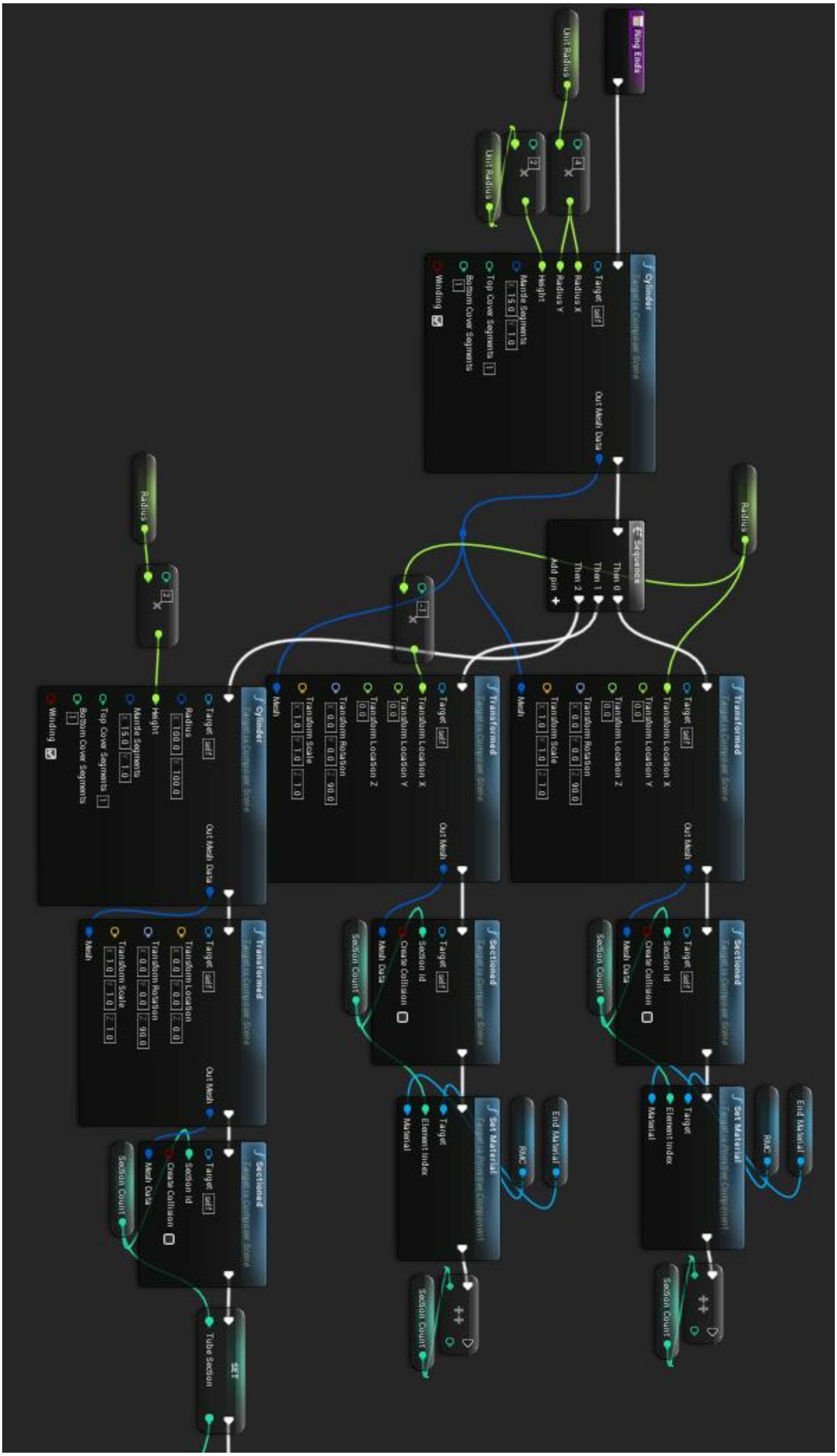


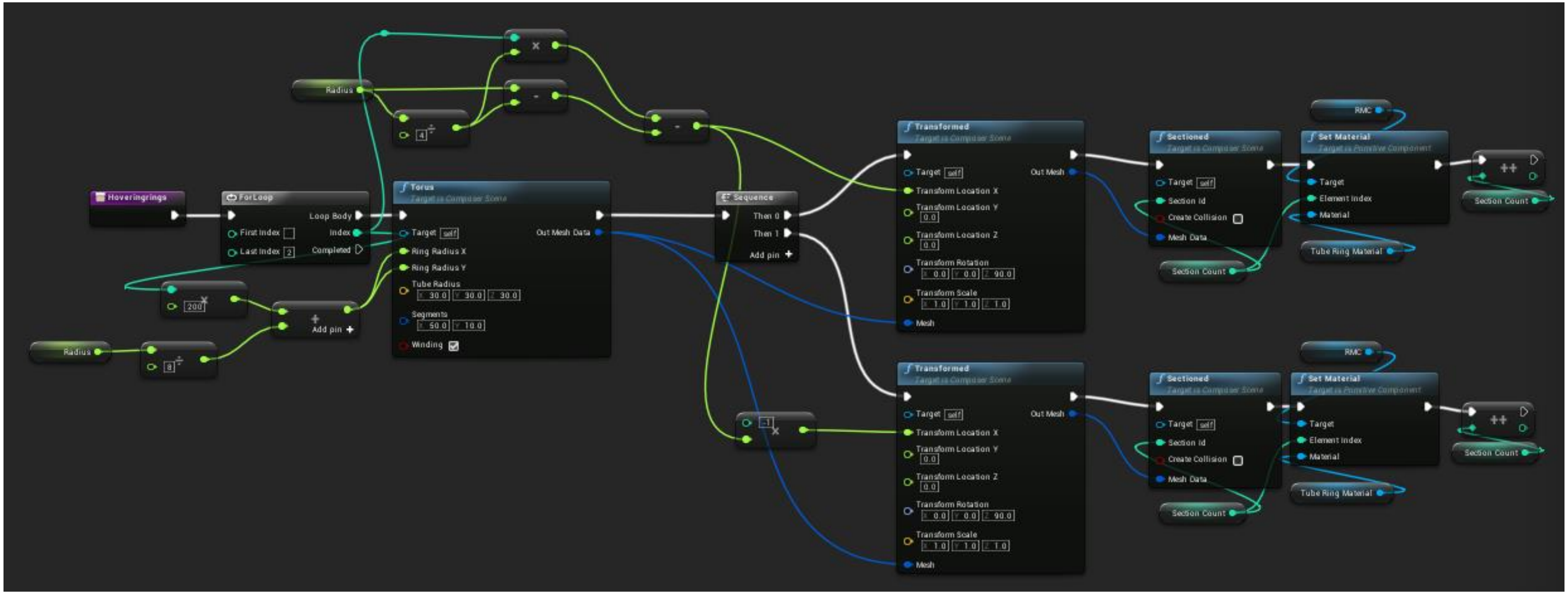
Reactor (Original Scene) + Subgraphs








Ring Ends
Unit Radius
Unit Radius
Cylinder
Target is Composite Scene
Target self
Radius X
Radius Y
Height
Mantle Segments
Top Cover Segments
Bottom Cover Segments
Winding
Out Mesh Data
Sequence
Then 0
Then 1
Then 2
Add pin +
Radius
Radius
Transformed
Target is Composite Scene
Transform Location X
Transform Location Y
Transform Location Z
Transform Rotation
Transform Scale
Mesh
Out Mesh
Transformed
Cylinder
Radius
Sectioned
Target is Composite Scene
Section Id
Create Collision
Mesh Data
Section Count
Set Material
Target is Rhino Component
Target
Element Index
Material
RMC
End Material
Transform Location
Tube Section
SET
++


Hoveringrings
ForLoop
First Index
Last Index 2
Loop Body
Index
Completed
Radius
4 ÷
×
-
-
200 ×
+
Add pin
Radius
8 ÷
Torus
Target is Composer Scene
Target self
Ring Radius X
Ring Radius Y
Tube Radius
X 30.0 Y 30.0 Z 30.0
Segments
X 50.0 Y 10.0
Winding ☑
Out Mesh Data
Sequence
Then 0
Then 1
Add pin
-1 ×
Transformed
Target is Composer Scene
Target self
Transform Location X
Transform Location Y
0.0
Transform Location Z
0.0
Transform Rotation
X 0.0 Y 0.0 Z 90.0
Transform Scale
X 1.0 Y 1.0 Z 1.0
Mesh
Out Mesh
Sectioned
Target is Composer Scene
Target self
Section Id
Create Collision □
Mesh Data
Section Count
Set Material
Target is Primitive Component
Target
Element Index
Material
RMC
Tube Ring Material
++
Section Count
Transformed
Target is Composer Scene
Target self
Transform Location X
Transform Location Y
0.0
Transform Location Z
0.0
Transform Rotation
X 0.0 Y 0.0 Z 90.0
Transform Scale
X 1.0 Y 1.0 Z 1.0
Mesh
Out Mesh
Sectioned
Target is Composer Scene
Target self
Section Id
Create Collision □
Mesh Data
Section Count
Set Material
Target is Primitive Component
Target
Element Index
Material
RMC
Tube Ring Material
++
Section Count

Appendix D