LUIS MONTAÑA AND CARLOS MOTTA

VISUALIZATION OF ARCHITECTURAL MODELS IN VIRTUAL REALITY

[August 4, 2016 at 18:06 – classicthesis version 1.1]

VISUALIZATION OF ARCHITECTURAL MODELS IN VIRTUAL REALITY

LUIS MONTAÑA AND CARLOS MOTTA

A study of viability

October 2015 – version 1.1

Don't find fault. Find a remedy.

— Henry Ford

Dedicated to our families and friends.

ABSTRACT

With the recent resurface of virtual reality multiple projects have been developed to make use of the capabilities given by head-mounted displays such as the Oculus Rift or the Project Morpheus, the previously mentioned projects range from videogames, to more academical, like the display of historical set ups using such technologies for instance. This project aims to make use of the Oculus Rift to display housing models. These models are an important part of a well-established practice in Colombia where a construction company builds a single apartment or house to show potential clients how the construction will look when the project is finished. This practice however, takes a considerable amount of time and when the construction has finished the model is demolished, creating a lot of wastes. Using the Unreal 4 Engine and computer graphic algorithms the project intends analyze the viability of creating a tool to successfully display realistic virtual models of said construction projects on the Oculus Rift in order for the user to have an enjoyable experience and also give them the ability to visualize how the project will look like.

Keywords: Oculus Rift, Virtual Reality, 3D Models, Computer Graphics, Housing.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [16]

ACKNOWLEDGMENTS

We would like to say thank you to everyone that believed in our work and in us as persons and professionals. It is important to note that this project was a collaboration between Carlos Motta and Luis Montaña. The following paragraphs state our personal acknowledgements starting with Carlos Motta and followed by Luis Montaña.

I, Carlos Motta, personally believe that this work could not be finished without the unconditional support of our project director, Andres Adolfo Navarro, who accompanied us throughout the entire process and gave us valuable feedback for this project. Also, I would not be here writing this document if it were not for my parents, who invested money, time and an unbelievable amount of faith in me, and also my partner Luis Montaña who worked as hard to make this possible. Without a doubt I would like to dedicate this to my father whom I love and respect. Lastly, from the bottom of my heart I would like to say Thank You to everyone who supported me.

I, Luis Eduardo Montaña thank from the bottom of my heart to our thesis director Andres Adolfo Navarro for supporting our project, believing in the project and for giving us his unconditional support. To my family of course, my parents and my sister for supporting my career choices and in general for being there for my my whole life. I am thankful to my friends Mauricio Cano, Jessica Dominguez, Ivan Daniel Mesias and Daniel Montenegro for being there for me and making my life way more fun. To all the people in the Computer Science lab and to the Computer Science Director Antal Buss for giving us advice and being so patient. Finally, my thanks to my friend and thesis partner Carlos Motta, who shared most of my passions and views during our time in college, for putting through with me and working all these months by my side to make this possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Virtual reality is a concept that can be traced back to the 60's and has had a relatively slow development. However, this has changed in recent years. For example, tech giants like Facebook and Sony are seeking to have a share on this new market with projects such as Oculus Rift[3] and Project Morpheus [2] respectively. Considering the growing interest in such projects, it is an appropriate time to develop and research technologies in this field.

With the constant urban growth of the city of Cali, Colombia. It is evident that projects involving construction are going to keep increasing as the demand for new housing and recreation building projects rises. The creation of housing models, a building that intends to show how the project will look on its final stage, is a practice that has been used for countless years as the main way to show a client what he will be buying. However, this practice is not efficient, not only consumes a considerable amount of time in the process of building the models and has to include furniture, but also when the project is done, the building is destroyed, creating a big amount of waste of material.

The recent breakthroughs in the area of virtual reality might allow to bring new and innovative solutions to issues in areas such as telepresence, gaming, education, etc. However, it is necessary to determine whether virtual reality does improve the previous established solutions.

This project aims at creating a prototype of a 3D house and studying the acceptance among potential buyers or renters (people with age 19 onwards), which in this case will be people with and without technical knowledge. Another important objective of the project is to evaluate certain aspects related to Virtual Reality such as graphical quality, simulation sickness and perceived realism by the users.

Lastly, this document will present the future work that could be born from the development of this project as its duration presents a challenge to how far the project can go, the document will also show the final results, an analysis of the results and the changes that these results could bring to the development of a potential tool.

1.1 PROBLEM DESCRIPTION

There are tools to visualize buildings on regular screens, such tools are limited by the size of the screen, with user interaction capabilities. However, those systems can not emulate the feeling of being in the actual house, and some of those tools do not even have enough computer graphics techniques applied to them as to make the visualization more similar to the real product. There are some other tools that show animations with superb graphics but they just have no interaction with the user.

This project intends to create a prototype that allows the visualization of houses using a standard measurement system, meaning that the models are going to have measurements similar to those of the real world. Additionally, computer graphic techniques will be used as well as basic interaction with the virtual environment. As it was explained in the introduction, virtual reality is a field that seems to have a very promising future since some industry giants have decided to make use of it for some of their future projects. Facebook hopes to use the Oculus Rift not only for making video games but also for telepresence on medical appointments for instance. On the other side, Sony intends to use its Project Morpheus on the Play Station 4 for gaming mainly.

1.1.1 *Formulation*

The problem lies in how to use virtual reality and the Oculus Rift alongside computer graphic techniques to visualize 3D housing models while allowing the user to interact with the environment?

1.2 OBJECTIVES

In the following sections the general and specific objectives are listed.

1.2.1 *General Objective*

The general objective of this project is to analyze the viability to create a prototype which incorporates computer graphic techniques, such as global illumination, together with virtual reality, in order to visualize 3D housing models while giving the user the ability to interact with the environment by using the Oculus Rift.

1.2.2 *Specific Objectives*

- Develop a prototype that allows the visualization of 3D models including houses and furniture on a virtual reality ambient which will be displayed using the Oculus Rift.
- Use computer graphic techniques that can fit the need of visualizing architectonic 3D models which include texturing, lighting and basic global illumination techniques.
- Integrate a mechanism of interaction which focuses on the use of the Oculus Rift.
- Validate the user experience and the viability of the system as a commercial tool.

2

THEORETICAL FRAMEWORK

In the following chapter the concepts that relate the most to the project will be explored. Apart from giving a brief explanation on each topic, some of these topics will also be accompanied by a mention of relevant projects of academic precedence.

2.1 3D TRANSFORMS

In computer graphics, all elements have a position in space, each element's position is represented by a set of coordinates: X, Y and Z. All elements can be transformed in the virtual space, for these purpose, there is a set of mathematical operations which allow to modify multiple conditions of the elements.

The most commonly used transformation in computer graphics is the translation, where a point with the form P = (X, Y, Z) transforms into a new point P' = (X', Y', Z') as pictured in Fig 1. This type of transformation is used to move an object from its origin to a different place in the three-dimensional space.

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

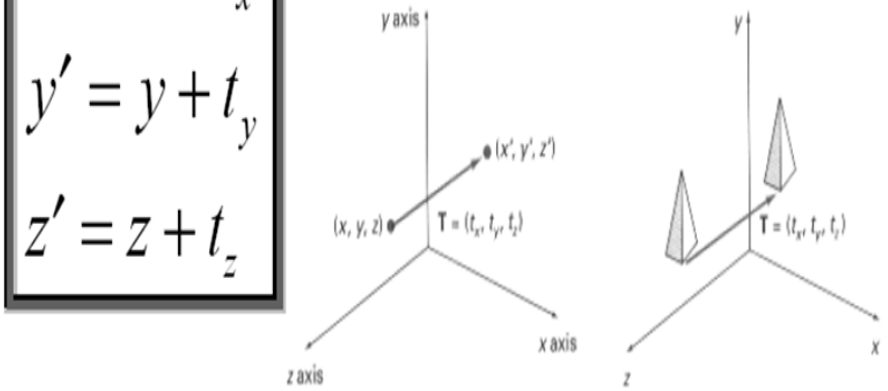


Figure 1: 3d Translation. [10]

Another type of transform is scaling, this operation is used to increase, decrease or distort the size of a given element, if the scale rate for each of the three coordinates is the same (Sx = Sy = Sz), then the size of the object increases or decreases uniformly. If the scale is less than one, the object has its size decreased, if the scale is more than

one, then the object's size increases. When the rate of scale on one or all of the coordinates is different, then the object is deformed and does not keep its original shape.

The last type of the most used transforms is the rotation, which is used to make the object face on multiple directions, the most common type of rotations are the ones that are made parallel to one of the three coordinate axis. Using rotations in more than one axis can make an object face in directions the parallel rotations can not, but these rotations are slightly harder to use.

The three types of parallel-to-an-axis rotations are usually differentiated by video game engines using the names of aircraft rotation axis to represent the three of them more easily, the names used for each rotation are shown in figure 2.

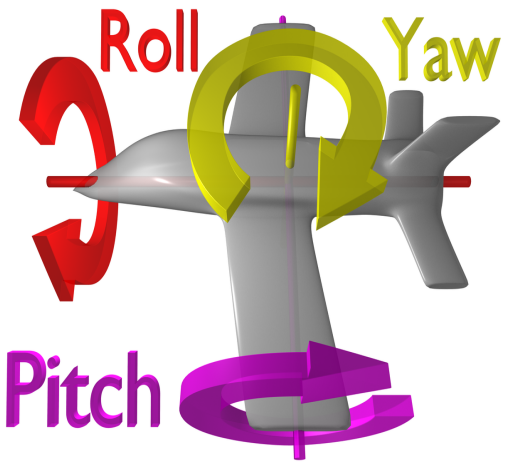


Figure 2: The 3 types of aircraft rotations. [19]

2.2 VIRTUAL REALITY

Virtual Reality has been the holy grail of computer graphics, the maximum expression of what a computer can do to the human brain, make the user believe he is physically on a virtual environment.

A common misconception about virtual reality is that it only involves things with visually simulating a place. Virtual reality intends to create sensory experiences, which include all the other senses besides sight. Aside from head-mounted displays, there are tool that give feedback on touch or even devices that replicate smells.

As expressed by Chalmers and Ferko in his study related to realism[8]. Virtual Reality is the maximum expression of realism, because it com-

bines believability and a physical representation of the world that it is supposed to be representing.

In computer graphics the most common way of developing projects related to virtual reality is by using a head-mounted display or HMD for short, the HMDs are used to limit the sight of the user so they can only observe what it is shown on the display, a set of headphones is almost always used as a complement, to completely immerse the user into the virtual ambient as sound and sight are the easiest senses to give feedback to.

Recent developments in the field of virtual reality have placed virtual reality as one of the most popular subjects on tech-related companies. Projects like Facebook's Oculus Rift [3], Sony's Project Morpheus [2] and HTC's Vive, have managed to considerably increase projects that incorporate virtual reality, some of these projects include games, telepresence, and visualizations, among others.

The three devices previously mentioned are pictured in Fig 3.

Figure 3: Virtual Reality Headsets [25]

2.3 REALISM

Realism is a quality that objects created to represent elements from the world posses (Some of these objects can be Paintings, Drawings, 3D models, among others). There have been multiple attempts to classify it, one of the most popular classifications in computer science is Ferwerda's [24], His approach proposes three types: Physical, Photorealism and Functional.

Physical realism as stated by [8] is the easiest to quantify. If there is an exact match in the spectral irradiance at all points in both the

real and virtual scene, then it can be said that physical realism was achieved. One of the biggest challenges when trying to achieve physical realism is that everything has to be modeled precisely, in addition, current monitors are not able to display the ranges of luminance that real life scenes have.

Photorealism is frequently used when studying computer imagery, to achieve Photorealism, there needs to be no discernible difference between a photograph of the real scene and a computer rendered image. Again Chalmers [8] notes that a photograph is still a representation of reality and depending on multiple variables it can or can not make justice of the real scene.

Finally, functional realism is one that provides enough information related to the actual scene to a user, so perform an appropriate task. An example used by Chalmers [8] is that of a aircraft builder, the material properties of the pieces don't really matter, but the relative size and shape does matter in order for the user to have the appropriate information required to build the aircraft.

2.4 PERCEPTUALLY-BASED RENDERING

Perceptually-Based rendering is a special form of rendering that aims to increase rendering efficiency while keeping an image fidelity, this is done by exploiting the limits of human vision. As shown by the studies performed in Elehelw's article [13] a couple of features are constantly checked in favor of others

Subconsciously humans check certain features such as light reflections and specular highlights, 3D surface details and depth visibility obtained a high amount of attention, with light reflections and specular highlights receiving the most attention out of all the features. On the other side, 2D texture details and edges and silhouettes which were considered pretty important received almost no significant attention.

This approach can help people decide which features should be the biggest focus on a development, or which characteristic should have the most resources allocated. The approach can also make more optimal projects, for example, if a feature that doesn't receive too much attention has less resources allocated, the reduction of allocated resources can increase the performance of a given project.

2.5 IMMERSION

Immersion is one of the most important aspects when it comes to virtual reality; considering that virtual reality seeks to make the subject

feel like he is in a virtual ambient, it is of utmost importance that the user feels like he or she is physically in the world. Elements such as sound, images and story are intended to help users immerse themselves in the virtual world created by programmers and designers alike.

One of the most important things to consider when talking about immersion is how not to break it. In their article *Behaviour, Realism and Immersion in Games* Cheng and Cairns [9] suggest that after certain level of immersion is achieved, breaking the immersion is a very difficult task.

Cheng and Cairns performed multiple big changes like lowering graphical fidelity and even changing physical laws to their ambient; their test subjects, all of which had at least 6 years of experience with computers, were filmed to determine changes on their demeanor when the big changes were applied, surprisingly no discernible impact was found. Even after performing the Rickert surveys, some subjects mentioned they hadn't noticed the changes until they were mentioned and all the subjects expressed they had an enjoyable experience.

Other studies like Brown's (which inspired Cheng and Cairns') suggested that inconsistencies in realism could break immersion, so it is still a debated field as most subjects in the computer graphics field.

2.6 ILLUMINATION

Global illumination refers to a variety of effects that allows shadow to be generated below and near the edge of the objects present in a scene and color bleeding for instance.

The role of these algorithms in the field of computer graphics have allowed new and diverse techniques to appear like real-time diffuse global illumination on voxelization [26] and real-time diffuse global illumination using radiance hints [21]. In order to visualize realistic scenes, these algorithms simulate light propagation and its interaction with multiple objects with various geometries in a three dimensional scene.

Global illumination also includes diverse processes that involve focal points of light, its reflection, redistribution in the scene, shadows and environment light absorption. These variables are the key in order to achieve realistic scenes.

Among the global illumination algorithms, there are a few that are commonly known and used; some of these algorithms are: Ray Tracing (Fig. 4) and Radiosity (Fig. 5). Ray tracing in particular is used

to simulate specular reflections of light on a surface by tracing a ray from the camera to the object while having an image of the scene between them, so that each ray can take into account the colors of each pixel that it passes through in the image in order to calculate the resulting color. This algorithm is also used in a wide variety of effects like reflection, scattering and dispersion for instance.

Radiosity on the other hand, calculates the propagation and reflection of light diffusely in a scene. This is done by storing the illumination grade of each surface, hence the illumination of an object not only considers light sources, but the reflection of other surfaces as well.

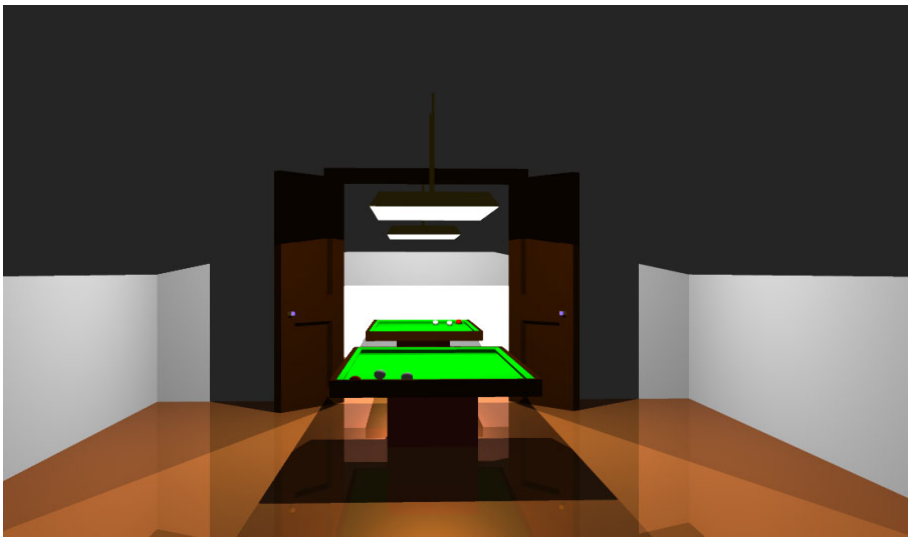


Figure 4: Ray tracing. [28]

There are many different algorithms used for global illumination. The following list shows some of the most known [18].

- Path tracing
- Bidirectional path tracing
- Metropolis light transport
- Photon mapping
- Irradiance caching
- Light tracing

Figure 5: Radiosity. [28]

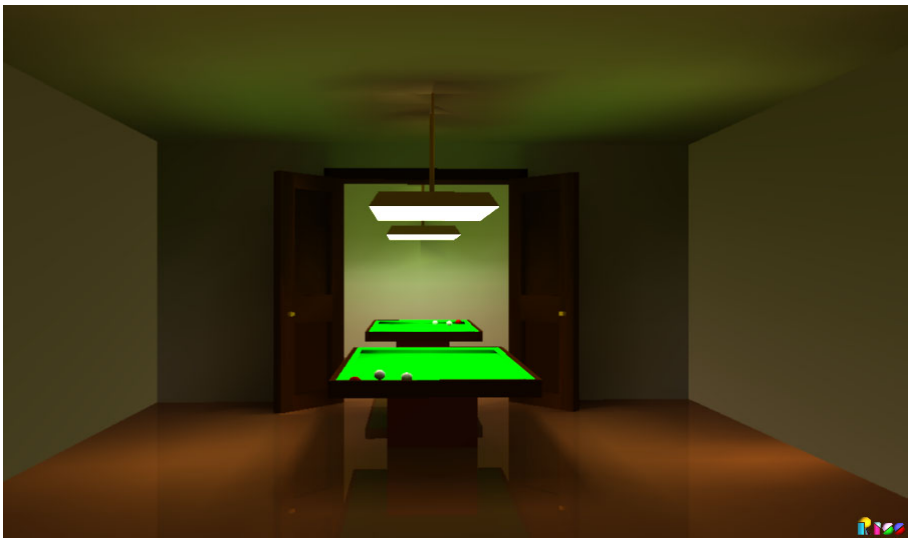


Figure 6: Ray tracing and Radiosity. [28]

Nevertheless, the cost of some of the above mentioned algorithms can be very high depending on a variety of factors such as the number of polygons in a scene. In order to reduce the computational cost of these algorithms, there are techniques that simulate light interaction between surfaces in a scene. One of these methods tries to achieve this by using bouncing lights, which places strategic points in a scene until the rendered scene resembles the one using the global illumination algorithms. However, this process requires time, tests, and error handling along the process.

2.7 3D MESH

A 3D Mesh is a set of vertexes edges and faces that are used to model objects in computer graphics. The most common way to represent this models is through the use of triangles (although some other convex polygons are used) [14]. As Figure 7 shows, when the number of polygons used to represent something increases its fidelity does too, this happens because it is possible to represent a higher number of details such as wrinkles, folds and other characteristics that are small but give a better resemblance to a model.

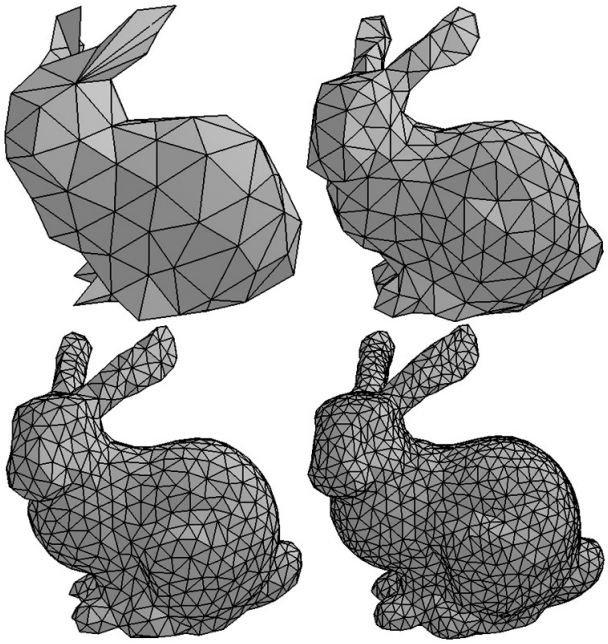


Figure 7: A rabbit model, using an increasing number of polygons. [22]

2.8 NORMAL MAPPING

Creating an extremely detailed model may look like the ideal way to increase the realism a model shows, however, this is a very expensive method, as each additional polygon increases the computing time of whatever is used to render the model. During the SIGGRAPH of 1996 [17] Krishnamurthy and Levoy envisioned a way to take information from a high poly model and transforming it on a displacement map that used Non-uniform rational B-splines to "place" over a low poly model.

This eventually evolved to the use normal maps, which were stored in a texture. The use of normal maps allowed the low poly model to be independent of the high level poly and is still highly used nowadays.

A normal map gets created by calculating the diffuse lighting of a surface, which uses the intensity of the light on it. If a bitmap with three channels is used, a more detailed vector can be obtained, each one of those channels represents a spatial dimension.

An example of a mesh, which had a normal map extracted from it and a plane with said normal map applied to it can be seen in Fig. 8

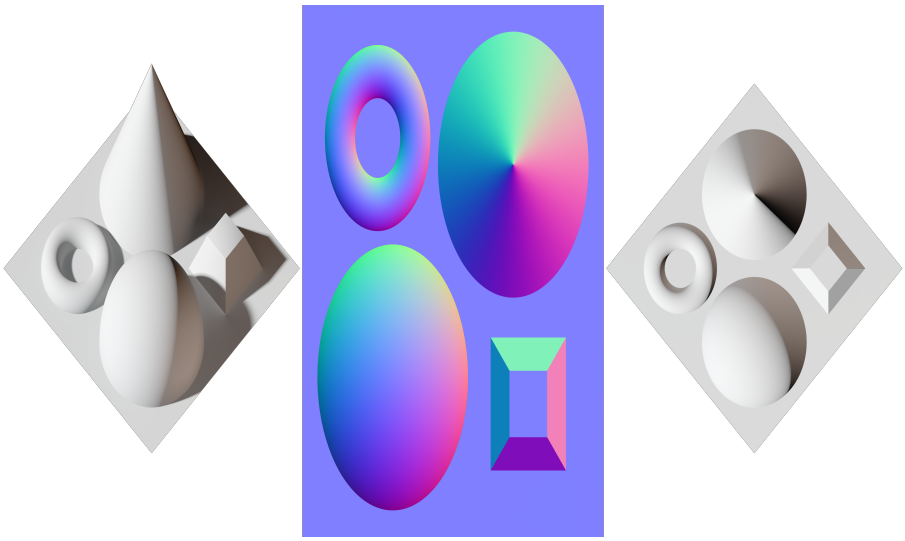


Figure 8: Original Mesh, Normal Map and plane with Normal Map [12]

As seen in Fig 8 the result is not perfect, but it works most of the times, and people really cannot notice the difference unless they get too close to the mesh to realize it is a plain texture.

3

PRELIMINARIES

In this chapter there will be an exploration on researches related to this project, the following section contains important aspects regarding the overall system such as differences and similarities between what it was designed and the systems proposed in each paper.

3.1 CONVENTIONAL MANUAL VIRTUAL REALITY RECONSTRUCTION, WALL EXTRUSION AND OBJECT MAPPING

The paper shown in [27] discusses the growing importance of virtual reality to visualize architectural designs. The authors present a semi-automated method of conventional reconstruction that goes from a 2D architectural model to a 3D model using common 3D authoring tools already available such as kinetix 3D studio Viz and Softimage 3D.

The main concern of this paper centers around the question of whether or not it is possible to convert current designs into 3D models fast and without much human input. Hence, three main steps arise from this, which are wall extrusion methods, object mapping techniques and conventional manual virtual reality reconstruction management.

The wall extrusion process takes 2D drawings of a wall structure and transforms them into a 3D model. This process can be accomplished by almost every modeling tool and it is very simple to do in that regard. However, extruding walls is a simple process as long as there is a clean representation of the 2D wall polyline structure that is readily understood by the authoring tool or modelling tool. Throughout the process of extruding walls, many problems can be found such as incorrect input methods, bad internal data organization and so on, and as a result, these problems might need human intervention in the future.

Object mapping is the process of placing pre-built 3D object models in the 3D environment as it is specified in the corresponding floor plain drawing associated to it. This process is usually done manually and it is generally a time-consuming task.

Finally, reconstruction of ceiling and floor is included in the management aspect of manual virtual reality reconstruction. It is a process that outlines perimeters given a floor grid, and by overlaying the 2D

floor plan both the floor and ceiling can be placed on the 3D world. Also, this process includes a manual component, hence additional textures and adjusting normals can lead help to improve the perception of realism of the model.

To conclude, the semi-automated reconstruction process presented in [27] tries to minimize the need of human interaction and maximize the output of the overall process, and even though the automation is around 10 to 15%, the fact that manual labour is reduced implies less errors and less tasks. Consequently, the main portion of this project requires human interaction and optimizing time-consuming tasks such as wall extrusion, therefore, automated methods to do this process could be useful for this project.

3.2 VIEW FRUSTUM CULLING AND PORTAL CULLING

The authors in [20] discusses an experiment where they search for a way to tackle the frame rate problem of visualizing 3D architectural models on mobile devices. This approach tries to optimize performance on a mobile device by applying view frustum culling and portal culling for interactively visualizing architectural models as smoothly as possible.

View frustum culling algorithms avoid processing polygons that are out of the view volume. This view volume defines an area where only the polygons inside it are rendered. Similarly, portal culling defines an area to be rendered as a cell, and each cell is connected by a portal. As stated in [20, p. 2], *The fundamental idea is that the viewer is inside a cell, and objects belonging to other cells can only be seen through portals; culling can therefore be applied to all objects falling outside the portal areas*.

The approach used in this article avoid rendering polygons that are out of the user's field of view, since rendering them would be unnecessary as it demands more processing power. In order to do this, the system proposed used a Mobi x3d player, which was intended to play X3D models on pocket PCs, but can be used on mobile devices as well. Additionally, it was found necessary to create a method to minimize the number of hidden polygons processed and to handle dynamic loading and flushing of data from the storage of the device in order for the 3D model to be explored without any disruption.

The main difficulty of this article comes from rendering many polygons and having to load a huge model with a very complex topology that exceeds mobile device processing power. Therefore, view-frustum culling, hierarchical view frustum culling and occlusion culling are algorithms that could be of use by having them greatly improv-

ing latency and frame rate stability when visualizing big architectural models in computers without much processing power.

3.3 CULTURAL HERITAGE AND VIRTUAL RECONSTRUCTIONS

This paper [23] discusses the interaction aspects of virtual reconstructions in museums, especially a cultural heritage project called **Etruscan** project, which seeks to recreate and restore the original Etruscan graves as well as its cultural context.

A virtual reconstruction is conceived as a digital ecosystem, meaning it should include a wide variety of things that are important to communicate to people considering the cultural context, such as visualization, storytelling, interactivity and inclusivity.

One of the main concerns of the paper is to tackle the barrier of interaction interfaces, which not only can be frustrating for the user to use, but they could make the user abandon the application shortly after using it. Therefore the paper focuses on some communication rules and criteria of importance to the authors to overcome this problem.

Exploring a virtual world is an important aspect of learning more about it, however, it is not the only aspect to take into account as it does not express anything about cultural heritages. The following citations highlight some of the considerations of the author.

A 3D reconstruction has to be considered as an "interface" whose final aim is to multiply the communicative potentialities of the cultural heritage, reactivating its relations in the space and time and its connections of meanings[23, p. 2]

Virtual reconstructions, in fact, can illuminate what is illegible, contextualize what is fragmented or isolated, and put back together cultural ties essential to the cultural identity of the object [23, p. 2]

And so, to achieve a more complete virtual reconstruction of cultural heritage in the **etruscan** project, there are a few characteristics considered mandatory for a digital ecosystem to be more immersive and a more storytelling experience; these are 3D reconstruction of a cultural context, inclusivity and interactivity.

Consequently, these characteristics are bound to the interface, the hardware used for the interaction with the user and the 3D world itself. The user needs to feel at ease when observing and exploring the world, the interface needs to be intuitive and the world must contain 3D objects as similar to the original ones as possible, which usually re-

quire digitalization of objects using techniques such as laser scanning, photogrammetry and computer vision, and manual modelling.

The main conclusion of this article centers around the important aspects of creating an immersive environment, where both virtual contents and communication aspects (where a multidisciplinary approach is imperative) are essential in order to allow the user to be a protagonist rather than a spectator as well as allowing the user to learn as he or she goes through the virtual environment. Hence, creating an immersive and inclusive experience is of relevance for this project due the fact that visualizing a model house needs to feel intuitive as the person the person is in the actual house with the furniture they would like to see in order for a person to even consider exploring the house a enjoyable experience.

3.4 MODELING LIGHT SCATTERING FOR VIRTUAL HERITAGE

The paper [15] discusses the importance of incorporating not only detailed realistic models, but also participating media, which simulates all the physical evidence that could be present in the real model such as dust, fog, smog, among others. The need to include an additional physical process on the textures and lighting present in a scene is a huge deciding factor to achieve a photo-realistic image, due to the fact that it adds more realism and alters the perception of the virtual representation that is being worked on. In the case of this paper, the **Kalabsha Temple** in Egypt is the chosen case of study, which was selected by the authors because of the potential participating media in scene (dust and sun).

Participating media considers the emission, absorption and scattering of light. This can be used to create effects such as dust, fog, smoke and flames. Participating media enhances the realistic appearance of rendering in contrast with traditional methods, which are rather plain and commonly used as good approximations for daily scenes.

The considerations presented by the authors were used to create a virtual reconstruction an ancient Egyptian temple using high fidelity documentation to create an almost identical temple. Additionally, participating media such as light and dust were added to test how they interacted in the scene. Light scattering is a great way add more realism and fidelity to a model by including particles to light like dust. Therefore, participating media could be used to aid with the lack of importance that current global illumination algorithms give to the medium in which the light travels, since these algorithms only considered light emitted by a source and its reflections for the most part, resulting in a the scene too clean and perfect.

3.5 LEVELS OF REALISM: FROM VIRTUAL REALITY TO REAL VIRTUALITY

Chalmer's paper [8] explores some efforts made on the field of virtual reality, and not only that, but it also identifies possible future researches regarding said efforts.

The paper exposes some rather interesting concepts, such as Types of Realism: Physical, Photorealism and Functional Realism. Each one of them has characteristics of importance to this research, each one of this types will be further explored on the theoretical framework section. The paper also explores believable realism, and the eight dimensions used in computer graphics, and expands on the idea that visualizations can not be pristine, as they tend to reduce people's perception of realism.

Furthermore, the paper explains ways of describing realism in a Cartesian graph, with the two axis being: a Believable axis and a Physically accurate axis. Virtual reality stands in the quadrant where realism is both believable and is physically accurate. One of the most important subjects explored in the paper is the comparison of real and virtual scenes. When comparing a photograph and the visualization of a virtual scene, the usage of perceptually-based metrics such as the Visible Difference Predictor (VPD) is rather common. Chalmers however, argues that a photograph is only a representation of reality and can not be used as a definite comparison between real life and a virtual reality.

In the article an approach where all five senses are taken into account is named as a Multi-modality approach, the objective is to have one of the five senses overloaded by the input, so small details perceived by the other four senses are ignored. Another approach explored, measures presence and it is noted however, that this metric alone is not enough to measure realism, so it is dependant on other metrics.

Previous knowledge, preconditioning and the nature of the task are also listed as important factors that are taken into account when comparing reality with virtual reality.

Finally, Chalmers explores three ideas that are of importance to multi-modal environments. There-reality, which checks if the immersive environment evokes the same responses from users as if they were physically in the real scene. The second idea he considers is related to the perception Equation, which is an equation that takes into account the five senses of a person to give a numerical value to an experience, so that they can be discretized. Lastly, Neurolinguistic programming is explained, this idea defines sensory inputs in terms

of five channels that represent the five senses and indicates that all experiences can be represented as a combination of these 5 channels, however Chalmers notes that this idea is a highly controversial topic.

3.6 BEHAVIOUR, REALISM AND IMMERSION IN GAMES

In this paper, Cheng [9] exposes the results of some experiments led by him and Cairns. During these experiments the main objective was to determine the existence of a threshold in order to analyze if people who crossed that line while gaming actually felt that their immersion was broken.

To test on this, they created a level where the qualities of behavioral and graphical realism would vary drastically over the course of the game. For example, at some point the gravity which was physically-driven, was completely changed making players jump great heights that they normally would not be able to. Another way the level changed considerably was by using low level textures, and so each level was made so that the user would play the game as it was designed first, meaning it had regular textures and the gravity and masses of the objects were not modified. Afterwards, the user would be faced with a room exactly like the one before but with all the changes mentioned.

The users were prepared for the experience through a set of photographs which showed how the game was played, but the photographs were just from the "normal" part of the course, thanks to this images the users predicted most of the game's way of playing.

Surprisingly, the users did not show any reactions to these changes breaking their immersion, some did not even notice the changes had indeed taken place.

3.7 A GAZE-BASED STUDY FOR INVESTIGATING THE PERCEPTION OF VISUAL REALISM

The aim of this paper was focused on determining features in which humans center their attention subconsciously while observing a set of images created by a computer. The article also describes things such as eye tracking and visual attention, both concepts were used to obtain the results of the researchers investigation.

The test was done by using 5 different images, 2 of them obtained by filtering the original texture image, another one was identical to the original texture image with added specular highlights and the last one was the real image, while making the participants of the test observe the images and answer a set of questions. A special device was used to track the eyes of the people. With the information obtained,

Elhew and his partners were able to determine which features were focusing on when assessing if the image was real or not.

Through the two tests performed (One showing a single image, the other one showing two images at a time), Elhew identified 5 visually salient image features:

- Light reflections/ Specular highlights.
- 3D surface details.
- Depth visibility.
- 2D texture detail.
- Edges/Silhouettes.

After these features were identified, Elhew and the rest of the research team observed and studied the dwell time of each individual on the mentioned features. The results showed clearly that light reflections and specular highlights was the most noticed feature followed closely by 3D surface details and depth visibility. 2D texture detail and the edges/silhouettes feature had a considerably big gap with the top three features.

The paper concluded that with the ability to identify specific image features that humans looked for subconsciously, video game developers, render artist and other professionals with 3D rendering needs could focus on allocating resources to specific characteristics or features of the rendering in order to increase the perceived sense of realism by the intended users of their respective products.

4 ANALYSIS

4.1 REQUIREMENTS

In this section the requirements that have been identified as vital to the development of the project are discussed, some of these arose since the conception of the project, some of them were brought up with the revisions and the decision to make the project more of a research rather than an application, see section 4.3.

4.1.1 *Functional requirements*

The following requirements are the functional requirements of this project, these requirements are descriptions of what the system was expected to do.

- The system should incorporate an already designed architectural model.
- The system must display 3D models, specifically architectural models, including houses, furniture and ornamental plants for instance.
- The system must incorporate computer graphic techniques such as: Global illumination, texturing, lighting, among others.
- The system must make use of virtual reality, in this case, the Oculus Rift.
- The system must allow users to calibrate the Oculus before showing the actual scene.
- the system must let the user have total freedom for him or her to explore the virtual environment.
- The system must display the capabilities of a video game engine by showing different elements that contribute to create a realistic environment.

4.1.2 *Non-functional requirements*

Non-functional requirements describe how the system should be, it describes what are the restrictions that limit the available options to achieve the final solution, and so it describes the overall quality of the

system. The Non-function requirements can be seen in the following list.

- The system shall be developed using a popular game engine that has Oculus Rift support.
- The system shall have a interface that does not block the view of the environmnet.
- The system shall have predefined PC requirements, as the Oculus Rift demands a decent computer to avoid visualization problems.
- The system shall be flexible in terms of overall maintainability.

4.2 USE CASE DIAGRAM

According to the requirements, Fig. 9 shows the use case diagram designed.

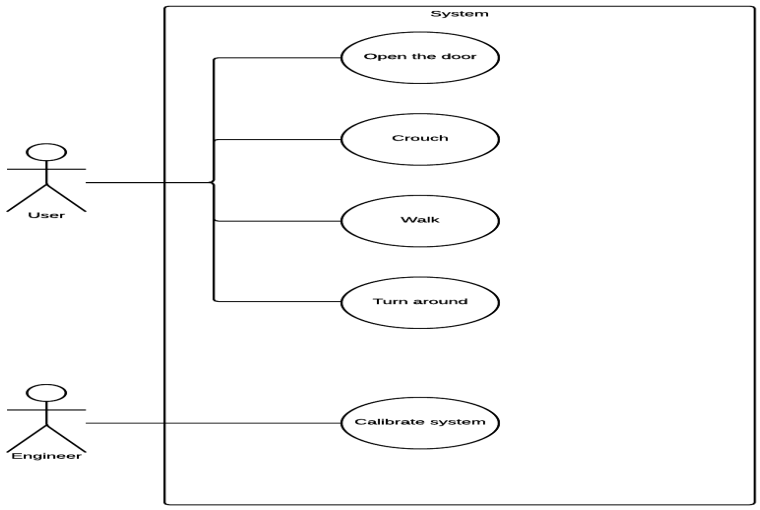


Figure 9: Use case diagram.

In diagram 9, there are two different actors, the user and the engineer or technician. The main role of the user is to explore the virtual environment, thus the only possible actions allowed for the user to do are, as shown in the diagram, minimal actions necessary to explore a virtual environment, such as walking, sprinting, etc. On the other hand, the engineer or technician is in charge of calibrating the system and guiding the user before he or she takes a step into the virtual world to ease the experience of virtual reality and if possible helping the user to avoid simulation sickness.

The idea of isolating any technical option away from the user comes from allowing the user to have a more immersive experience, while leaving other options such as movement and actions as simple as possible so that they do not become a burden for the user.

4.3 GENERAL ANALYSIS

The aforementioned requirements were mostly based on the objectives specified in section 1.1, however, the preliminaries mentioned in chapter 3 had a big influence over the construction of the model and guided the decisions made to implement the project, in regards to immersion and subjects related to immersion the article exposed on section 3.6 was the most influential among them, as the paper exposed that when a user reached a certain point of immersion, big details that would otherwise break immersion or give the user an unpleasant experience, were ignored or did not make enough impact to break the immersion acquired by the user, further discussions of this will be found in section 7.1.1.

Another equally important research was the one mentioned in 3.3. The semi-automated methods considered to reconstruct 3D models with minimal human interaction was one of the ideal scenarios for this project, due to the fact that modelling and design were not the main concern of this project, mainly because it is outside of the initial scope of the project. Although some issues regarding reconstruction and modelling methods were found, and they will be further explored in section 6.

In 3.2 frame rate problems in mobile devices are mentioned and solutions regarding frustum culling and portal culling are explained, however the technical limitations of a mobile device were not considered to be significant as this project is going to run in a desktop computer. Nevertheless, a frame rate problem is in fact a menace to be reckon with and finding ways to tackle this is very important.

Additionally, one the most influential aspects that were considered when exploring each possible requirement regarding 3D environments and virtual reconstruction and user experience are mentioned in section 3.3. In this article, the authors contributed greatly in listing specific points that affect user experience and the inclusive relation it has with interactivity. Inclusivity was also an enormous aspect to consider, due to the fact that it increases the sense of immersion and it helps to create a desired virtual environment. These considerations affected greatly in deciding how the interface was going to be created and it even helped to take into account considerations such as how much freedom a user can have and how the controller should be.

5 DESIGN

The previous chapter covered the system requirements, which established a baseline to start with the design process. The following section will cover the overall design of the 3D architectural model, the interface and the main components of the system.

5.1 THE INITIAL MODEL AND SOME CONSIDERATIONS

The initial architectural model created in SketchUp and provided by architect David Tobar as seen in Fig 10, had to undergo a process of transformation in order to take it to an optimal state, so that it could be ready for virtual reality. In order to do this, the model needed to have a better structure and organization of the present textures and faces, since it was going to be imported into a video game engine. Consequently, cleaning the model required a lot of manual work as it is a process that can not be completely automated, as stated in 3.3.

Figure 10: Architectural model by David Tobar.

Understanding what needed to be changed and what not was a big problem since models created in sketchUp had unique characteristics as seen in [6], such as double sided faces or triangulated faces, unorganized normals and little to none instances of the different objects present in a 3D model. All of this issues are discussed in section 6.1 and subsection 6.10.1. The architect that provided the model did not

consider these problems as he modelled it, hence fixing everything related to this needed to be done.

The model had to be imported to a 3D modelling tool to facilitate the organization of the structure of the model. In this case, 3DS Max was used as it is free for students and it has robust technical support and an easier way to handle UV mapping as well as compatibility with SketchUp.

Finally, the next step was to create a virtual environment as realistic as it could be created in a video game engine, where the improved version of the architectural model could be placed. The following sections focuses on the details regarding the design aspects considered throughout the process.

5.2 MODELLING

Prior to modelling, the most important aspect to consider was to have an actual model, which is a mathematical representation of an object that exist on our world (for further information on 3D models refer to section 2.8). The first model used for this project is a one-floor house that was built in Rozo, Valle del Cauca. This model was kindly provided for this project by David Tobar, a professional architect. Fig. 11 shows how the model looked at the start of the project. Note that the furniture inside the house was removed since they were assets created in SketchUp as fillers, in other words they were too simplistic for what is needed in this project.

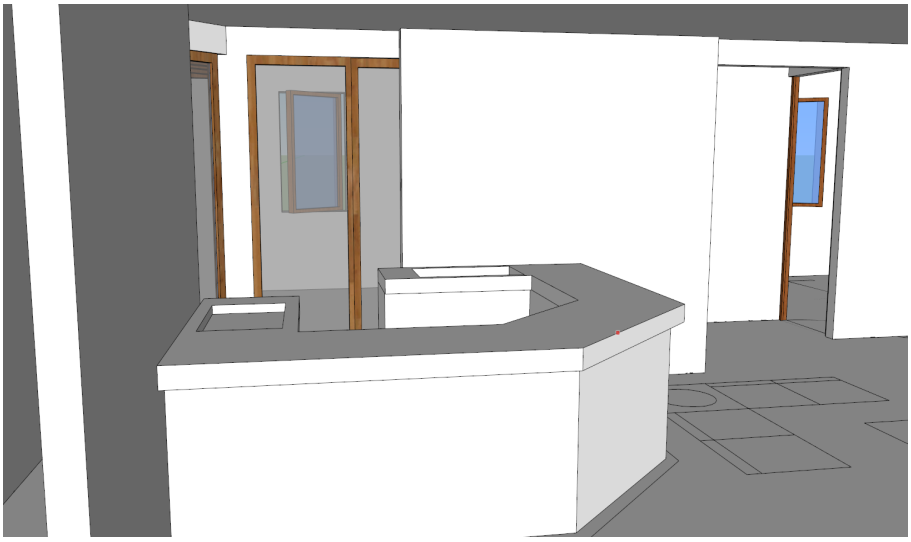


Figure 11: Model without furniture.

Modelling is a process that requires knowledge, expertise and experience, as it is a purely artistic process, and even though tools such

as SketchUp diminish the gap between the users and 3D modelling itself, creating realistic assets is in a completely different level. This requires sculpting, texturing and the use of shaders to create a simple 3D asset independently of what it might be. Therefore, the assets used in this project were taken from different sources that offered them at no cost and were considered to be decent enough to use them in this project.

5.3 INTERFACE

The system has no visible UI for users while exploring, since it obstructs vision and it could interfere with the requirement of allowing the user to immerse in the virtual environment. An interface also adds a bit more of complexity and it might cross the border of being a feature and become a nuisance to the user. Fig 12 shows how the virtual world is seen from the user's perspective and what was displayed on the computer's screen.

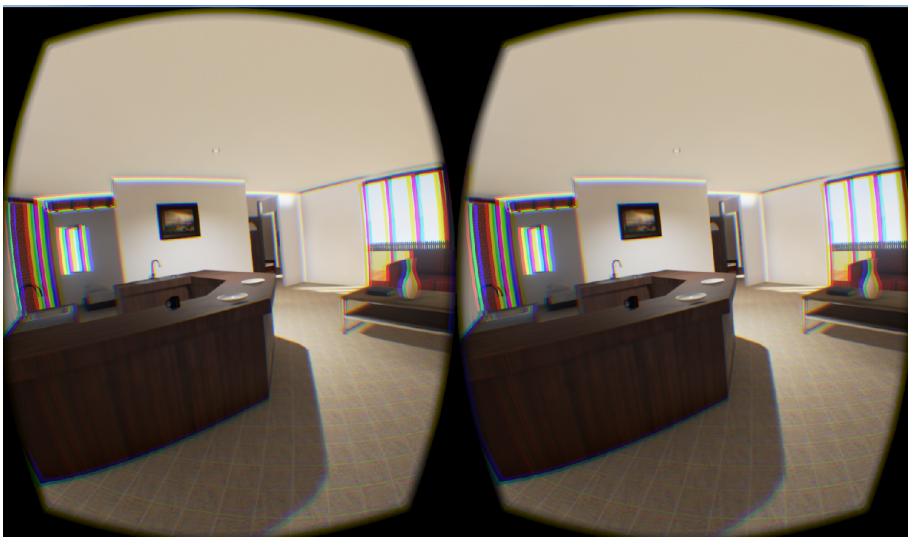


Figure 12: Screen vs Oculus Rift.

There will be an initial level where the user can familiarize with both the controls and the Oculus Rift, this level will be used for calibration, as the Oculus has one dial to each side which calibrates the distance at which the screen will be far away from the eyes, this can help users have a clearer image and avoid eye fatigue. This level can be seen in Fig. 13.

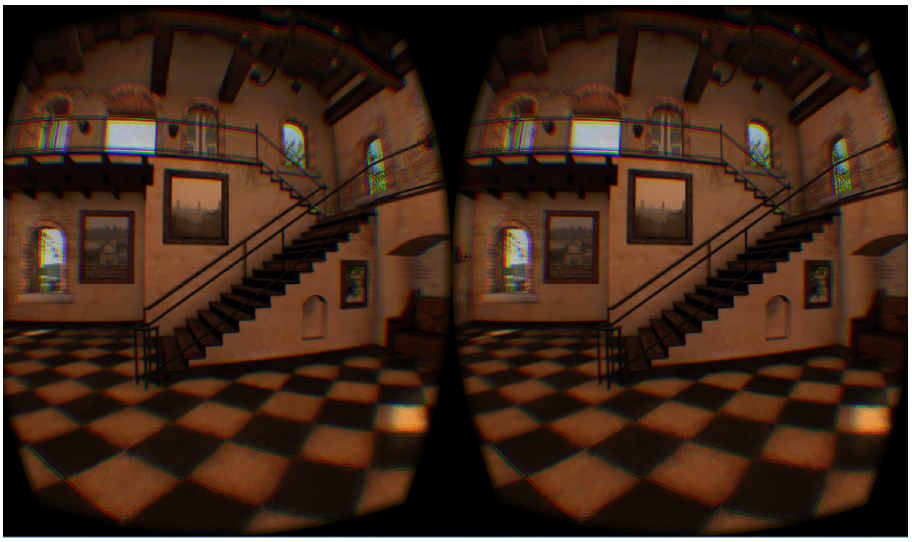


Figure 13: Calibration level.

5.4 APPLICATION COMPONENTS

The system consists of the Oculus Rift, a controller, and a computer that supports the application as shown in 4. Additionally, the system was designed to have no UI interface as mentioned in the section before. In addition to that, the application was designed to have intuitive controls with no more than two commands, movement and action commands, since aspects such as interactivity and lesser complexity are considered to be very valuable for this application.

Figure 14: Oculus Rift DK2 [29]

Figure 14 shows an image of the Oculus Rift, which is the main component of this application, as the entire project centers around the idea of visualizing realistic architectural models in virtual reality. The Oculus Rift is a virtual reality head-mounted display that is currently in a development state. It has limitations such as compatibility issues and frame rate problems. However, it fulfills the need of visualization in virtual reality and it can be easily acquired from the Oculus VR website.

Figure 15: Controller. [11]

Figure 15 shows the controller chosen. This component was originally optional, but it was decided to incorporate it to the application as the tests explained in section 7.1 showed that it was easier for people to move around using a stick and interacting using buttons rather than having them use the key board and mouse, like the standard used for first-person shooter games on PC.

5.5 APPLICATION DESIGN

The application follows the game loop of traditional video games, as it can be seen in Fig 16. The game loop consist of five basic states, initialization, load of assets, update, render and clean-up state. In order to keep the game running it is necessary to receive input from the user, update the game and render to keep these states in a loop as long as the application is running. This loop is managed by the game engine, therefore no additional functions were needed.

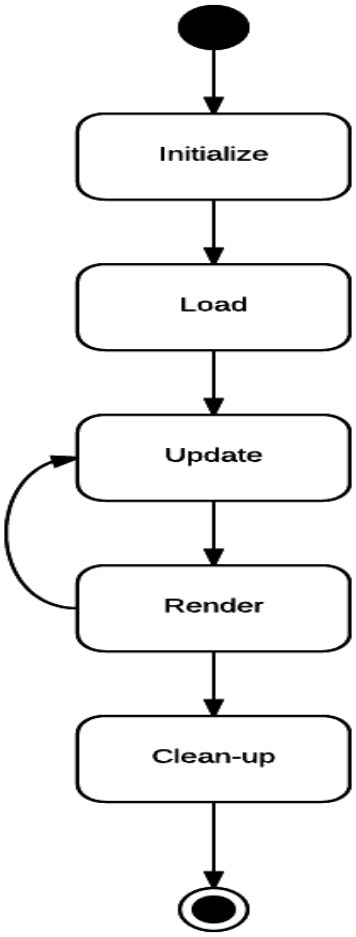


Figure 16: Generic Game Loop flowchart diagram.

While the game is running, the user can interact with the environment by either looking around the scene, moving or interacting with objects directly. Fig 17 shows a finite state machine where every possible action that a user can do is displayed. In this finite state machine, it can be seen how the user will be standing still as long as he or she does not move, so only user input can make it to move and no additional states regarding movement are considered. Basically, the user will be either moving, interacting or standing as he or she dives into the virtual environment.

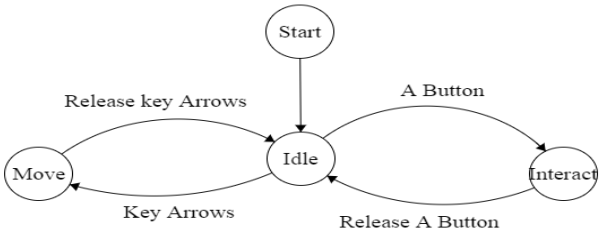


Figure 17: User actions represented in a Finite State Machine (FSM).

Additionally, every component of the finite state machine shown in Fig 17 has unique interactions with the components and classes of the Unreal Engine 4. They are of great importance considering the role they play at determining movement anomalies or frame rate problems caused by an overload in the input-update-render loop mention above for instance. Figures 18 and 19 show sequential diagrams regarding the most important states shown in Fig 17.

As it can be seen in the sequential diagrams, Unreal Engine 4 has its own hierarchy of objects unique to the game engine. This is explained in further detail in chapter 6. The user interacts with actors, which are the most common used class in the Unreal Engine 4. These actors contain components that defines its behaviour. For example, a Character actor will always have a component called character movement component, likewise, an actor will always have a root component, which can be any subclass of scene component that gives the actor a location, rotation and scale and it is also applied hierarchically to all components underneath it.

Figure 18 shows the sequential diagram of a simple movement action triggered by the user. The user sends a movement command by using movement keys, then the actor updates the coordinates by updating the root component with the new transformations and finally the character moves and the new translation is displayed on the details panel.

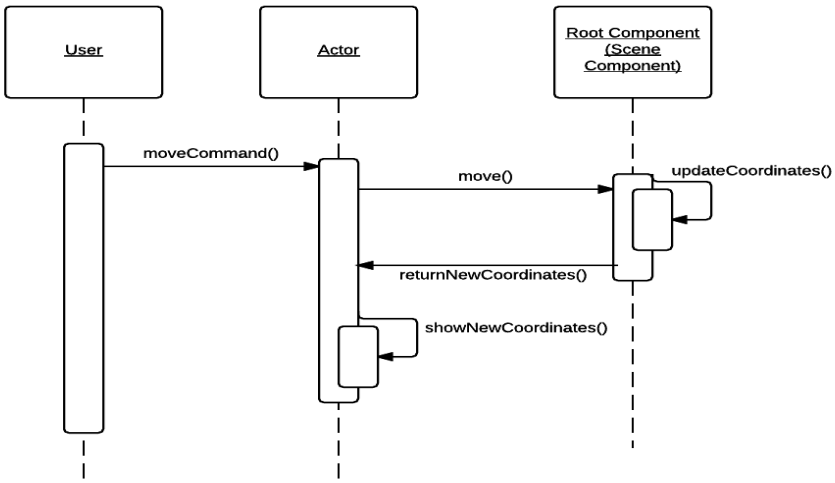


Figure 18: Simple movement action sequence diagram.

Fig 19 shows the sequential diagram of a simple interaction where the user turns a light on and off. This interaction is created by either blueprints or by scripting. The user uses the interact key, then the blueprints enables input for the actor and it allows the player to press a key and execute events that directly affects the actor in a particular way, in this case the actor would turn a light on and off.

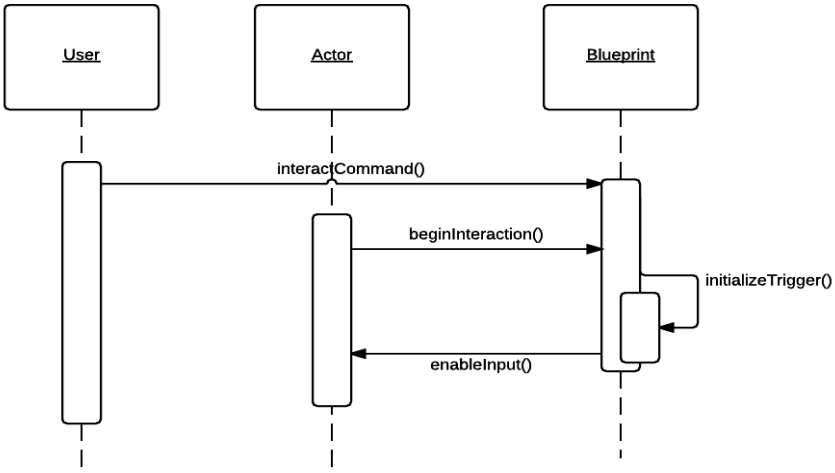


Figure 19: Simple interaction sequence diagram.

Finally, interaction with the system requires the user that is going to explore the virtual world to have a technical person to be in charge of adjusting the Oculus Rift to match his frustum and monitoring his actions in order to avoid situations where the user walks away and

trip over a real life object by accident. Even though the person is going to move using a controller, it is important to consider the possibility of this happening, as some people have a more immersive experience than others. Once this is set, the user is then put into a training level as it can be seen in Fig 13 so he can get used to moving using the Oculus rift. Lastly, the user is set free into the actual 3D environment.

5.6 STANDARD APPLICATION FLOW

While this project was faced with multiple technical issues that will be mentioned in the implementation chapter 6, an ideal (where none of the technical issues arise) application flow would be the following:

1. Obtaining a 3D model of a house.
2. Exporting the model to FBX forma so Unreal can process it correctly.
3. Customizing materials of the model in Unreal.
4. Place furniture and decoration in the model.
5. Placing reflection boxes and illumination accordingly.

Due to the technical constraints and the requirement for a first time setup of all the illumination features the flow went as follows:

1. Obtaining a 3D model of a house.
2. Fix geometry of the model in SketchUp.
3. Export model for UV mapping (DAE format).
4. UV map the model for customization in Unreal.
5. Exporting the model to FBX forma so Unreal can process it correctly.
6. Setup illumination configuration.
7. Place furniture and decoration in the model.
8. Placing reflection boxes and illumination accordingly.

6 IMPLEMENTATION

In this chapter the overall process regarding the development of the project is covered. This process takes into account every major detail of each step followed from the preparation of the model to the final version of the product.

6.1 PREPARING THE MODEL

The initial model created in SketchUp had various problems that had to be eliminated in order to proceed to the next step in the process of improving the model.

The first step consisted in fixing the normals of each face, as Figure 20 shows some faces were painted blue, this meant that the face had its normals pointing inwards, hence a face would only be displayed if looked from inside the walls and would not be shown in the actual rendering otherwise.

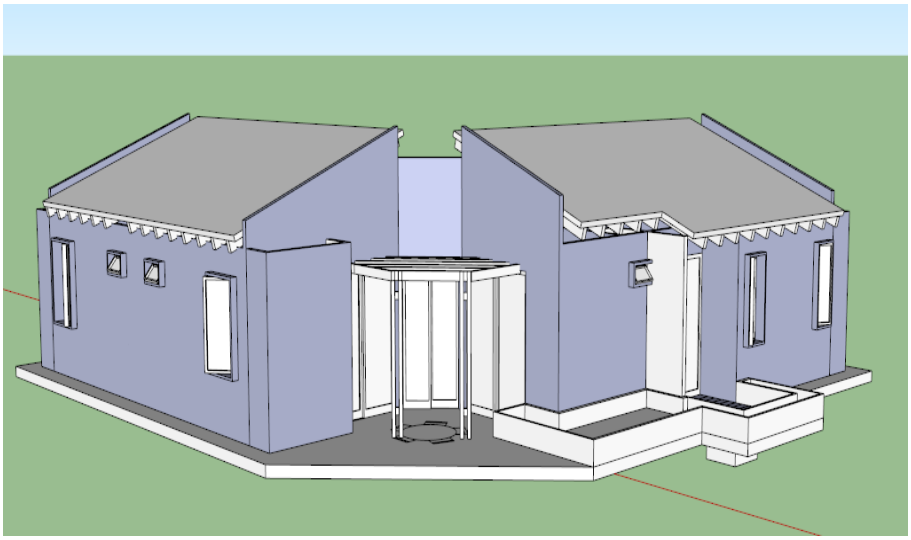


Figure 20: Model with flipped normals.

This is a common problem when modelling using Google SketchUp [6], and although the fix is as simple as right-clicking on the face and selecting the option to flip normals, it is a tedious process. Every face had to be checked in order to get the whole model displayed properly on 3DS Max and in the Unreal Engine 4 afterwards.

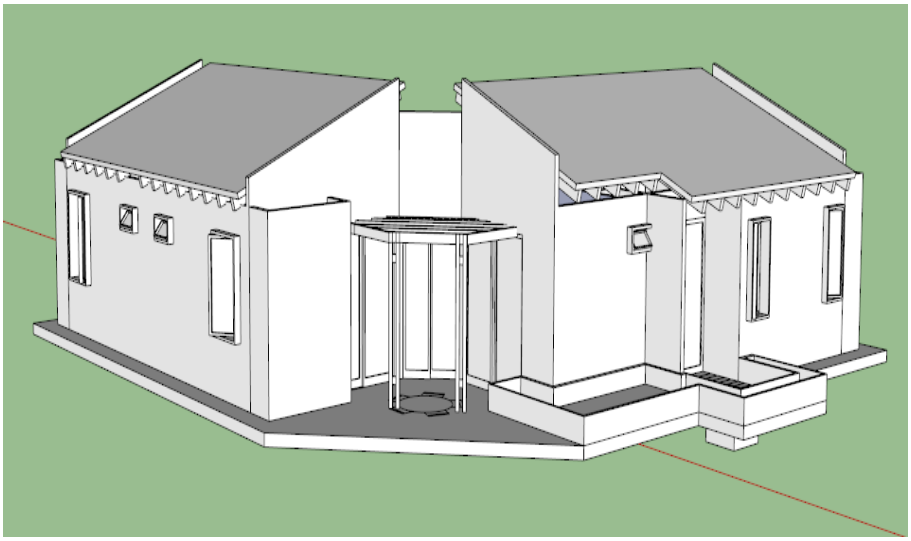


Figure 21: Model with fixed normals.

Figure 22: Textured model.

As seen in Figure 21, all the flipped faces were fixed. After this, the house was imported to 3DS max for the texturing and UV Mapping process described in section 6.2. Exporting the model from the 3D authoring tool was not particularly difficult, however, the export options were not flexible enough as to avoid problems such as triangulated faces and double sided faces. This caused some technical difficulties later on the process; this particular issue will be further explored in subsection 6.10.1.

After the process with SketchUp was done, the model was exported in Collada format (.dae for short) for it to be used on 3DS Max.

6.2 TEXTURING THE MODEL

After the model was imported to 3DS Max, one of the most important parts of the project started: the usage of realistic textures. Realistic textures are an important part of what the industry of computer graphics uses for realistic rendering, as using a high poly model wastes a considerable amount of resources. To solve this issue there are common methods to lessen the charge of using these detailed textures such as normal mapping.

Normal mapping was the technique used for the texturing process, a highly used technique to make low poly meshes look very similar to a high poly mesh with a lot of detail, to get further information regarding normal mapping read section 2.8.

There were multiple ways to obtain the normal maps. For example, one of them was to extract the normal map from a high poly mesh, however the house's model was too simple so this method would not be useful, as the textures would look out of place. Another option was to manually do the normal maps with Photoshop or other graphics editing software, this however would require a lot of work just on this problem, therefore a more flexible solution needed to be found.

The solution then, was to use a software that approximates normal maps using an image. The software used to obtain the materials that were missing was called CrazyBump [1], other software was used like PixPlant and a special plug-in for Photoshop. Nonetheless, the best looking Normal maps were obtained by using CrazyBump.

The following image displays the process of obtaining all the maps needed for the mesh.

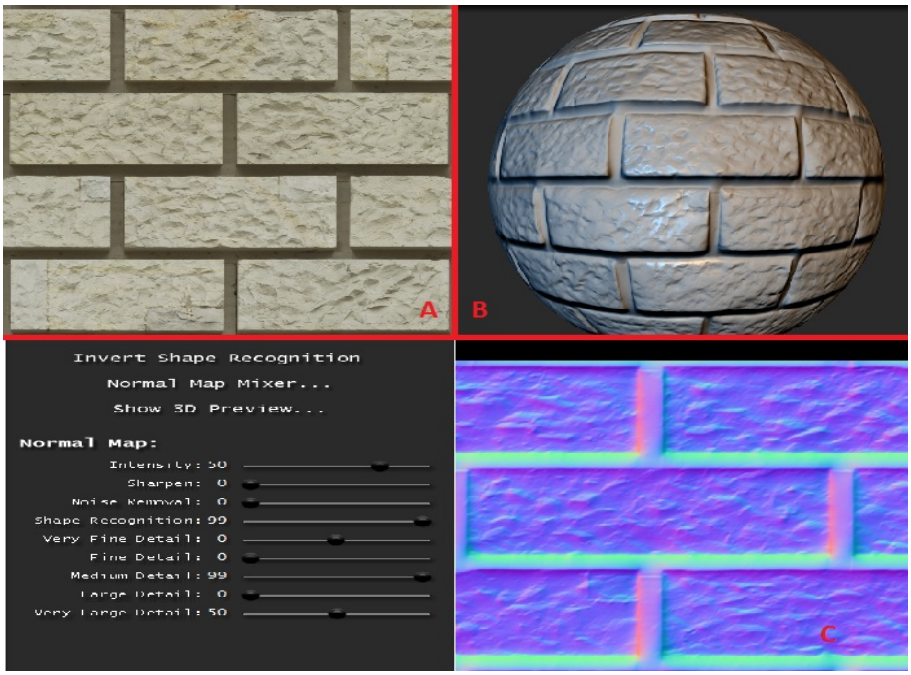


Figure 23: Normal map created by CrazyBump.

Fig 23a. Shows a regular texture of a brick wall, Fig 23b. shows how the material will look with the normal map obtained by Crazy Bump, figure 23c. Shows the normal map and the parameters that can be tweaked to obtain a more (or less) detailed normal map

It should also be noted that not all materials were just a Basic texture and a normal map, as some of the materials needed imperfections, dust or scratches in the final version. This was done in order to increase realism because some objects looked either too perfect or clean, this part of the process is detailed in subsection 6.7.2.

After obtaining the normal maps, each part of the model needed to be textured, it was decided to split the model in the following parts so they could be UV mapped and debugged more easily:

- Walls (split into 3 parts)
- Floor
- Doors
- Windows
- Ceiling

6.2.1 *UV mapping and unwrapping the model*

After splitting the model, the next step to follow was to apply UV mapping to the overall model, as it was absolutely necessary to ensure that each texture was placed correctly over the mesh. Not counting the walls, which had a seamless texture, every model had to have its UV maps customized in order for them to look real, when the texture was looking too big on the model, or the opposite, being too small for the model for instance.

3DS Max had a very good unwrapping tool, even though it was not very intuitive. After a long session of unwrapping, all the models were done, the materials had to be applied to each model or face and the UV maps were customized as stated above, the materials had to be saved on a newly-created library of materials on 3DS max, otherwise they would not be exported correctly; more information regarding the issues found during this step are available in section 6.10.

6.3 EXPORTING THE MODEL

The model was saved on 3DS Max's standard format (.max), this format however was not supported by Unreal Engine 4. The model had to be exported, the format that was recommended the most on different forums was the FBX format, which is highly used on video game development and is one of the formats owned by Autodesk, which also is the developer of 3DS Max.

3DS Max allowed a good amount of characteristics to be exported and had some customization as to what to export, some of those things included: lightning, cameras, double-faced models, among others.

After the model was obtained in the FBX format the final part of the process started: developing everything in a game engine.

6.4 GAME ENGINES

A game engine is a suite of tools designed to make game development more accessible to game developers. In other words, it allows the game to run on it, it exists to help game developers with a platform that does common tasks used in game development, such as rendering, physics, scripting, input and output and many other functions. Additionally, a game engine offer reusable components to help game developers bring games to life; using these components can simplify the complexity of the process by giving them an interface to work with for instance.

A game engine however not only brings the possibility of making games, it can also be used to do pre-visualization or even architectural visualization. This has become an option for many studios to expand into due to very powerful 3D tools and next generation level graphics that certainly make this a very viable option. Among the game engines already available in the market, there are some that are very popular, which are: Unity Game Engine, Unreal Engine 4, CryEngine, Hero Engine, Rage Engine, GameSalad, among others.

6.5 SELECTING THE GAME ENGINE

At the time the project started there was a wide variety of options for developers that needed a game engine. Among the popular engines there were two that were considered as viable options to be used on the project's development: the Unity Game Engine and Unreal Engine 4.

These engines were considered due to the needs of this project regarding architectural visualization. A game engine that could provide powerful next-gen level graphics, Oculus Rift support, intuitive tools to create realistic environments and finally, the ease of use of the engine was needed, as every task related to the project had to be done according to schedule and learning a new engine could delay this process. Therefore, Unity Game Engine and Unreal Engine 4 were considered to be the most adequate, and also they are among the most popular in the industry of video games.

At first Unity Game Engine appeared to be the most viable option, because it has Oculus Rift support, familiarity was not a problem, and it had very good graphical capabilities. However, an announcement made at the time from Epic Games regarding the new Unreal 4 Engine and its new marketing schema influenced our final decision. The announcement stated that everything the game engine had to offer was becoming free, with a 5% royalty fee if the game of application was published. To have the chance to work for free on an engine used by so many AAA studios was a major factor in deciding which game engine to use.

Nevertheless, the decision was not based solely on that, each engine was tested individually with pre-existing models and using the model provided by architect David Tobar. These tests consisted on using the native built-in functions that each engine has with the intention of grading its graphical capabilities in terms of realism. This was done by taking photographs of the models shown in Fig 24 and Fig 25, and having people compare them and categorizing them as realistic and not realistic. Most people believed the model imported in the Unreal

Engine 4 was superior. It is important to note that the model shown in Fig 23 did not have the same textures as the model shown in Fig 25, however, people did not care too much as they focused on the overall model more than details such as textures.

Figure 24: Rendering with basic illumination techniques in Unity.

Figure 25: Rendering with basic illumination techniques in Unreal Engine 4.

Based on these tests, documentation and general public commentary, it was decided to use Unreal Engine 4, because it had Oculus Rift support, a more complex material manager, as well as the ability to create 3D virtual environments with photo-realistic graphics using complex particle simulations and many other functions. It had advanced dynamic lightning and other power tools that facilitated the creation of more realistic environments. Another important fact

was that the creators of the engine, called Epic Games, had developed Unreal Engine 4 for a longer time than Unity, this could be noted as soon as you entered the working environment. Finally, Epic Games had extremely useful tutorials and documentation that were used extensively during the project's development.

In addition to this, the interface was simple, and some highly useful assets such as furniture and blueprints were found in projects provided by Epic Games for free that could be used in projects developed in Unreal Engine.

6.6 ARCHITECTURE

The architecture of the application proposed for this project utilized the hierarchy and structure of the Unreal Engine 4 classes, as well as blueprints; Programming in UE4 could be done by using C++ code and using blueprints, which was a visual programming method offered by Unreal Engine 4 as an alternative to using a text-based programming language like C++. Using the graphical programming facilitated the creation of simple gameplay.

Additionally, the system's architecture, as shown in Figure 26, was composed of a computer running the application created using Unreal Engine 4, a controller, the Oculus rift and the user who will be exploring the virtual environment.

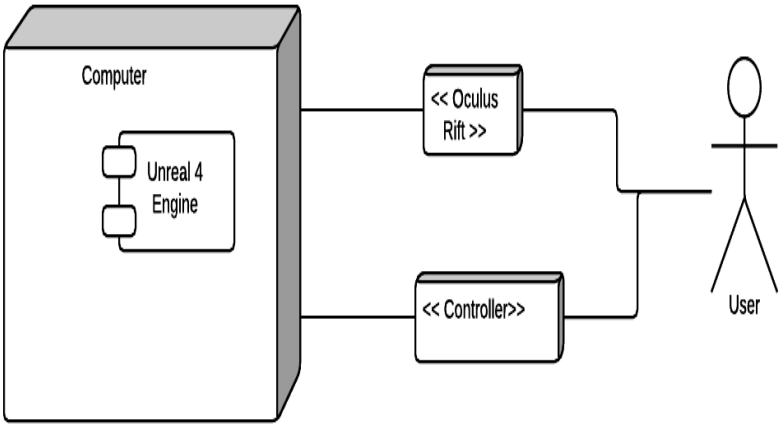


Figure 26: Architecture diagram.

The architecture used by Unreal Engine 4 to define classes can be seen in Fig. 27. Each class defined a template for a new actor or object, and each project needed to have at least one module, that si-

multaneously contained classes defined using the C++ language and additional macros provided by the engine.

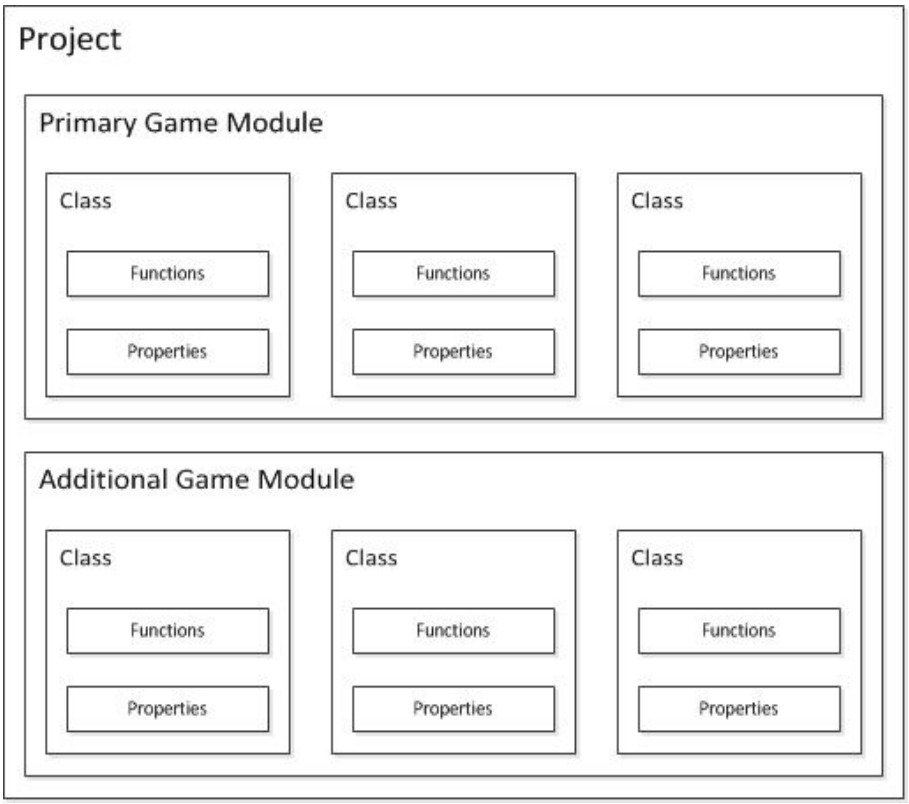


Figure 27: Unreal Classes Architecture.

Unreal Engine 4 had many classes for programmers to use, one of the most common class was the Actor class, which was considered the base class of every spawnable object. In other words, it was the class that allowed objects to participate in the scene as Actors. Actors and components were generally used to build up everything and as it is shown in figure 28, every Actor could have many components and each scene component, since they had a transform, could be nested to one another.

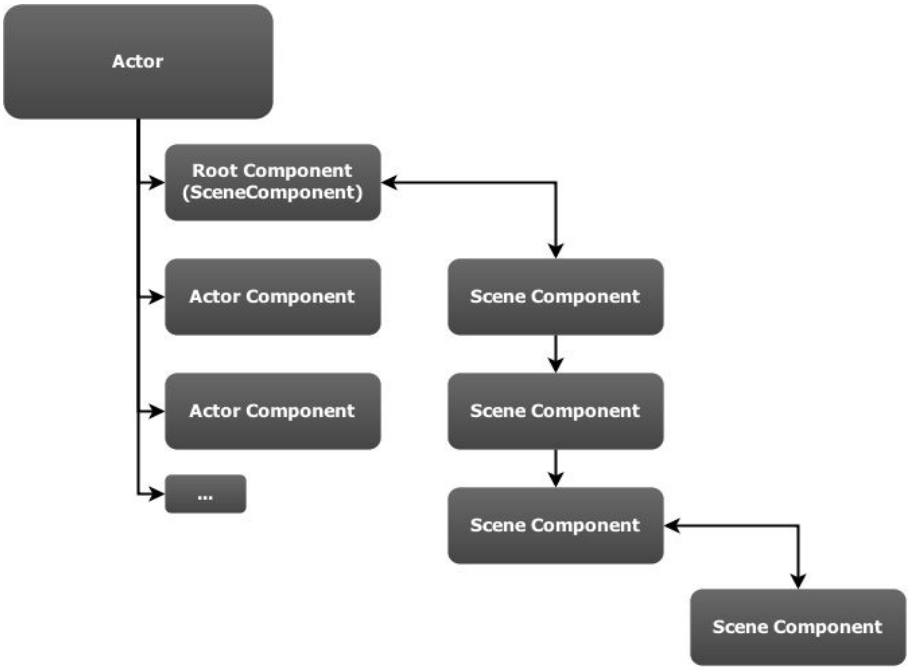


Figure 28: Actor and Component Hierarchy.

6.7 DEVELOPMENT ON UNREAL ENGINE 4

This section will describe the entire process of the creation of the scene, covering every step: from the importing of the models, tweaking the materials and using Unreal Engine 4 blueprint system, to the lighting systems and the issues faced throughout the development of this project.

6.7.1 *Importing the FBX models*

After the selection process was finished, the main features of the Unreal Engine were studied and the model was fixed using 3DS Max, it was then decided to start with the development on the engine. The first step was going through the importing of the models, which was pretty straight forward, it consisted on dragging the .FBX file obtained from 3DS Max into the content manager of Unreal Engine 4,then the engine would convert and organize the model. Figure 29 displays the default options available for the fbx importer on Unreal Engine 4, which were not modified too much, except on regards of auto-generating collision, which made a box surrounding the mesh; this option was not good for a lot of the meshes, since they were too complex and needed custom collision boxes.

Figure 29: Options for fbx models importing.

The transform options are related to the default positioning of the model, how far will it be displaced from the center, what kind of rotation it should have and if the model should be scaled to a different size of the original, this option could have been selected if we had not already done it on 3DS Max previously, therefore no additional options were necessary.

This process was repeated for each component of the model, albeit with some small fluctuation. After the importing phase was over, every model had to be placed correctly on the scene. Some of them had their scale modified even though they were imported in the exact same manner that the others were. Also some objects such as the windows, which had instances of it, had to be rotated and it was a process of trial and error until the entire scene was placed correctly in the newly created 3D environment.

6.7.2 *Materials in Unreal*

Following the set up of the scene, the materials of the models needed to be tweaked. 3DS Max has some very advanced material configurations. Unfortunately, only the most basic form of materials could be imported (the advanced materials would show up as blank materials and were only available for 3DS Max usage). These basic materials

are usually conformed by the texture, a bump map and a specular map in some cases.

Thankfully Unreal offered a more advanced way of handling materials, with enough knowledge a single material could wrap around a single object, such as the material for the couches. Figure 30 shows what a basic material looks like in Unreal Engine 4.

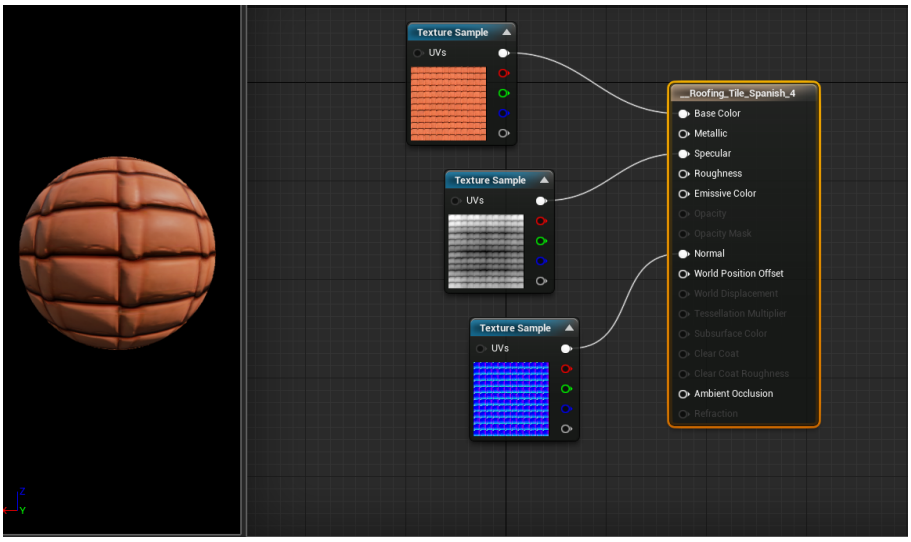


Figure 30: Basic material setup on Unreal Engine 4.

As figure 30 shows, the material characteristics are highlighted while others are shadowed. This happens due to the fact that the shadowed options are not commonly used, for the project's materials the most commonly used characteristics were the following:

- **Base Color:** This is either the base texture or the color that will be displayed for the material.
- **Metallic:** This option's value can go from 0 to 1. It is recommended to go for either 0, which means the material is non-metallic or 1, which means the material is indeed metallic.
- **Specular:** The specular option is used to define a surface's shininess and highlight color
- **Roughness:** This option specifies how rough a material is, if a material has a very low roughness it will have the capability to bounce light and create reflections. On the other side, if a material has high roughness it means its surface is uneven and does not bounce light in a way that can produce reflections.
- **Emmisive Color:** This characteristic specifies if the material glows. With certain algorithms it can be made so just a specific color

glows, making materials glow in specific parts and not the entire body of the object.

- **Normal:** This is one of the most used characteristic in materials. Using a 2D texture, a material can "simulate" a 3D surface. This is used to get details like dents or elevations in the surface without using extra polygons.

There are more characteristics, as it was mentioned, Unreal Engine 4 has a very advanced material setup, however the ones specified above are the ones used on the project, so the other ones were not relevant for this document.

Figure 31 shows a material we used for the house, specifically for the floor. This is a far more complex material than the one showed on figure 30. See the appendix section for a bigger version of figure 31.

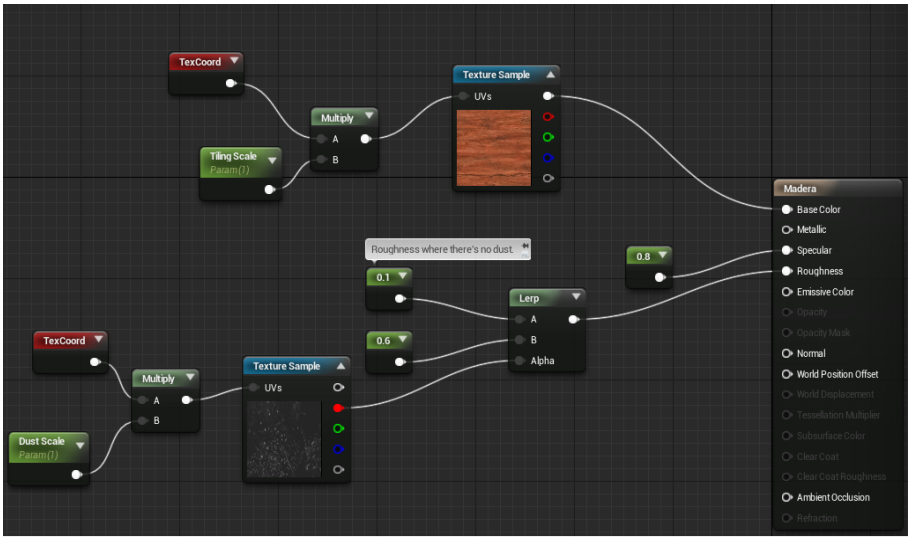


Figure 31: Basic material setup on Unreal Engine 4.

In the Unreal Engine 4, graphical programming goes from left to right, going from the first boxes to the left all the way to the final material to the right. The material shown in Fig 30 was parameterized so that it could be instantiated and used on different meshes. The two parameters are Dust size and Texture size, both are a scalar number, and both are used to make a texture bigger or smaller. This scalar is multiplied by the TextCoord variable, which is basically a 2D vector containing the UVs for each textures.

After getting scaled accordingly, the base color texture goes straight to its matching slot on the final material, but the dust texture still has some processing left to be done. For this particular material the dust

was just a spot where the material did not shine as much as it should, and so to solve this, a linear interpolation was used. In UE4 a linear interpolation has three inputs, an A value, a B value and an alpha value, which in this case was the texture.

The way the linear interpolation operation (lerp for short) works is the following: As values for the alpha channel approach a 0 value, the result of the operation is value A, if the value for the alpha channel approaches 1 then a value of B would be the result of the operation, and then the result of the lerp would go into the roughness slot.

To summarize, the spots where the texture was black would make the lerp output 0.1, which would mean the material had a low roughness where the dust material was black and would reflect light almost perfectly, on the contrary, the white spots in the texture would make the lerp result on a value of 0.6; this would make the material "rougher" and it would not reflect light as much. The final material can be observed in figure 32. Since the dust has a very small size the material was zoomed.

Figure 32: Wood Material with dust.

This particular material can be considered to have a fairly simple material script, objects that we obtained from other sources, like Epic Games' examples to show the capabilities of Unreal Engine 4 have some far more complex material scripts. Some of the techniques applied by those are: a single material that will wrap the whole object as a result of a well-thought UV map.

6.7.3 *Blueprints in Unreal*

Unreal Engine 4 features a system to script. This system is called Blueprints, which is based in the concepts of using nodes as a way of graphical scripting. Each node usually has inputs and outputs, which are usually easy to identify since a word describing the input is by the side of the dot that represents the input/output.

Blueprints are a very interesting concept. Graphically they look like the material creator shown back in figure 31, blueprints are functionally different though, due the fact that they use complex functions embed on a small node.

There are several types of blueprints in Unreal Engine 4, it was decided to use a couple of them for the project, however we considered a brief explanation of the most commonly parent classes used for blueprint was in order. The types of blueprints are:

- **Actor:** Object that can be placed or spawned on the world.
- **Pawn:** This is an actor that is controllable and can receive input from a controller.
- **Character:** This blueprint is a pawn with the special ability to walk around.
- **Player Controller:** this is an actor that has the responsibility of controlling a pawn used by the player.
- **Game Mode:** This defines the game that is being played, the rules to it and other facets of the game.
- **Actor Component:** This is a reusable component, it can be added to any existing actor.

To fulfil the particular needs of this project, it was decided to use the following blueprints: a game mode, a character blueprint and a mesh generation blueprint. These are briefly explained in the following subsections, in which it would be explain how the blueprint was created, and how that particular blueprint works on the project.

6.7.4 *Game Mode Blueprint*

As the brief introduction of blueprints explained in subsection 6.7.3, the game mode blueprint is used to specify the game rules. Figure 33 shows the blueprint used in the project, which displays the basic layout of the game mode blueprint.

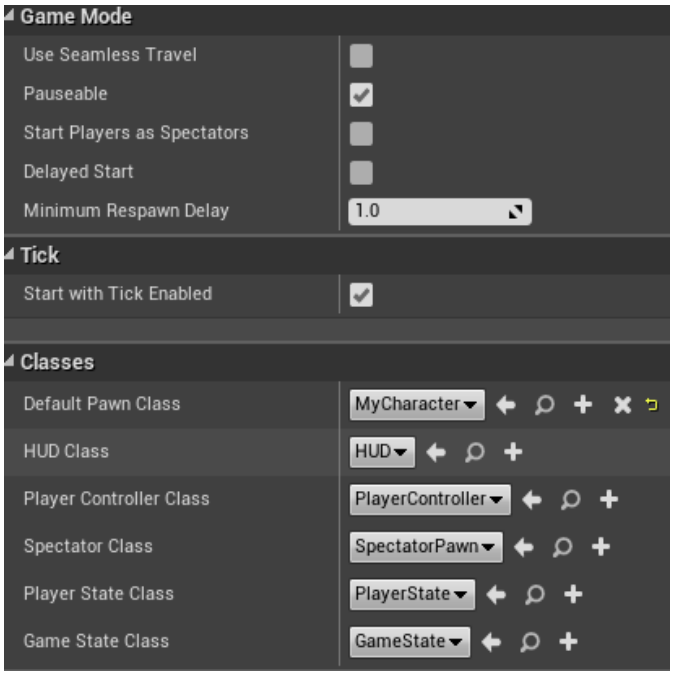


Figure 33: Game Mode Blueprint

It is important to note that this particular blueprint does not have a script screen as most of the other blueprints have, this is due to the fact that the blueprint is defined as a data only blueprint. However, a script might be added if the programmer so desires.

As Figure 33 shows, this blueprint is mostly composed of boolean statements that consider very basic characteristics that most games possess. For this particular project the seamless travel was not important as the project size is small compared to most video games levels. It was agreed to leave the pauseable option on just in case it could be used in the future or in case it could be observed that too many people got nauseous during the test. The other three options on the game mode part and the tick part were not modified for the project.

The most important part of the game mode blueprint would be the classes tab. Here it was defined which other blueprints would describe the way the project worked. The default pawn class for instance, is directed to the character blueprint created specifically for this project, which contains the basic movements a person would use when visiting a model house. A bigger description of this blueprint will be detailed on subsection 6.7.5.

The HUD (head-up display) was not necessary as a health bar or other elements on the graphical interface would distract users or cause them to not reach the initial threshold mentioned in Cheng's article [9], which is also discussed on section 3.6. The player controller was not used as everything needed for the pawns control was spec-

ified in the character blueprint and the input settings. The spectator and game state class were also ignored since they are used to specify specific aspects that the project does not have, such as spectators or specific game states like inventory state or quest-line progression.

6.7.5 *Character Blueprint*

As explained in the previous subsection, the main controller for the project's default pawn was established using this blueprint, as opposed to the game mode blueprint, this is not a data-based blueprint so it contains scripts. Nonetheless, specific parts of the script are briefly displayed on the subsequent paragraphs, and only the most important things are be shown; for further information on the character movement script, check the appendix section.

First of all, the character blueprint is divided in multiple "functions". These functions were enclosed in a small rectangle, which is the default way of showing a set of nodes work together. Figure 34 shows the script used for the character movement. The character movement is defined by two movements, moving forward and moving to the right, the opposite movements: moving backwards and moving left are obtained by using a negative value of moving forward and moving to the right.

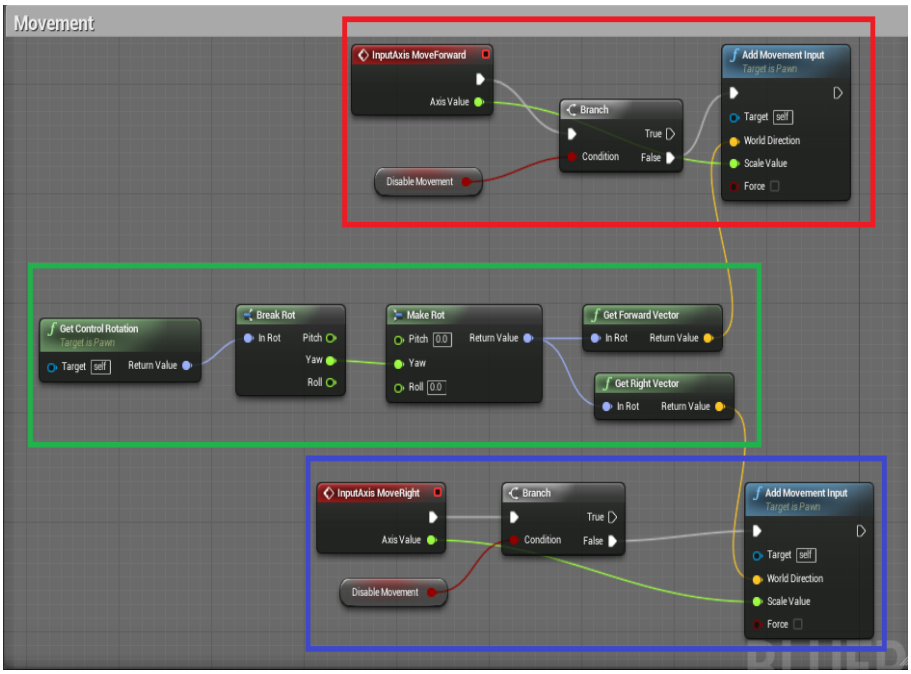


Figure 34: Movement Script

Basically, the script is split into three big parts, each of them is surrounded by a rectangle of a different color: Movement forward

(red), Movement to the right (blue) and obtaining the pawn's rotation (green). The two red nodes on the red and blue rectangles of figure 34 represent the user input that is intended to move the pawn forward and to the right respectively. After getting the user's input, the script uses a branch to check whether a condition is truth or not, in this case, if the variable DisableMovement is false, then it would not stop movement. This variable is commonly used to stop movement in cutscenes or animations, it was intended to use this functionality in case the door interaction or the light switch were finished as extra features to the project.

After the movement check was done, the final movement was passed to the pawn by the two blue nodes at the far right of the script, these nodes received the approval for movement from the branch, got the rotation direction from the green node sequence, which will be explained in the next paragraph and a scale value defined by the user input. This scale value was used to define how fast the pawn could move. Finally, the target of the movement was the pawn.

Finally, the green set of nodes of the script were used to obtain the pawns specific rotations for the world direction slot on the final movement nodes which would dictate the direction of the movements of the pawn. The initial node (Get Control Rotation) was used to obtain the full rotation set of the pawn. As it can be seen in the second node (Break Rot), this set of rotations could be broken apart in the three types of rotations explained in section 2.1, when a normal person walks, it only rotates in the yaw axis, as it is the only rotation needed to describe the pawns' movement. After the yaw was obtained, it was used on the third node (Make Rot), which created a rotator and passed it to two different nodes, one created a forward vector from it, the second node created a right vector, the output of those two nodes were send to the slots of world direction of the respective movement.

There are some extra components to the movement blueprints. One for the mouse input which is fairly straight forward since it just takes the input directly to the output node, so there is no considerable processing that needs explaining. The other component of the movement is the script that lets a user jog through the house by pressing a button, which consisted of the input which had both the pressed and released options. If the jog button was pressed, the character speed would change, otherwise, it would check if the pawn was crouched, which would reduce the speed, otherwise, the character moves around normally.

Both scripts can be seen in figures 35 and 36 respectively.

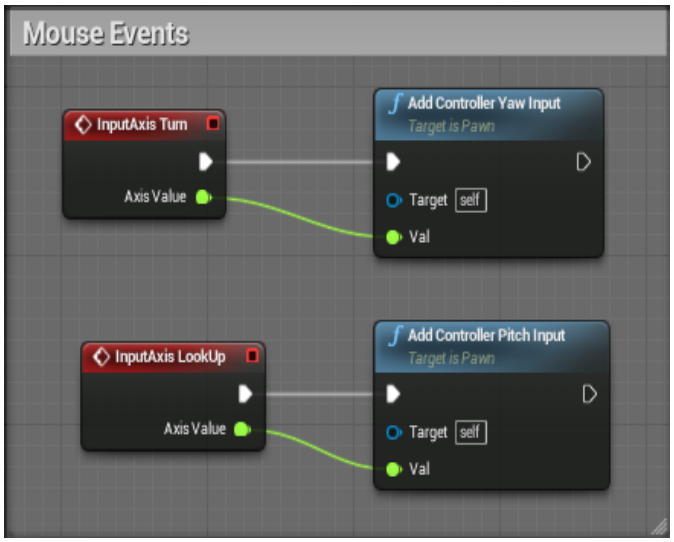


Figure 35: Mouse Handler script

Although the set up for the movement is fairly simple (it can be found on the Unreal Engine 4 documentation web page), the documentation is not entirely self-explanatory, so it was decided to introduce and explain how the script works.

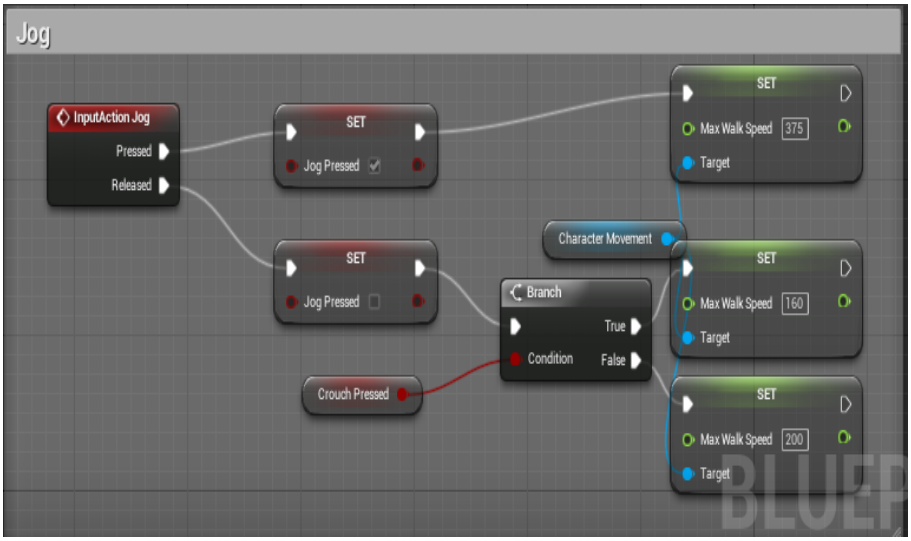


Figure 36: Movement Script

6.7.6 *Mesh-Generation Blueprint*

The project is placed over a plane of a considerably big size, therefore one of the issues faced during the development was how to limit the user's movement so they would not go out of the bounds of the terrain and fall helplessly to an infinite loop. The most obvious reason

was to place walls on each side of the house and make it impossible for people to walk over the edge.

This approach however, was not well accepted by the surveyed people, as shown in the results of the open questions in section 7.1.2. Since most people considered it too simplistic, or wondered what was behind of such high walls, it bothered them enough to write it on the survey. So it was decided to use a different approach: using a small mesh and replicating it multiple times to enclose the terrain, a problem arose from this approach however, which was regarding having to place many instances of the same object, and managing said instances would be troublesome, as there would be a need to select each one of them individually if any changes were to be made.

Unreal Engine 4's blueprint provided a very simple solution which addressed both problems effortlessly: there was no need to instantiate every single mesh to enclose the terrain and everything would be stored in a single place, the blueprint, which is easier to manage. This blueprint would create a row of meshes that were next to each other, hence 4 rows of fences were created.

Since the blueprint script is very extensive, it will be explained part by part; the first one, shown in figure 37 shows the first steps that were taken to generate a row of fences.

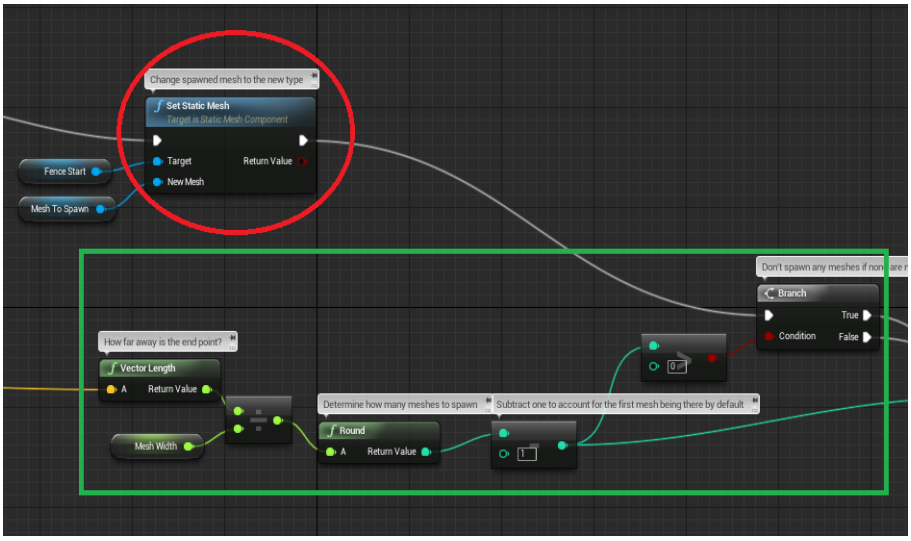


Figure 37: Mesh generator, Part1.

On the upper part of the script surrounded by a red circle, it can be seen how the blue node had three inputs, which were the the script itself, the mesh that was going to be spawned multiple times and the position of the starting mesh, then the node would send the mesh to a branch statement. The bottom half of the script surrounded by a

green square, started with the end position vector and then used a multiply function to produce a vector that ignored possible vertical displacements (both of these nodes are not picture in figure 37 as it would make it very difficult to read), the output of this operation was used on the first visible node, which returned the vector length, the vector length was then divided by the mesh width in order to calculate the number of meshes that needed to be spawned by using a round function after the division.

After the meshes were generated, the quantity of meshes created was subtracted by 1 to account for the mesh that was already placed. After this subtraction, the final amount of meshes needed were obtained, that number would be then sent to two places, one to a comparison that would then go to the branch statement mentioned in the first part of the last paragraph. This branch verified that the algorithm would not spawn extra meshes.

In figure 38 the process continuation can be observed, after the branch statement output was obtained, it is sent to the loop as seen in the part of the image enclosed by a red rectangle, which would have the value of one as starting point and the number of meshes minus one as end point.

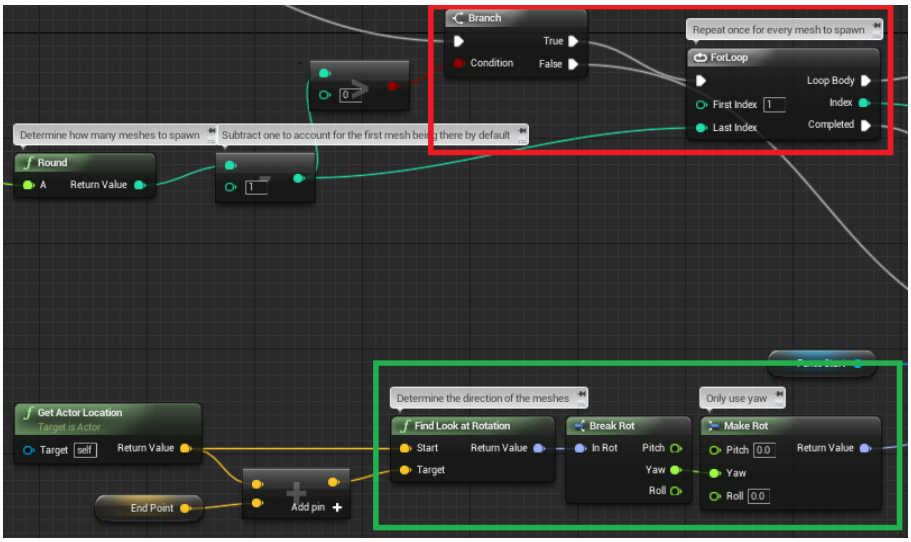


Figure 38: Mesh generator, Part 2.

On the bottom half of the second part of the script, using the same logic used on the character movement blueprint, the mesh rotator is obtained with a slight variation, the get actor location is used as in the movement blueprint, however, the end point is also used to determine the direction in which the meshes should be placed.

This was done by using an sum operation between the mesh starting point and the end point. Afterwards, the result was used as an input to part enclosed in the green rectangle, specifically to the node called "find look at rotation", which calculates the rotation that is needed for the mesh to be facing towards the end point. After the facing rotation was obtained, a process identical to the character movement blueprint was done, the yew was extracted from the rotation and then was passed along to the final phase of the script.

The final part, and the more confusing due to the multiple functions it has attached to it, is the loop phase and the positioning, this can be observed in figure 39. For the positioning, as it can be seen on the bottom side of the figure. A final node called "Set World Rotation" enclosed in a orange rectangle can be seen in figure 39, where the white line input is a flag that verified if the process was finished, the target input was the first mesh and the new rotation input was the rotation obtained in the bottom part of figure 38. This node positions the mesh correctly in the world.

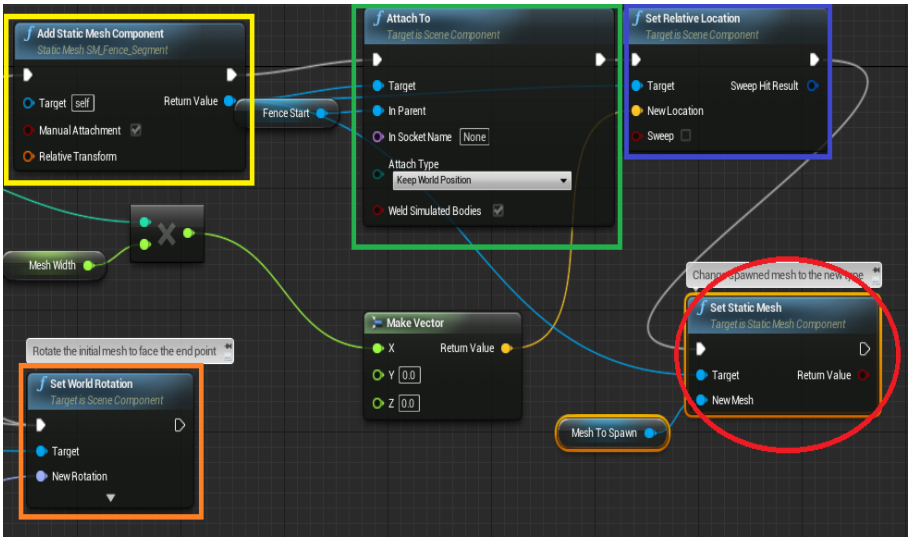


Figure 39: Mesh generator, Part 3.

On the top side of figure 39, the loop functionality is described. The first thing to be done was to use the node add static mesh component enclosed in the yellow rectangle, which created a new component given a name, in this case, the fence's name. After this process was done, the new element is attached to the original mesh (in the node enclosed with the green rectangle) with a children relation established. After the attachment was done, the new relative position for the new element was created (using the node surrounded by a blue rectangle), using the index of the loop and the size of each fence a proper new position was created with a displacement in the X axis. Finally the component is given a custom mesh (using the node cir-

cled in red), in the case of this blueprint, the fence mesh. The loop is executed until the ending condition is met and every single new instance is attached to the original mesh and they are lined with a relative position to the original one too.

The final result can be observed in figure 40.

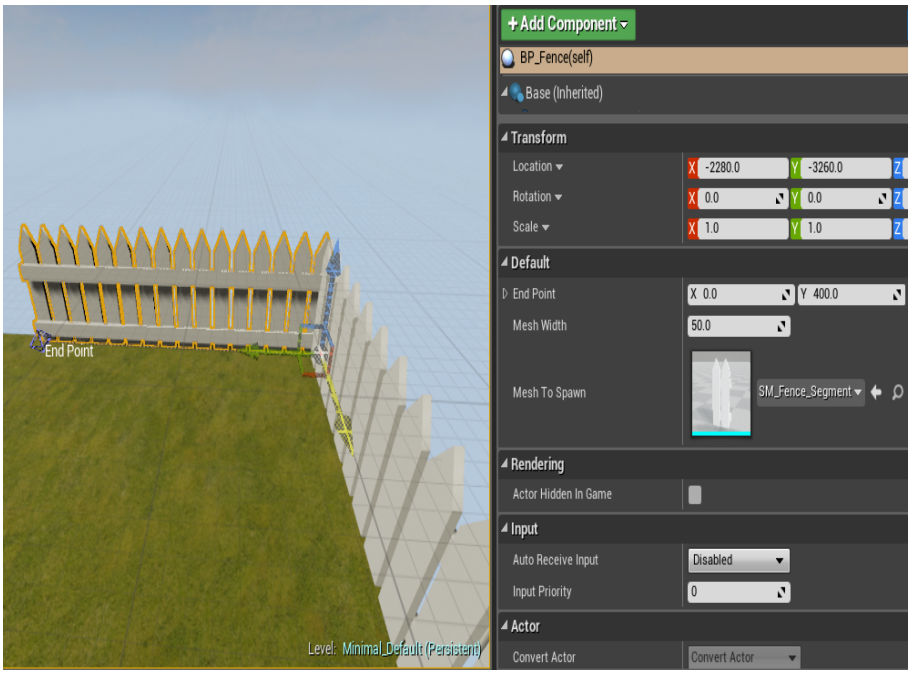


Figure 40: Results of the mesh generator.

6.7.7 *Particles in Unreal*

Unreal Engine 4 offered the possibility to use a very powerful system: the particle system, which allowed artists and users of the engine to create visual effects by using the cascade or particle editing system that Unreal Engine 4 provided.

Particles are points in space [4] that can be spawned and can have unique behaviors, these resemble many different things such as lightning bolts, laser beams, sprite sheets, etc. Once a certain particle has been created, it can be accessed in the Unreal Engine 4 in the form of an asset.

On Unreal Engine 4, a particle system was closely related to the various materials and textures applied to each particle. The particle system basically controlled the behavior of the particle, this behavior was set by using multiple emitters that joined together, form a particle.

The following are some useful terms and concepts that are of vital importance in order to understand the basic concept of what a particle is and its functionality. For further information refer to Unreal's documentation [4].

- **Module:** The module describes the behaviour of a particular particle.
- **Emitter:** An emitter was used to emit a single type of particle, multiple emitters were commonly used on a particle system.
- **Emitter actor:** One of the vital objects on a particle, this object resides in the scene, therefore holding a reference to a particular particle system.
- **Particle system component:** This is the component that allows particles to be used as templates in blueprints.
- **Cascade:** Cascade was the particle editing system provided by Unreal Engine 4, it was visually similar to the blueprint editor, minus the event graph and component explorer; these two components were replaced by the emitter editor and the curve editor which shows which properties change over time.
- **Type data modules:** These controlled the types of particle that were to be created.
- **Distributions:** Distributions were a set of data types that were used in multiple ways to handle different operations such as interpolations, number ranges and randomization.

For the project the advantages offered by Unreal Engine 4's particle system were barely grasped, for the sake of observing its particular characteristics and how people reacted to the particle system, it was decided to use a fire particle (this particle is part of unreal sample assets and can be used free of charge on any Unreal-related project), which was used on a campfire just outside the model house, since it has a big yard, things like a grill or a campfire are not uncommon, the particle system can be observed in Figure 41

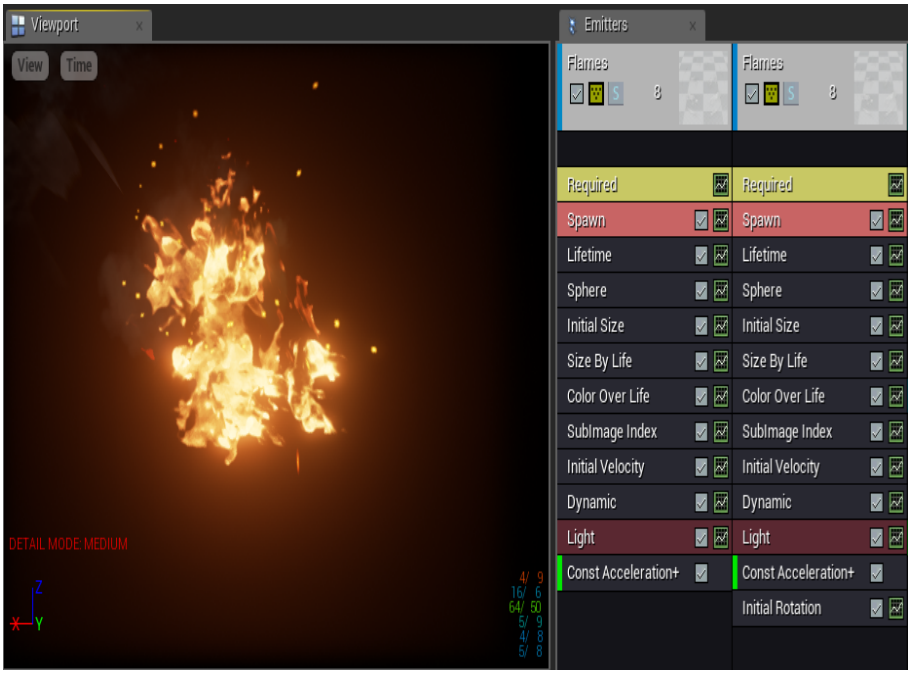


Figure 41: Fire Particle.

6.7.8 *Lightning in Unreal Engine 4*

The following section covers in detail every major aspect of lighting in the Unreal engine 4, which was an important feature to achieve the desired "realistic" daylight environment as mentioned in the problem description in section 1.1.

The human eye and brain expects light to be in a scene and interact with the objects present in it, therefore lighting features were mandatory components of the 3D world due to the improvement to the sense of immersion that gives the user. Hence, lighting features in the unreal engine 4 was of high importance for this project, due to the sense of realism that it provides to a 3D scene as stated by the results of the test developed by [13] and the above mentioned facts.

Unreal Engine 4 has a wide variety of options that developers can use, such as precomputed lighting and dynamic lighting, which contributed to the scene by giving believable lighting and shadows to the objects that were present in the environment. Furthermore, Unreal Engine 4 lighting system is very flexible and efficient [7]. It allows lighting such as precomputed lighting to create complex light interactions that cooperate in harmony with the objects in a virtual scene without sacrificing performance to make it look astonishing.

Among the available lighting features we have lightmass global illumination, reflection environments, indirect lighting cache, static lights, stationary lights, movable lights, light functions, light propaga-

tion volumes, distance field ambient occlusion, ray tracing, and many more. Although every option available could contribute by adding amazing lighting and shadowing to the scene, there are key properties in each one that offer a more adequate solution to the specific needed for the 3D model and the environment created for this project. The following sections explain in more detail every lighting feature used as well as the key features considered to incorporate into the project.

6.7.9 *Precomputed Lighting*

Among the available precomputed lighting that the Unreal engine 4 offers, it was decided to use lightmass global illumination, reflection environment and stationary lights, as these types of lights fit into the daylight scene that it was decided to create.

Firstly, the lightmass global illumination creates lightmaps with complex light interactions that can be used to create effects of shadowing and diffuse interreflection, which is one of the most visually important global illumination lighting effects as it allows light to reflect in all directions without affecting the viewing position or direction. Additionally, by pre-computing portions of the lighting contributions of the different lights present in the scene, it creates a scene where the contribution of every light bounce creates a more attenuated and evenly distributed light throughout the 3D scene.

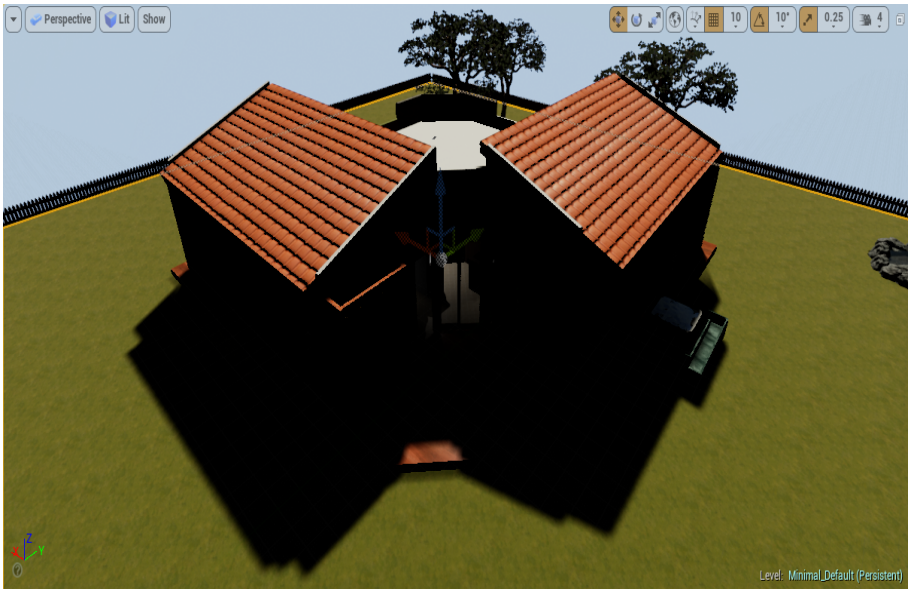


Figure 42: Lightmass feature with zero indirect light bounces.

As can be seen in Fig 42, having no light bounces results in direct lighting, and having one or more light bounces affects the overall lighting, resulting in a more attenuated light. Figure 42 shows the ini-

tial scene built by lightmass and using directional lights wit no light bounces, and figure 43 shows the exact scene, but this time applying six light bounces.

Figure 43: Lightmass feature with six indirect light bounces.

The light bounces produced by the lightmass can create effects of color bleeding depending on how saturated the base color is (also called diffuse color). The saturated color takes into account every material present as the light bounces. However, by controlling the base color of each material, each light bounce contributes to create reflections and shadowing. Fig 44 shows the color bleeding caused by the saturated base color of the different materials around the wall.

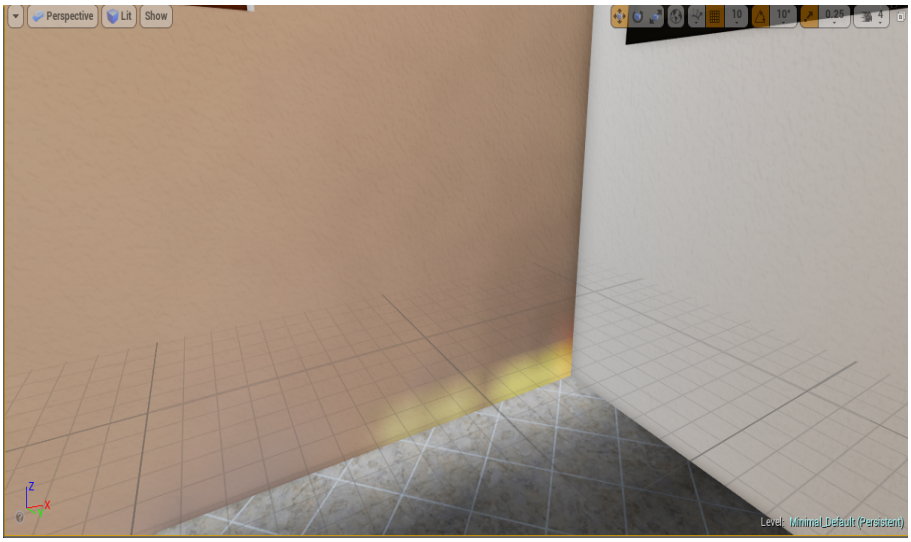


Figure 44: Color bleeding caused by diffuse color saturation in a room of the house

Furthermore, as it was mentioned before, lightmass calculates indirect shadows automatically, but it can be useful to exaggerate these shadows to transform the scene and enhance the perception of proximity in a scene, also useful for artistic purposes. This process of light manipulation is called ambient occlusion. Ambient occlusion demands high quality lightmaps to look good due to the dense lighting areas around the corners of objects, however since is a precomputed lighting method used for creating realistic shadowing to the scene, there is a considerable trade off to consider whether to remove and improve performance or not. Fig 45 shows the scene without ambient occlusion, whereas figure 46 show the same scene with ambient occlusion. The difference between these figures is slim, as the main difference focuses on the shadows and the parameters that make these shadows look more exaggerated, and as a result, more "realistic".

Figure 45: Ambient occlusion disabled.

Figure 46: Ambient occlusion enabled.

In addition to the lightmass global illumination, the other lighting feature used was called reflection environment. This feature provides the ability to add an efficient glossy effect that simulates reflections in every area of a scene where is placed, more specifically, it provides indirect specular lighting to the scene. To create this effect, the engine captures the static scene and reprojects it onto simple shapes. This is placed in the scene by manipulating ReflectionCapture actors. This feature was used mostly in materials that contained textures such as glass and wood because of the desire to minimize brightness issues when mixing together lightmaps (which resulted from the lightmass global illumination feature) with cubemaps (which resulted from us-

ing the reflection environment feature). Otherwise, depending on the roughness of the materials, when the cubemap reflection were mixed together with the lightmap indirect specular, it could result in overly bright reflections where the ReflectionAction actors were placed. This feature is shown in Fig 47.

Figure 47: Reflection environment feature.

6.7.10 *Dynamic Lighting*

Unreal engine 4 offers the option to work with dynamic lighting, which aids with the problem of casting dynamic light and shadows into the scene. Among the available features it was decided to use movable lights, distance field ambient occlusion, ray traced distance field soft shadows and dynamic shadow casting as these were efficient methods to create the desired dynamic lighting component needed in the scene.

Movable lights cast completely dynamic light and shadows, they are also fully customizable. However, they can not be baked into the light maps and do not support indirect lighting. Figure 48 shows the details panel of a movable light and how they can be customized. Under the transform section of the panel a mobility section can be found and changed from static to stationary or movable. Setting this mobility to movable indicates the Unreal Engine 4 that the selected actor is going to be considered as dynamic.

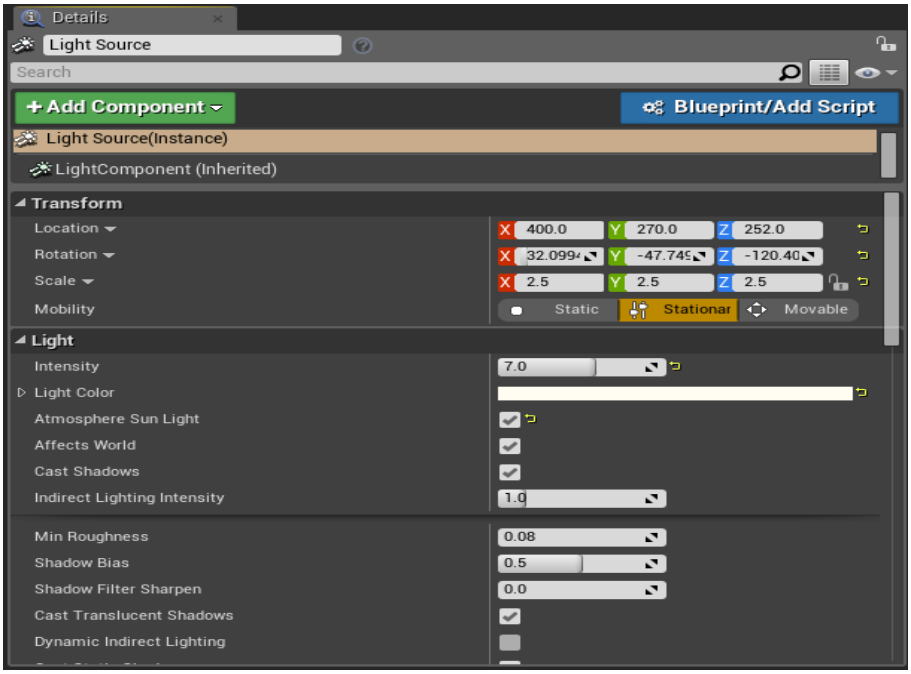


Figure 48: Movable light details panel.

Another feature supported by the engine is the distance field ambient occlusion. This generates precomputed ambient occlusion from signed distance fields volumes around rigid objects, in other words, a signed distance field stores the distance to the nearest surface at every point present in a mesh, and so this is very useful to cast soft shadows. This feature is dynamic, therefore it supports dynamic scene changes by using world coordinates instead of local coordinates. The following figure 49 illustrates how this features is seen in our project using the lighting only perspective mode. This mode was chosen to illustrate this feature as it is easier to see the soft shadows crated by the distance field ambient occlusion. It does not matter if objects are moved or destroyed, as they will affect the occlusion.

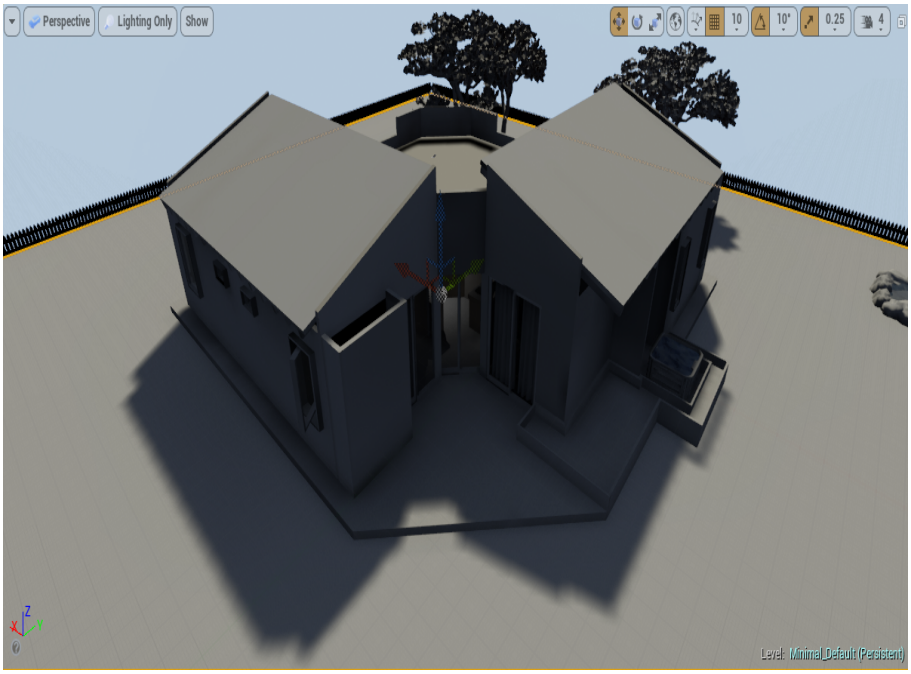


Figure 49: Distance field ambient occlusion soft shadows.

The ray traced distance field soft shadows uses the same data as the distance field ambient occlusion feature, which computes efficient shadowing from dynamic meshes. This feature traces a ray from the point being shaded through the scene's signed distance fields towards each light. Basically this ray casting can be used to compute efficient area shadowing from the dynamic meshes created by the signed distance field. Additionally, it shares almost the same limitations as the distance field ambient occlusion, but they behave much better in the distance. Figure 50 shows the scene with this feature enabled.

Figure 50: Ray traced distance field soft shadows.

Finally, dynamic shadow casting and specifically, directional light cascading shadow maps was used to cover the whole scene dynamic shadows without paying the cost of having many cascades to cover a large view range, and so dynamic shadows are faded into static shadows over the distance. This reduces the costs, it is not very noticeable and sometimes even considered indistinguishable.

6.8 OCULUS RIFT DK2 DEPLOYMENT

The Oculus Rift DevKit 2 was the newest version launched by Oculus VR by the time this project was started, which in comparison to the DevKit 1, it had great improvements regarding refresh rate, low persistence support, a positional tracking, greater screen resolution, and many other improved and/or new features.

This head-mounted display, as the center of this project, had to run as smoothly as possible in order to enhance the virtual reality experience. Therefore, we followed a guide regarding virtual reality best practices; this can be found in the Unreal Engine 4 documentation in [5]. This guide marked a few required adjustments to make the user have the best experience regarding aspects such as world scale, simulation sickness, Virtual Reality character settings and many more. The recommended configuration of every aspect mentioned in [5] was considered in order to handle the known issues and work around

them if possible. These are explained in more detail in the following subsection.

Additionally, to make the Oculus Rift work on the computer where the project was going to be tested, a software was needed to be installed, which could be found in the Oculus VR website [29]. Additionally, fewer adjustments were needed to ensure a better experience, which are explained in the following subsection.

The following subsections explain in more detail the overall setup process and the adjustments needed in order to follow the best practices that were just mentioned.

6.8.1 *Virtual Reality Best Practices*

Development of content and usage of virtual reality in the Unreal Engine 4 requires adjustments to improve problems regarding world scale, simulation sickness, character settings, content considerations and dealing with known issues.

To begin with, the first consideration to consider was the world scale, which was particularly simple to deal with, as it was decided to use a comparison table where a distance measures units where mapped into unreal distance units, hence the world to meters variable in the world settings were adjusted to 100. By default, objects placed between 0.75 to 3.5 meters from the player's camera are best viewed in virtual reality, and so by setting this world variable to 100 unreal units, the world scale would be set to meters. The following figure shows the table with the equivalent distance measure units.

Distance	Distance in Unreal Units(UU)
1 Centimeter	1 Unreal Unit
1 Meter	100 Unreal Units
1 Kilometer	100,000 Unreal Units

Figure 51: Distance Units VS Distance in Unreal Units

The world setting had great impact on the simulation sickness. If the world settings were altered indiscriminately, the immersion factor could be broken and the user might react through simulation sickness. To avoid this, it was mainly decided to not override the field of view manually, check the game with different people before testing it, and to try and keep the frame rate as high as possible. However, keeping high frame rates is a very demanding task for this project, due to the mandatory realism aspect bound to it, and so very high specifications

were required. For this reason, a balance between performance and quality needed to be assessed.

In addition to this, the next aspect to be considered was the character settings, which provided a standard measure recommendation of the height and width that a person should have inside of a virtual environment in the Unreal Engine 4, so the that this would not brake the immersion of the user. Basically, height and width should mimic real life measurements, and so it was decided to keep the default settings as they did not seem to affect both user experience and immersion.

Finally, the known issues that virtual reality has in the Unreal Engine 4 needed to be worked around. Nonetheless, the main concern in this was the fact that the amount of resources needed for rendering and displaying images in the Oculus Rift is really high, and so at the moment, there are features that do not work or present problems. Issues such as texture blurring/vibrating, screen space reflections and normal mapping were mentioned in [5] and workarounds were given for each one.

In this project, the main concern was and still is the frame rate problem, as it was the most noticeable issue seen in the tests in chapter 7. To workaround this, it was decided to sacrifice performance to achieve a rather stable frame rate. However, this restricted the ability to use the engine to its fullest potential.

6.9 SYSTEM SPECIFICATIONS

The application is currently being tested on a computer with an Intel Core i5-2320 CPU at 3.0 GHz, a 8.0 GB RAM, a Nvidia Geforce GTX 560 TI and a Windows 7 64-bit operating system. The project can run up to 150 FPS in desktop mode and in a range going from 39 to 65 FPS in Oculus Mode. Additionally, the system has Oculus Rift support and it focuses on desktop application mainly, so further improvements should be made in order for the system to run on mobile devices or other platforms. Finally, an input controller was used in order to facilitate movement and actions commands to the user.

Additionally, the system does not provide any additional information of objects outside of the virtual environment, so it needs to have someone to watch over the user that is currently exploring the virtual environment, as it is necessary to provide assistance and safety to the user.

6.9.1 *Oculus DevKit 2 Setup*

To get the Oculus DevKit 2 to run in the computer where it was going to be tested, it was necessary to download and install two pieces of software directly from the Oculus VR website. The first one is called the Oculus Rift DevKit 2 Run-time, which is an Oculus Service that manages the configuration of specific features such as eye relief settings, profile information, display mode, among others. The other piece of software is called Oculus Rift SDK, which as its name states, is the software development kit that provides the programmer with a set of tools to develop specific assets for the Oculus Rift.

This project uses the Run-time mainly as developing was not the only task, as the entire process of modelling needed to be dealt with. Hence, the tools provided by the Run-time covered the need of connecting the computer with the head mounted display, which facilitated interaction with the Oculus Rift greatly.

6.10 TECHNICAL ISSUES, SOLUTIONS OR WORKAROUNDS

While the development was ongoing, multiple issues arose, this subsection intends to list the most relevant, with the intention to not only show the difficulties faced during the implementation, but to state problems regarding virtual reality that were encountered.

6.10.1 *Google SketchUp Model and 3DS Max*

As explained in sections 6.1 and 6.2, multiple problems arose with the initial models, not only the house itself, but the furniture had texture problems. These issues held back the development of the project until assets could be improved, and problems of creating new materials and importing new textures were fixed.

In order to make this process easier and faster, each individual model needed to have their normals fixed to avoid issues such as incorrect shadows or shadow bleeding appearing in the models. The model should be normal mapped or the texture mapped to match the shape of the model. However, the former can a better choice since it allows the usage of different materials and textures if that is something that needs to be done or multiple instances of the object are needed.

The last problem faced during the 3DS Max development was regarding the export tool that handled the materials. 3DS Max provides some advanced materials for rendering, such as V-ray materials, unfortunately, those materials were created specifically for 3DS max and

thus could not be exported, when models with these materials were exported they would appear as though they were white materials in the engine.

To aid this problem, the solution used was fairly simple, a material library was created, containing the materials that were used on the model, this solution however allowed only basic materials to be exported, so the advanced materials that 3DS Max offered were discarded.

6.10.2 *Unreal Illumination, Shadows and Collisions.*

During the beginning of the projects development, it was decided that static lighting was to be used on the model, as most of the components were static objects in the scene. This approach was unfruitful however, when the lights were baked, the shadows obtained were clunky and highly unrealistic, these shadows can be observed in figure 51,

Figure 52: Inacurate shadows

In order to solve this problem multiple forums were consulted, and possible workarounds were found such as making sure no UV map overlapped on the channel designed to store light information (in Unreal there is usually two channels containing UV maps, one of them maps the texture over the mesh geometry, the other channel is used to map the shadows over the texture), or increasing the size of the UV maps so the information could be stored correctly.

The first approach was taken and it fixed some shadow-related issues like the shower or the toilet having incorrect shadowing on them, however, it did not solve the issue of objects projecting inaccurate shadows on other objects; the second approach was a bit harder to implement, as it was impossible for us to increase the size of the UV maps without leaving the boundaries of the UV map grid, which would just undermine the previous approach partial solution.

On one of the forums the recommendation was to use dynamic lighting (for further information refer to subsection 6.7.10); dynamic lighting implied an increase in computations and allocated resources, but at that point in time, it was the best option available. On the project settings, specifically on the section regarding rendering, the Gbuffer was enabled and so was the Distance Mesh field options in order to enable dynamic lighting features. Additionally, the world light (the sun "emulator") had to have some of its properties modified, the ray tracing capabilities were enabled as well as the cascaded shadow maps. The results can be seen in figure 53.

Figure 53: Inaccurate shadows

As seen in figure 53, the shadows were fixed after applying the method just explained. These shadows were placed correctly in the objects they should have been projected in regard of the ambient light's position.

6.10.3 *Oculus Rift issues*

As the project approached its final stage, the Oculus Rift integration was necessary, when the project was integrated to the Oculus a big issue arose: The frames per second dropped drastically when the first test was performed on the Oculus Rift.

As seen enclosed in the red oval, figure 54 shows that the project was running at 60 FPS, it was decided to limit it to that amount since more FPS would just waste resources and 60 FPS has become the staple of excellence in the video game industry, however the project was perfectly capable to run smoothly up to 120 FPS.

Figure 54: FPS of the project running on the PC

The first time the project was used alongside the Oculus Rift, the FPS obtained were a lot like what figure 55 shows, note that this and figure 56 are not looking as smoothly as it should. This happens as a result of the screenshot taken from the computer and not the Oculus Rift, so the image look distorted inside the Oculus Rift display.

The frames per second showed a considerably big drop, going from 60 steady to a 14-23 range on the FPS count, this caused people that were asked to try it out before the test, documented in section 7.2, to get simulation sickness to the point where they had to stop wearing the head mounted display due to the nausea.

Figure 55: FPS on the Oculus, Screen percentage 170

After applying some of the techniques described in subsection 6.8.1, most prominently the usage of the command HMD SP which reduces or increases the Screen Percentage parameter, a steady frame rate ranging from 39 to 65 was achieved by using a 120 screen percentage, as show by figure 56

The screen percentage parameter represents a percentage of the overall resolution of the scene. While tweaking the screen percentage graphical quality was sacrificed in order to obtain a smoother frame rate, when a balance of both of them was achieved, the second test was performed as illustrated by section 7.2.

Figure 56: FPS on the Oculus, Screen percentage 120

7 TESTS AND RESULTS

7.1 FIRST TEST

Following the first iteration of the development, which included: the house model, some furniture and basic lighting settings, it was decided to make a survey to determined how potential users perceived the project on its most basic form, and take some advice on things that could be improved in next versions.

Figures 57, 58 and 59 show how the project looked at the time this survey was made.

Figure 57: Visualization V1-1.

As seen in these figures, the version of the project at that time was on a very early version, the house was empty and there was nothing outside of the house but a basic floor with a grass texture on it, also, the only light that was affecting the house was a directional light "emulating" what the sun looked like.

The visualization works with a first person view, the character which can not be seen is just an empty pawn with a camera attached to it. The input for the computer were a mouse and a keyboard, using a setup that is commonly used in first-person shooter games, which included movement through WASD keys. There was a functionality to sprint trough the house, however it was decided that it was not nec-

essary, as one of the goals of the project was to let the user explore the virtual environment slowly, while looking through the entire world and letting him or her focus on paying close attention to details if possible.

Figure 58: Visualization V1-3.

Figure 59: Visualization V1-3.

For this survey 15 people with no technical knowledge and 11 people with technical knowledge were surveyed.

The intention with the distinction was to put in a group a people with a certain level of expectations regarding the project due to previous experiences on the field and people that were experiencing this type of project for the first time or didn't have specific expecta-

tions due to lack of knowledge and experience related to this type of projects. The qualities that made the surveyed people belong to either group are the following:

Technical people were people with considerable knowledge on subjects related to virtual environments, video games or programming: among the people that were part of this category were hardcore gamers, computer scientists and electronic engineers.

On the other side, the non-technical group was conformed by people that did have the previously mentioned characteristics.

In the survey they were asked about the perceived graphical quality regarding some characteristics of the scene, with level 1 being non-realistic and 5 being realistic.

The following questions are the ones that were intended to be qualified with points from 1 to 5:

- How would you qualify the overall realism on the scene?
- How would you qualify the illumination observed on the scene?
- How would you qualify the reflections on the scene?
- How would you qualify the shadows observed on the scene?
- What level of realism would you place the furniture at?

To finish the survey two open question were asked: one regarding things the user did not like, and the second things that the user thought could be improved.

7.1.1 *Closed Questions*

Tables 1 and 2 include the results of the survey, specifically the closed questions. The open question results will be explored in section 7.1.2

Age	Gender	1st Question	2nd Question	3rd Question	4th Question	5th Question
44	F	5	5	5	5	5
17	M	4	3	4	4	5
50	M	5	5	4	4	5
52	M	3	4	3	5	3
45	F	3	5	4	4	5
16	F	5	4	4	5	4
21	M	3	4	3	4	4
30	M	4	5	4	5	3
28	M	5	3	5	4	3
27	M	4	5	4	4	5
25	F	3	5	5	5	3
24	F	3	3	3	3	3
22	F	4	3	5	3	4
21	M	3	5	4	3	5
19	M	2	3	4	5	3
18	M	5	3	4	3	5
Average		4.064	4.334	4.33	4.4	4.33

Table 1: Non-Techincal Survey

For the non technical survey, the average for the first question (regarding overall realism) was 4.13, which was overall the lowest. This suggest that even when things considered to be extremely important for realism had high scores, some other things were missing, in subsection 7.1.2 further discussion about some causes to explain this characteristic being the lowest on average will be available.

Age	Gender	1st Question	2nd Question	3rd Question	4th Question	5th Question
80	M	4	4	3.8	3.8	4.8
23	M	4	5	4	5	5
25	M	4	3	4	3	3
19	M	5	4	4	4	5
23	M	3	5	4	4	4
20	M	4	5	5	4	5
21	M	4	4	5	5	5
18	M	4	5	5	5	5
21	M	4	5	5	5	5
20	M	4	5	5	5	5
20	M	4	4	4	4	5
Average		4.0	4.45	4.43	4.34	4.7

Table 2: Technical Survey

Question number two, which intended to measure what people thought of lighting scored 4.4, making it the second highest. It was believed that the highest score got that score as a result of the Unreal Engine's 4 amazing lighting features, as the project makes use of their

dynamic lighting algorithms and ray tracing, which give some very beautiful results when used.

On the other hand questions number 3 and 5 had the same average score of 4.33, which is not surprising as there were not many reflections and furniture, however, the positioning of these on the 3D world had a considerably good impact on the users.

Lastly the highest rated question was number 4 with a mean of 4.46, which was used to measure the shadow quality which is completely normal given that if the lighting looks good so will the shadows, and again thanks to Unreal Engine's algorithms some good visuals were achieved.

Overall, the non-technical people were particularly interested in the idea and gave some positive reviews regarding the project, the average score for the entire survey was 4.3, which is a good score taking into consideration that the model was in an early version.

For the technical people survey the average score for questions 1 through 5 were 4, 4.45, 4.43 , 4.34 and 4.7 respectively; this was a big surprise due to the fact that it was thought that the scores for the technical user test would be lower than their counterparts, since people with technical knowledge tend to be more thorough with both examination and scores.

It was a big surprise, however, that the lowest score on this set of scores was the one regarding shadows (4th Question), it got an average of 4.3, which was the highest on the non-technical survey 1. It was hypothesized that the lower score was due to the presence of some flickering shadows in a portion of the map and that could have impacted the technical users negatively, and non-technical users did not notice or choose to ignore this particular problem.

On the other hand, the highest score was related to the furniture scored a 4.7, considering most furniture are assets coming directly from Unreal examples that is not surprising at all, furthermore, that is one of the points that was expected to get a higher score overall. Additionally, with furniture of such high level of realism, one of the big considerations for the project was that tech-savvy people would be attracted by the high amount of detail this furniture has. It was expected that the furniture's high quality could divert in a certain way flaws the model would have, as Elhelw[13] highlights, unconsciously the human eye looks for certain characteristics, this furniture have quite the advanced materials, which make them look almost real, and so the noticeable difference between the house and some of the realistic assets could be strange.

One of the reasons the spawning point where the user fist appears on the virtual environment is placed where it is due to the fact it faces that furniture and gives a great first impression, figure 60 illustrates how the realistic couch asset looks when visualized on the asset explorer.

Figure 60: Couch on asset explorer.

The scores obtained in both surveys had little variation on the average. The overall realism was 4.0 and 4.06, which is understandable, as the project was on a early version, so the people on both sides considered there was a need to further improvement. Questions 2 and 3 got almost identical scores on both surveys with both questions getting a mean of 4.33 on the survey for non-technical people and a mean of 4.45 and 4.43 on the technical survey. It could be said that this questions had the most consistent scores. The project used dynamic lighting with ray tracing, which is an advanced illumination technique so it was expected this feature would achieve good scores on both surveys.

7.1.2 *Open Questions*

During the first survey two open questions were asked, this was done in order to obtain information that the closed questions might have missed. The main point of the open questions was to understand which of the project's features the people considered good and that would not need extra work, as it was known that existing features needed more work as well as elements that people considered to be missing from the project. The two open questions were the following:

- Which things in the project were not to your liking?
- Which of the project's features would you improve?

Table 3 exposes particular elements or characteristics that the non-technical tested users did not find likable. On the other hand table 4 shows features that the project had but the users thought could be further improved.

Issue	Percentage
Too empty	28
Lack of decoration	50
Colors	21

Table 3: Non-Technical Survey 1, Open Question Number 1

Feature	Percentage
Movement	14
More decoration	35
More plants on the garden	21

Table 4: Non-Technical Survey 1, Open Question Number2

As seen by both tables not many issues or things that could be improved were found, most users complimented the work done and liked the idea. One of the most particular things brought to light by these two questions was that non-technical people were not as interested in the graphical quality of the project, but rather in things such as the decoration of the house. They particularly did not like that the space was so big and yet so empty. The interesting part of this test was regarding the movement; most people had problems moving around (this particular issue was addressed with the usage of a Controller, See figure 15), yet only a 14% considered it to be an issue, not a single person mentioned something regarding collisions or any other technical yet visible features.

The technical users on the other side had completely different observations, as shown by tables 5 and 6. It can be observed that not a single topic was mentioned on both the technical and non-technical surveys, both groups of people were looking for completely different aspects.

Issue	Percentage
Can see through some objects	14
Wrong Collisions	28
Graphical quality	21
Poor textures	7

Table 5: Technical Survey 1, Open Question Number 1

Table 5 shows that most of the issues identified by the users were things that are commonly seen in video games, as the project was on an unfinished state, some of them were overlooked in order to have the test to be performed on time. Its also interesting to note that a considerable amount of people did not find anything wrong with the project. On contrary, people that tested the project considered that plenty of the features could be improved, since most of them had gaming backgrounds some of these features were understandable, since they did not feel as satisfied by simply walking around the model but requested things like Jumping or improved controllers.

Feature	Percentage
Textures	21
Collisions	28
Height of the camera	7
Controls	14
Jumping	21

Table 6: Technical Survey 1, Open Question Number 2

Comparing both questions it could be easily identified that non-technical people were looking for things that made their experience more immersive, such as having more furniture, paintings, plants, among others; they wanted the house too look as they expected a model house to look, fully furnished to help people pictures their lives in their potential new homes, its also noteworthy the fact that most of the things they wished to improved upon were the same things they identified as issues, since they had no idea how an engine worked or how the program was made, these issues were they only noticeable aspects to be improved according to them.

The technical users were looking for a better experience rather than having the house look better. In fact, they did not thought the house felt empty and even felt it was looking good without much furniture and decorations.

7.2 SECOND TEST

After one month of addressing all the issues that were identified during the first phase of testing and fixing multiple problems with the Oculus Rift performance (which are identified in subsection 6.10.3) a second test was performed. This test was done with the objective of identifying if previous issues were solved, if new ones were created and how comfortable the users were using the Oculus rift.

The test was split in three parts: closed questions that asked about the scene quality and the virtual reality experience, closed questions that aimed to obtain information regarding the Oculus Rift effect on people and the same open questions done in subsection 7.1.2.

For this test the project looked as figures 61 and 62 show, the level originally had more assets, such as foliage and even some butterflies flying around. However, due to performance issues they were cut out from the level. Also, as pointed by the results of the open questions exposed on subsection 7.1.2, the white walls were also changed to white fences, since people felt they were caged and did not particularly liked them.

As it can be seen in figure 62, aside from foliage and fences, a rocky fountain was added to show how water could be simulated, and more assets to the exterior of the house were added. However, it was decided to focus on the interior of the house due to performance issues, and so the exterior was left out rather simple without many assets. Consequently, it was decided to use the particle system provided by the engine by setting up a campfire outside of the living room as shown in lower right corner of figure 61.

Figure 61: Visualization V2-1.

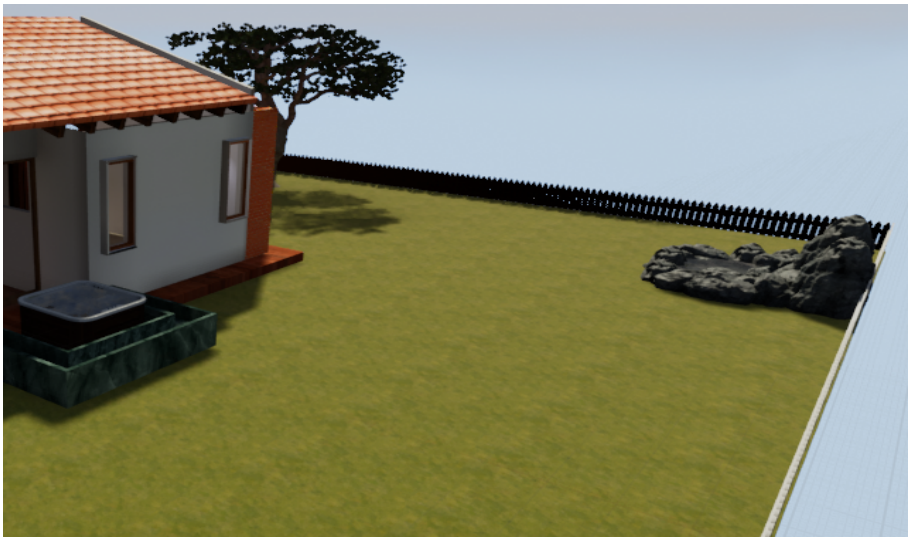


Figure 62: Visualization V2-2.

In addition to the already stated changes, new assets were brought in to fill every empty room as the one shown in figure 57. As a result of many non-technical people in the previous test criticizing the project due the lack of furniture, decorations and even materials. For this reason, it was also decided to fill every room with at least three or more assets and use no more than three materials with achromatic colors if possible.

For this test, most of the lighting features were added, as explained in section 6.7.8. The project was mostly in its final state when it was decided to perform this test. Aside from a technical issue that made the Oculus Rift frame rate to oscillate between 25 to 35 frames per second in different areas of the environment, the entire project was nearly finished. Also, a PS3 controller was used as stated in 5.4 to aid the need of simplifying movement in the virtual scene.

For this survey 15 people with no technical knowledge and 15 people with technical knowledge were surveyed. The following questions were intended to to be qualified by using a numerical value ranging from 1 to 5, where 1 was considered the worst and 5 the best. These closed questions were aimed to explore about the scene quality and the virtual reality experience.

- How would you qualify the graphical quality of the scene?
- How immersive was the virtual reality experience for you?
- How would you qualify the illumination of the scene?
- How would you qualify the shadowing of the scene?

Following the previous closed questions, there were another set of closed questions inclined to analyze different aspects regarding

the Oculus Rift. Consequently, these were intended to to be qualified using a numerical value ranging from 1 to 5, where 1 was considered to be too little and 5 to be much. These closed questions that aimed to obtain information regarding the Oculus Rift effect on people.

- How much eye fatigue did you feel?
- How annoying was it for you to have the head mounted display on you while exploring?
- How nauseated did you feel after exploring the virtual environment?
- How intuitive was exploring the virtual environment?

Finally, as stated before, the open questions did not change in order to compare and contrast opinions in both tests, these are explained in section 7.2.1 in more detail.

7.2.1 *Closed questions regarding the level*

Table 7 and 8 show the results of the survey of the technical people, meanwhile, table 9 and 10 show the results of the non-technical people.

Age	Gender	1st Question	2nd Question	3rd Question	4th Question
m	18	4	3	3	2
m	30	5	4	5	4
m	22	5	4	5	5
f	21	4	5	5	4
m	21	5	5	5	4
m	21	4	4	5	4
f	21	4	4	5	3
f	22	5	5	5	4
m	19	4	4	5	5
m	28	4	3	5	5
f	20	4	3	5	3
m	21	5	4	5	4
m	23	5	5	5	4
m	20	4	4	5	5
m	20	4	4	5	3
Average		4.46	4.2	4.9	4.2

Table 7: Technical Survey 2, Scene Quality

For the technical survey regarding scene quality (shown in table 7), the average of the first question was 4.2. This question was bound

to graphical quality and overall perceived realism in the scene, this then suggests that most people perceived realism in what was done, except for people who did not pay closed attention to details and instead tried to jump or interact with the environment more than it was possible. Others expected a realistic game and were displeased enough with the result to the point of considering aspects such as illumination, reflections and overall realism to be average. The second question had a mean of 4.0. Everyone mentioned that it was an enjoyable and immersive experience in general, in fact some of them even thought they were going to hit their heads or fall down as they came close to the different objects present in the scene. However, opinions felt as if they were biased on how little or how little much they felt nauseated.

On the other hand, the third question was the highest, as most of the technical people were impressed by how realistic this feature was. This question scored a mean of 4.8, which indicates that only one person considered illumination to not be worth of a 5. Finally, the fourth question was aimed to evaluate shadowing in the scene and scored a mean of 3.9. This was a rather surprising fact, as many people ignored shadows and most of them even asked if the project had any at all.

Consequently, the technical survey regarding the Oculus Rift experience (shown in table 8), as stated before had a rather different number rating scale, hence a 5 is not necessarily the best answer. The first question obtained a mean of 2.7, which indicated how much his or her eyesight was fatigued during and after the experience. This is an anticipated result, as this is very common in the field of virtual reality. The second question was aimed to determine if people felt that the setup provided for them to explore the virtual environment did cause uncomfortableness. This question scored a mean of 2, hence indicating that this was not a problem for most of them.

The third question was directed to the overall simulation sickness. And as it was mentioned, many people felt as if they were going to die, while others enjoyed the experience without many issues. The average for this question was a 3, which confirms what it was observed about this experience being unique for every person.

Finally the fourth question tried to determine how intuitive was the exploration and movement using the head mounted display. This obtained a mean of 4.4, which in this case indicated that it was very intuitive. This result might be due to having incorporated a controller and a chair that could rotate 360 degrees so that people could move easily.

Age	Gender	1st Question	2nd Question	3rd Question	4th Question
m	18	5	3	5	4
m	30	3	2	2	5
m	22	2	1	1	4
f	21	3	2	4	4
m	21	2	1	2	5
m	21	1	3	3	5
f	21	1	3	3	4
f	22	1	1	4	4
m	19	2	4	2	5
m	28	5	3	2	5
f	20	5	1	4	3
m	21	4	2	4	4
m	23	1	1	4	5
m	20	3	1	4	5
m	20	3	2	3	5
Average		2.8	2.134	3.33	4.7

Table 8: Technical Survey 2, Oculus Rift Experience

In addition to the technical survey, the non-technical survey indicated similar results in both sets of closed questions. The first set of closed questions regarding scene quality are shown in table 9. The first of this set of questions scored a mean of 4.2 as well as the second and fourth question. All these questions were aimed to evaluate the graphical quality and overall realism of the scene, except for the second question that was aimed to determine the level of immersion. As this results show, most non-technical people analyzed these questions as a whole, they consider they were complementary and scored them with the same rating number or higher value in most cases. The third question was rated higher than the other three. This question was aimed to determine how illumination was perceived by the testers. It scored the highest mean in both the technical and non-technical surveys, as it seems that most people in both surveys recognized illumination over shadows and many other details such as particles and materials more than anything else on the 3D environment.

The second set of closed questions is shown in table 10. As stated before, this set of questions tried to evaluate specific aspects regarding the overall Oculus Rift experience. The first question scored a mean of 2.1, which indicates that eye fatigue was definitely not the main problem that could have caused biased opinions of virtual reality. The next question scored an average of 2. Comparing this mean with the one obtained for the technical people, it is possible to determine that for both groups, the setup prepared to use the Oculus Rift and the head mounted display itself was not considered a problem.

However, most of the technical people believed it could be further improved.

The third question for the non-technical people scored a mean of 2.6 in comparison with the 3.1 for the technical people. In general, both technical and non-technical people, felt nauseous and some of them even sweat and label the experience as something that amazing, but not particularly something they were fond of. In addition to this, the fourth question, which tried to evaluate how intuitive the experience was for people scored the highest with a mean of 4.0, similar to the technical people, which also consider this to be rather intuitive.

To conclude this set of questions, it is possible to determine that both technical and non-technical people had very similar answers in regards of scene quality and the virtual reality experience. There were noticeable differences in the comments they provided for the open questions, but the variations in the closed questions were minimal.

Age	Gender	1st Question	2nd Question	3rd Question	4th Question
F	20	4	5	4	4
F	49	5	5	5	4
F	53	4	3	5	4
M	57	5	4	5	5
F	59	5	5	5	5
F	54	5	3	5	3
M	59	3	4	5	5
M	40	3	3	5	5
M	59	4	4	5	5
F	45	4	5	5	4
F	72	4	5	5	4
M	52	3	3	4	3
M	21	4	4	5	4
M	75	5	5	5	5
F	21	5	5	5	3
Average		4.2	4.2	4.86	4.2

Table 9: Non-Technical Survey 2, Scene Quality

Age	Gender	1st Question	2nd Question	3rd Question	4th Question
F	20	2	3	4	4
F	49	1	1	1	5
F	53	2	2	1	1
M	57	1	1	2	5
F	59	1	1	3	4
F	54	3	3	3	3
M	59	1	1	1	4
M	40	4	1	4	5
M	59	2	4	4	4
F	45	2	1	3	5
F	72	1	2	1	3
M	52	3	4	4	3
M	21	5	4	5	4
M	75	1	1	1	5
F	21	3	1	3	5
Average		2.13	2.0	2.66	4.0

Table 10: Non-Technical Survey 2, Oculus Rift Experience

7.2.2 *Open questions*

During the second survey the same questions asked in the first test were asked again. This was needed in order to fill the gap left by the closed questions regarding people's opinions. As stated in the open questions subsection 7.1.2 for the first test, the idea behind these questions was to understand which of the project's features were considered good or acceptable and did not need extra work, also which existing features needed to be changed or improved. The following questions were asked:

- Which things in the project were not to your liking?
- Which of the project's features would you improve?

Tables 11 and 13 show the issues that were disliked. On the other hand, tables 12 and 14 show the features that the project had, but users considered they could be further improved. These tables correspond to technical and non-technical people respectively.

Issue	Percentage
Can see through some objects	14
Limited ambient	14
Simulation sickness	35
Motion Blur (Blurry Textures)	21
Graphic Quality	7

Table 11: Technical Survey 2, Open Question Number 1

Table 11 show five issues that were identified by technical people. Among these features, simulation sickness was by far the most noticeable as it scored the highest percentage among them. Many people finished exploring the virtual environment and soon after felt nauseous and had to laid down. Simulation sickness is in fact an issue and there is the need to explore new ways to tackle this problem. In chapter 9 future tests and improvements are reviewed.

Additionally, technical people did not complained about graphical quality as much as the non-technical people, as table 11 this was the lowest scored issue. Another issue for the technical people was the fact that by moving their heads near a surface, they could see through them. This was really consider a problem for the non-technical people as they were more concern about what decoration. Lastly, the other two identified issues were indeed known issues, as stated in section 6.8.1, and the fact that different people were able to notice them exposes the fact that a better way to workaround these was necessary.

Feature	Percentage
Collisions	21
Movement (Jump)	35
Turning around	21
Controller	21
Object Interaction	28

Table 12: Technical Survey 2, Open Question Number 2

Table 12 show five features that technical people wanted to be able to do or improve. The most noticeable feature was the ability to jump, they felt the environment limited actions too much. A high percentage of people also requested object interactions. This feature was considered in the scope of the project, but decided not to include it and focus more on specific technical issues such as Oculus Rift frame rate problems. The controller, unlike the first test, was a PlayStation 4 controller and not the keyboard, which in the previous test, people found that it was really difficult to understand how to use it. In this

case, most people did not have problems, but considered movement to be strange overall. Finally, people highlighted collisions that they believed were not placed correctly or were not present at all.

The following table show the issues identified by the non-technical people. The non-technical people surprisingly did not mentioned simulation sickness as much as the technical people. However, both simulation sickness and the lack of decoration scored 14%, which was the highest among the eight issues shown in table 13. The rest of the issues varied greatly from difficulties to turn around to feeling claustrophobia and anxiety. Most of the non-technical people were over 30 years old, whereas the technical people were around 20 years old. Additionally, as mentioned in the previous test, non-technical people seem to focus more on details regarding decoration, color and aspects related to the house more than technical issues or graphical quality.

Issue	Percentage
Need better decoration	14
Turning around	7
Color	7
More info regarding movement	7
Simulation Sickness	14
Graphic Quality	7
Anxiety	7
Not enough sounds	7

Table 13: Non-Technical Survey 2, Open Question Number 1

Finally, table 14 show four features highlighted by the non-technical people. Many people did not have actual comments regarding what they liked or disliked, and so they considered that the virtual environment lacked better sound and the world setting could be better. In general, most of them could not find the words to describe what they wanted to say, due to the fact that this experience was rather new to them. Graphical quality was seen as something that could be improved. This was particularly explained by people who did not feel simulation sickness and explored the world looking at everything for around 5 minutes or more, unlike people who felt nauseous and stop exploring for less than a minute.

Feature	Percentage
Ambient	28
Graphic Quality	21
Simulation Sickness	7
Not enough sounds	7

Table 14: Non-Technical Survey 2, Open Question Number 2

All things considered, both technical and non-technical people were amazed by the virtual reality experience. There were too many people that felt simulation sickness in contrast to what was expected. After revising the thirty surveys, it was clear that most of the technical issues present were highlighted, such as blurry textures, collision problems and even frame rate problems in specific areas of the environment. This suggest that more work was needed in order to improve the overall experience, and the fact that the lack of experience limited the ability to determine what could be done and what not, was clearly identified by how many people were nauseated after they used the Oculus Rift.

7.3 INFLUENCE OF USER FEEDBACK ON THE IMPLEMENTATION

The user's feedback played an important role to improve the model over the development of this project. There were noticeable improvements such as the change that can be seen in the first test 59, where a whitewall was used to "lock" the user from seeing the emptiness behind the walls, most user did not like the wall or wanted to see beyond it, so it was decided to replace this wall a fence generation system specified in subsection 6.7.6 to allow them to have a complete view of the whole environment. Additionally, many other changes contributed to refine the overall project by adding specific aspects that users felk the project was lacking. Among these changes are, the increase of furniture elements, foliage on the model's garden and a changes on the materials, like the wood material shown in 32, which was decided to add imperfections such as dirt in order to make the house a more realistic.

8

CONCLUSIONS

To meet our first objective it was decided to developed a prototype as proposed, during this development multiple issues came into light that were not considered as the project started. First of all, modelling tools such as Sketch Up or AutoCAD do not invite architects or civil engineers to model buildings appropriately for a post process in a more advanced modelling software, the lack of UV maps makes it impossible for a smooth transition to a display tool, and this process makes the lack of an automated tool to avoid this tedious phase to be considerably longer. To reduce the time of development on a project of this type, the models have to be properly created, including the UV mapping of the models are a must, afterwards creating good-looking materials and applying them to the model would not be a difficult task if the UV maps are placed correctly and it would give the chance for the user to freely change the materials of the models to make the potential house look however they wanted it to look like, which was a feature that was asked a lot during the testing phase.

Another issue that delayed considerably the prototype development was the lack of free assets. Since creating the assets was not part of the project's objectives all of them were either obtained from Unreal's samples or free websites, hence it was a long process of checking if they had UV maps or actually giving them UV maps for them to be used in the project.

Another equally important objective of this project was to use computer graphic techniques to enhance the perception of realism of the 3D architectural model dsiplayed in the virutal environment. These algorithms were in fact the key point to achieve realism, and so by using the Unreal Engine's lighting features such as precomputed and dynamic lighting, it was possible to create an astonishing virtual environment.

These features provided by the Unreal Engine 4 are efficient and robust, with the exception of some dynamic features that were not always fully optimal due to GPU high demand and very few limitations like non-uniform scaling and poor quality in large objects for instance. In despite of these limitations, global illumination was achieved without many complications. The main problem that arose after using these features was the Oculus Rift itself, as it demanded top-notch hardware, and so a balance between performance and quality was needed.

It is important to note that with this project it was intended to analyze the viability of a tool to aid with the problem of visualization of architectural models using virtual reality. This certainly includes a laborious process of transforming a simple 3D architectural model into a one that could be perceive as realistic, and so this process alone implied a lot of manual work and the use of software without much technical expertise. Also, when lighting features were added to scene, specially when dynamic lighting was added, a lot issues appeared, such as deformed shadows, algorithms not working properly due to bad organized model structures and limited supported components for Ambient Occlusion. At the beginning of this project it was decided to include features like real life movement in a closed environment and interactions with the environment. However, many of these were not implemented due to the the need of improving the virtual reality experience, which had more priority.

Additionally, the main concern of this work was the Oculus Rift interaction and usability, in order to give users a pleasant experience. This however was a huge problem throughout the project, having to work with a rather unstable software created by Oculus VR that had too many limitations regarding compatibility, stability and visualization issues, was indeed a limitation.

During the Oculus Rift interaction some problems were faced, it was not a surprise however, since the Oculus Rift is on its beta stage and not many developers have used it extensively so most of the troubleshooting was either a hit or a miss. The solutions provided on forums worked for some people and did not work for some others, version changes also hindered the project culmination; version 0.7 was not longer able to support laptops so the most advanced hardware available was also rendered useless. These projects might become more stable as the Oculus reaches its release date, but for now, lowered graphical quality, frame rate issues and the motion sickness caused by the frame rate really hinder it.

The testing phase showed that the project had high acceptance among users as most people liked the concept, the project was praised for the graphical quality achieved and the immersion produced by it even if it could be improved, but small issues like more decoration (which was reduced to increase the frame rate) were a setback for the project to score higher, the next iterations of the Oculus and Unreal Engine could definitely solve most of the technical issues found during the project's development and take that final step towards what was envisioned with this project's development.

As far as viability goes, the project received very good reviews, the users felt it was a good idea albeit they mentioned a lot of improvements if the prototype was to ever be used as a sales tool, it can be

considered to be a few steps away from being a tool that can actually be used commercially. On the technical side there are multiple shortcomings: the frame rate drop is unacceptable as it makes people get simulation sickness, making the tool unpractical; the constant updates make it a problem too, since at the beginning of the project laptops were supported by the Oculus runtime and the project ran well enough on them, but as a new version came out, laptops were not supported at all. Stability is a must if the process is to be formalized, a better computer might improve the frame rate issue and give the project more options to include more models without taking a hit on performance.

Lastly, with this project it was decided to analyze viability, not only in terms of user's acceptance, but also in terms of how viable this project might be as a commercial tool. Clearly, the viability of this project seen as a commercial product is rising every year, and soon it could become a useful tool as hardware and virtual reality are becoming more and more stable and commercial. However, visualizing a realistic 3D environment in virtual reality has too many problems that limit the viability of this project as a commercial product, and hardware demand is one of them, therefore working on a budget is indeed a problem. Aside from the hardware limitations virtual reality is bound to, the overall process of adjusting an already existing 3D architectural model, or even creating a 3D architectural model demands time, knowledge and unfortunately, it is a process that cannot be automated as of right now.

9 FUTURE WORK

The project obtained some impressive acceptance among the surveyed people and there is room for a lot of improvement for future projects that are interested on developing projects related to this one in particular.

First of all, small improvements regarding the illumination could be done, making a thoughtful calculation of the user visibility to manage the lowering quality of the shadows when they are far from the user's position, since our project lowers and even removes some shadows if they are too distant from the user, however the user can still perceive the low quality or lacking shadows.

As shown by the test results most people were interested in having a more visually attractive house, since increasing the mesh count would decrease the Oculus Rift performance, and including more models implied obtaining more models and using UV mappings, even though this particular issue was ignored. Also the focus of the project was to evaluate the usage of virtual reality by using the Oculus Rift as well as the overall acceptance among potential users, so adding more objects to the scene was not as relevant.

Another interesting subject to further explore was the material editor on the Unreal Engine 4, and its effects on virtual reality, which only the surface was explored on this project, and the issue with the Oculus Rift, that was solved by lowering the quality of the scene, had a negative impact on the advanced materials as features such as the bump map were barely noticeable after the screen percentage was lowered.

The sound in particular was one of the features that was neglected the most, the project had a background sound that was useful when the user was outside the house since it sounded as if they were on a garden, nonetheless small things as steps on different surfaces, screeching doors, or the noises a normal home produces would highly increase the immersion felt by individual users.

One of the biggest and most important future research regarding this project and any other virtual reality project would be an intuitive movement scheme; people struggled with moving around the project, multiple solutions were envisioned, the use of mouse and keyboard, a controller in conjunction with the head track of the HMD and even the controller alone to handle the movement and head track, none of

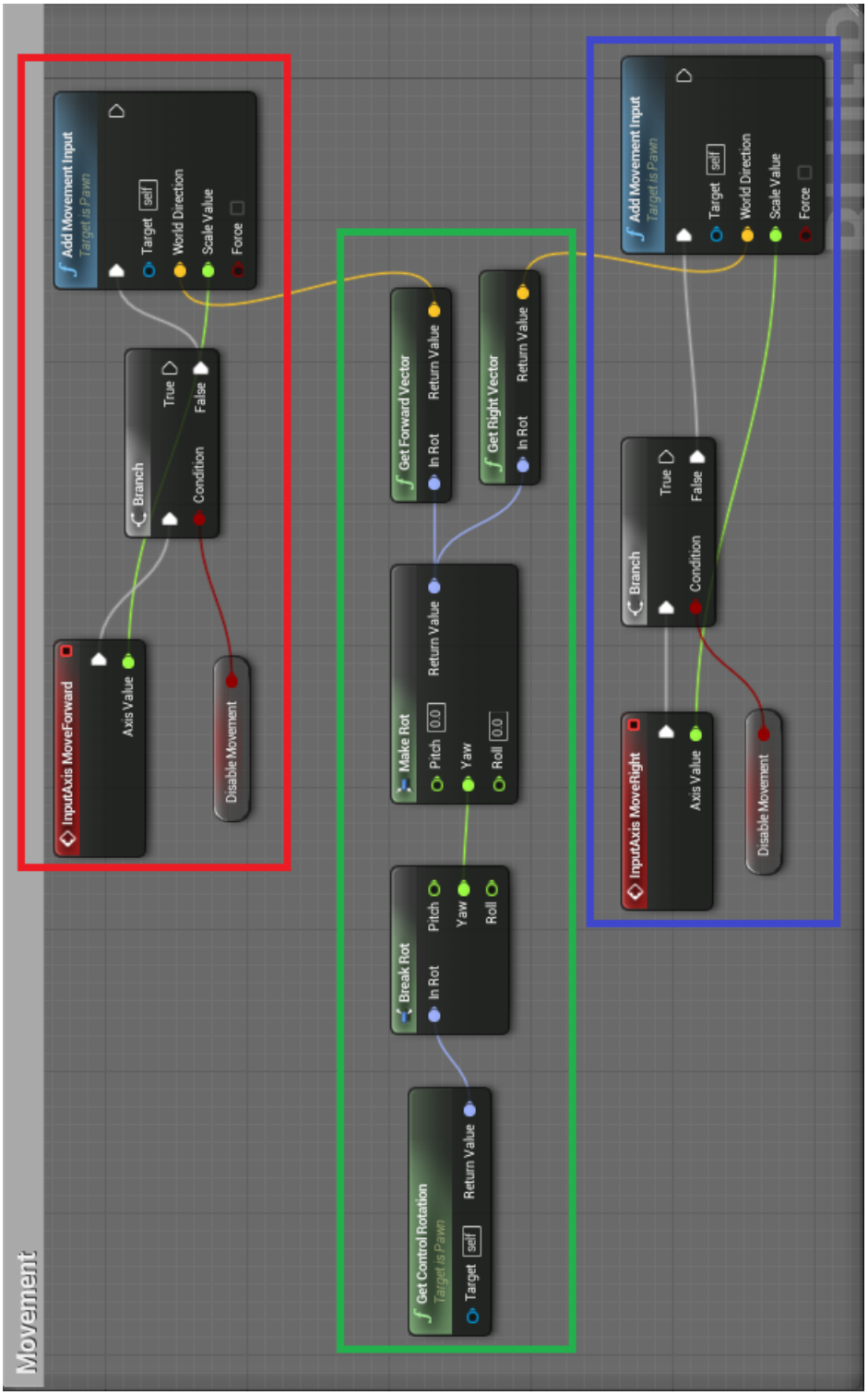
them were completely satisfactory as people still got confused. The obvious answer would be to use the Oculus Rift and a camera like the one used with Microsoft's Kinect to track the movement of people, the biggest issue with this is the wiring, as the wiring is necessary to reduce delay and other interference wireless protocols have, which make the display useless as it can display in real-time the user's movement.

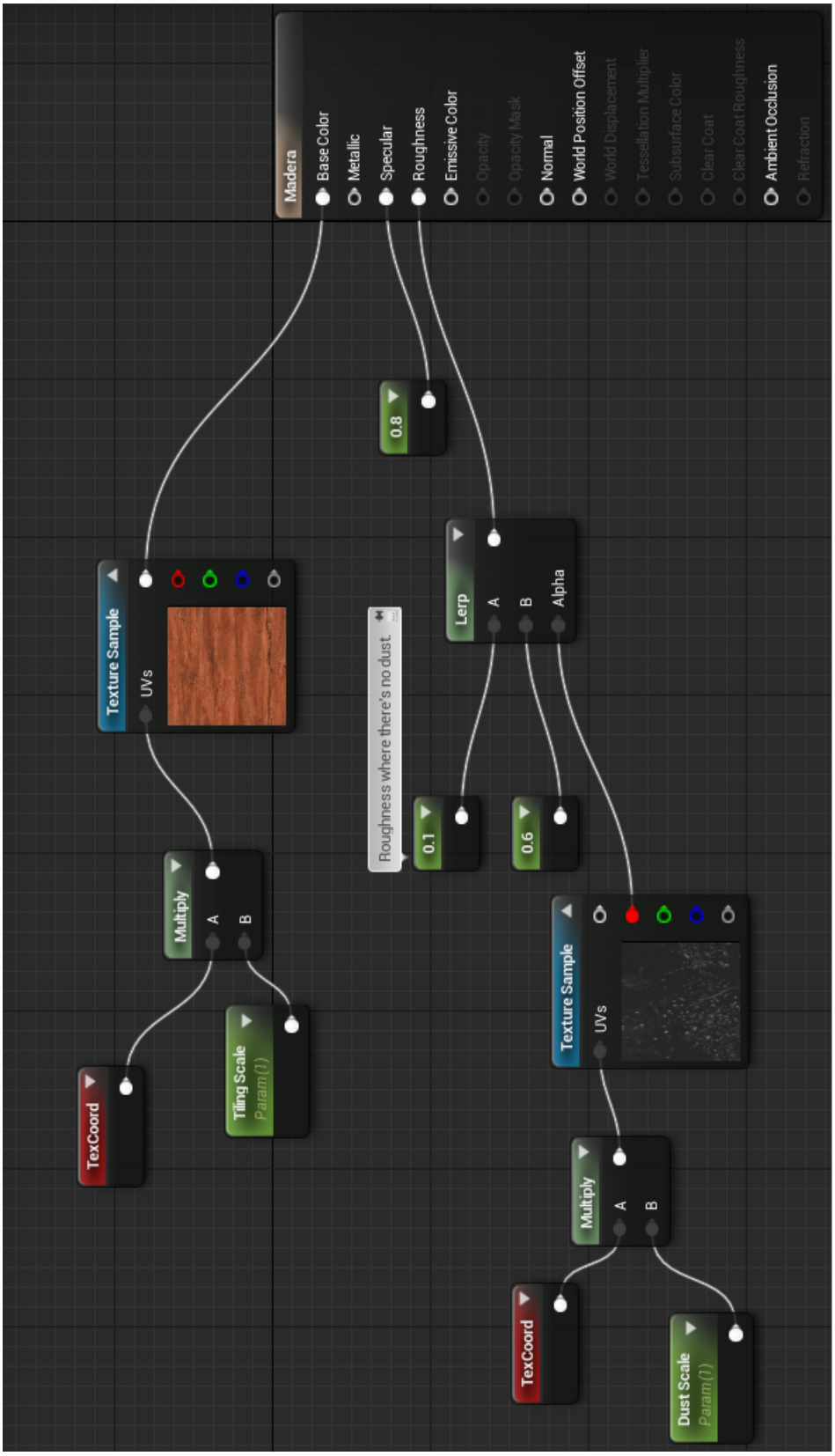
Lastly, one of the most requested features and one that was not added was interaction with elements, opening doors or windows would prove to be pretty easy on the programming side of the project and it would have a positive impact on the user's immersion. Consequently, having the user pick up small objects such as cups or plates would help them feel as if they had more influence over the virtual environment.

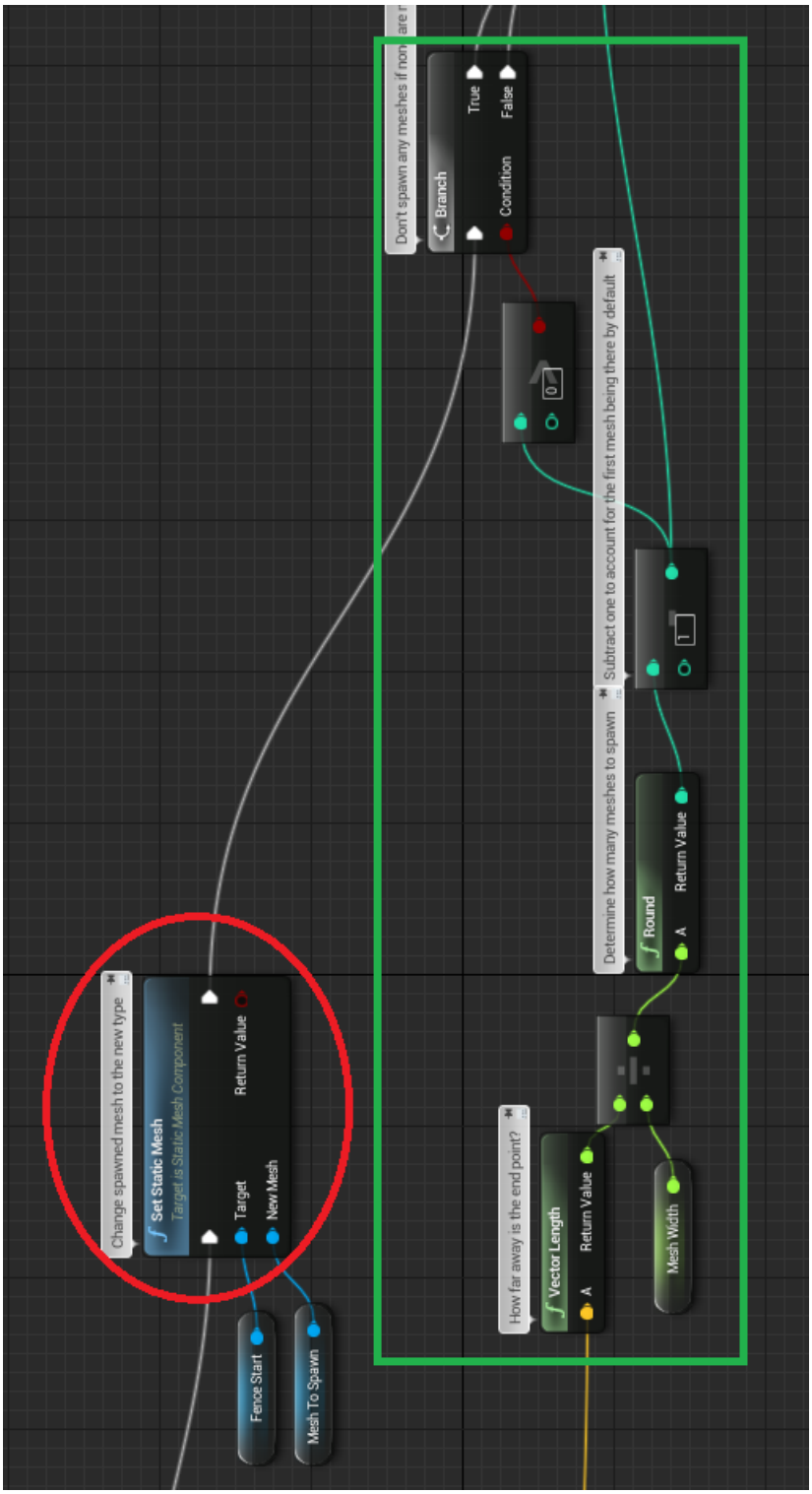
Part I

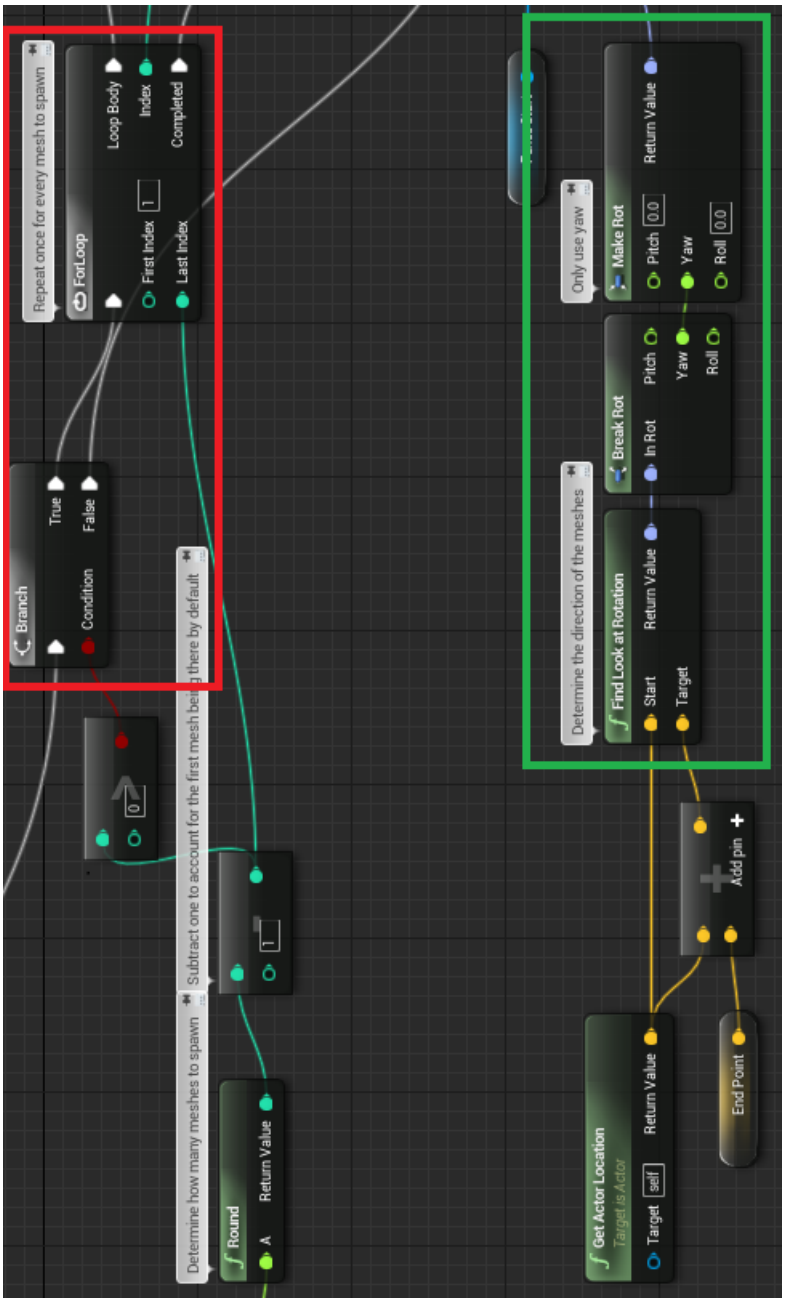
APPENDIX

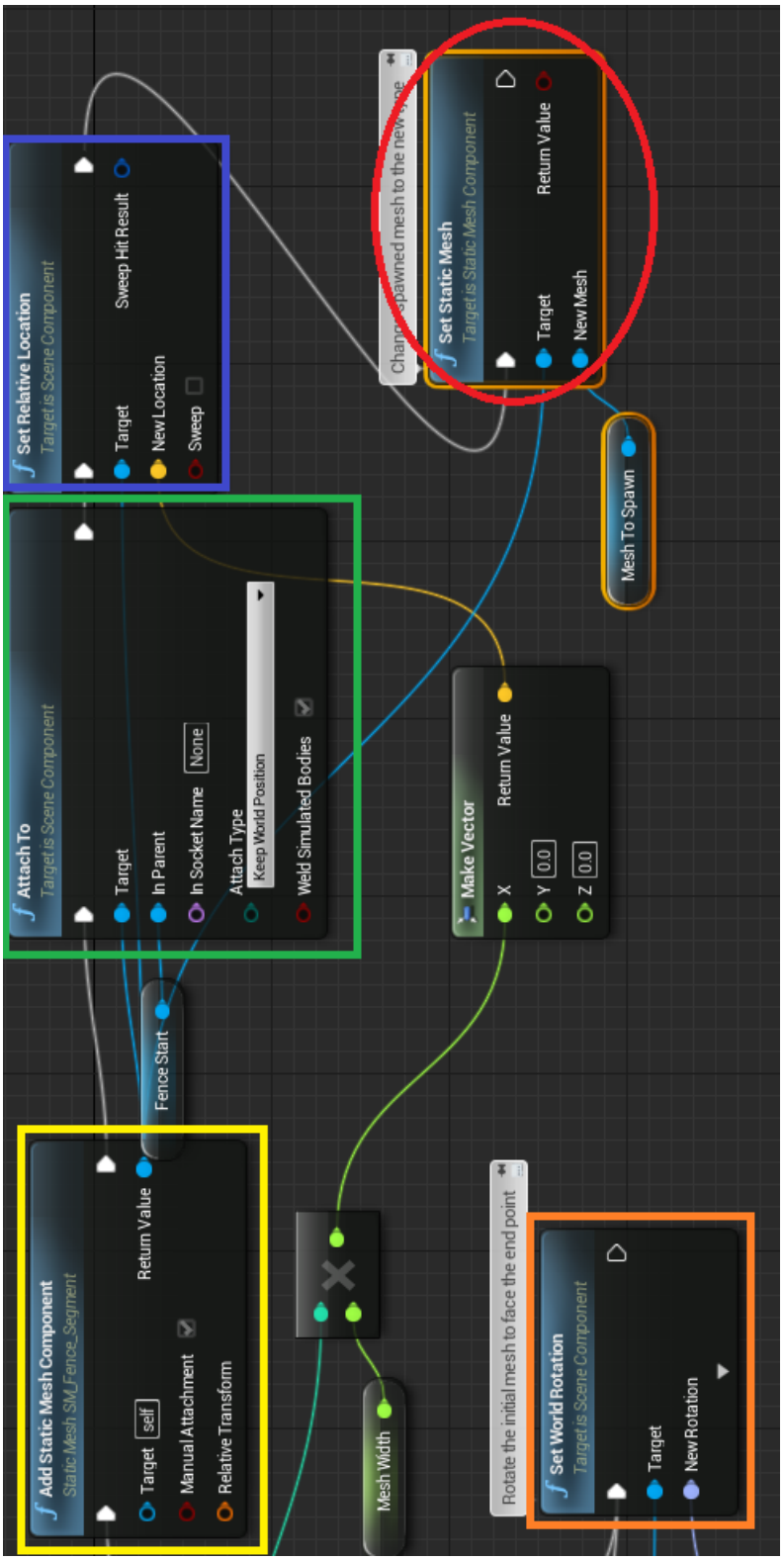
APPENDIX


Movement
InputAxis MoveForward
Axis Value
Add Movement Input
Target is Pawn
Target
self
World Direction
Scale Value
Force
Branch
True
False
Condition
Disable Movement
Get Forward Vector
In Rot
Return Value
Get Right Vector
In Rot
Return Value
Make Rot
Pitch
0.0
Yaw
Roll
0.0
Return Value
Break Rot
In Rot
Pitch
Yaw
Roll
Get Control Rotation
Target is Pawn
Target
self
Return Value
Add Movement Input
Target is Pawn
Target
self
World Direction
Scale Value
Force
Branch
True
False
Condition
InputAxis MoveRight
Axis Value
Disable Movement


Madera
Base Color
Metallic
Specular
Roughness
Emissive Color
Opacity
Opacity Mask
Normal
World Position Offset
World Displacement
Tessellation Multiplier
Subsurface Color
Clear Coat
Clear Coat Roughness
Ambient Occlusion
Refraction
0.8
Texture Sample
UVs
Lerp
A
B
Alpha
Roughness where there's no dust
0.1
0.6
Multiply
A
B
TexCoord
Tiling Scale
Param(1)
Texture Sample
UVs
Multiply
A
B
TexCoord
Dust Scale
Param(1)


Don't spawn any meshes if non
Branch
True
False
Condition
Subtract one to account for the first mesh being there by default
Determine how many meshes to spawn
Round
Return Value
A
How far away is the end point?
Vector Length
Return Value
A
Mesh Width
Change spawned mesh to the new type
Set Static Mesh
Target is Static Mesh Component
Target
New Mesh
Return Value
Fence Start
Mesh To Spawn


Repeat once for every mesh to spawn
ForLoop
Loop Body
Index
Completed
First Index
Last Index
Branch
True
False
Condition
Subtract one to account for the first mesh being there by default
Determine how many meshes to spawn
Round
A
Return Value
Target
Get Actor Location
Target is Actor
Return Value
self
End Point
Add pin
Determine the direction of the meshes
Find Look at Rotation
Start
Target
Return Value
Break Rot
In Rot
Pitch
Yaw
Roll
Only use yaw
Make Rot
Pitch
Yaw
Roll
Return Value


Set Relative Location
Target is Scene Component
Target
New Location
Sweep
Sweep Hit Result
Attach To
Target is Scene Component
Target
In Parent
In Socket Name
None
Attach Type
Keep World Position
Weld Simulated Bodies
Add Static Mesh Component
Static Mesh SM_Fence_Segment
Target
self
Manual Attachment
Relative Transform
Return Value
Fence Start
Mesh Width
Make Vector
X
Y
0.0
Z
0.0
Return Value
Set Static Mesh
Target is Static Mesh Component
Target
New Mesh
Return Value
Mesh To Spawn
Rotate the initial mesh to face the end point
Set World Rotation
Target is Scene Component
Target
New Rotation

BIBLIOGRAPHY

[1] CrazyBump normal mapping. http://www.crazybump.com/. Accessed: 2015-06-23.

[2] PlayStation project morpheus. http://blog.eu.playstation.com/2015/03/03/gdc-2015-project-morpheus-update/. Accessed: 2015-06-23.

[3] Oculus VR development kit 2. https://www.oculus.com/en-us/dk2/. Accessed: 2015-06-23.

[4] Unreal Engine 4: particle systems. https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/index.html, . Accessed: 2015-08-23.

[5] Unreal Engine 4: virtual reality best pratices. https://docs.unrealengine.com/latest/INT/Platforms/VR/ContentSetup/index.html, . Accessed: 2015-09-10.

[6] Sketch Up help. http://help.sketchup.com/en/article/36260. Accessed: 2015-06-23.

[7] Unreal Engine 4 lighting and shadows. https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/index.html. Accessed: 2015-09-20.

[8] Alan Chalmers and Andrej Ferko. Levels of realism: From virtual reality to real virtuality. In *Proceedings of the 24th Spring Conference on Computer Graphics*, SCCG '08, pages 19–25, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-957-2. doi: 10.1145/1921264.1921272. URL http://doi.acm.org/10.1145/1921264.1921272.

[9] Kevin Cheng and Paul A. Cairns. Behaviour, realism and immersion in games. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1272–1275, New York, NY, USA, 2005. ACM. ISBN 1-59593-002-7. doi: 10.1145/1056808.1056894. URL http://doi.acm.org/10.1145/1056808.1056894.

[10] Wikimedia Commons. Yaw motion in an aircraft. https://upload.wikimedia.org/wikipedia/commons/5/54/Flight_dynamics_with_text.png, 2007. Accessed on 04-04-15.

[11] Wikimedia Commons. Play station 4. https://en.wikipedia.org/wiki/PlayStation_4, 2013. Accessed on 04-04-15.

[12] Wikimedia Commons. Normal mapping. https://commons.wikimedia.org/wiki/File:Normal_map_example_with_scene_and_result.png, 2013. Accessed on 04-04-15.

[13] Mohamed Elhelw, Marios Nicolaou, Adrian Chung, Guang-Zhong Yang, and M. Stella Atkins. A gaze-based study for investigating the perception of visual realism in simulated scenes. *ACM Trans. Appl. Percept.*, 5(1):3:1–3:20, January 2008. ISSN 1544-3558. doi: 10.1145/1279640.1279643. URL http://doi.acm.org/10.1145/1279640.1279643.

[14] James D. Foley, Richard L. Phillips, John F. Hughes, Andries van Dam, and Steven K. Feiner. *Introduction to Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994. ISBN 0201609215.

[15] Diego Gutierrez, Veronica Sundstedt, Fermin Gomez, and Alan Chalmers. Modeling light scattering for virtual heritage. *J. Comput. Cult. Herit.*, 1(2):8:1–8:15, November 2008. ISSN 1556-4673. doi: 10.1145/1434763.1434765. URL http://doi.acm.org/10.1145/1434763.1434765.

[16] Donald E. Knuth. Computer Programming as an Art. *Communications of the ACM*, 17(12):667–673, December 1974.

[17] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 313–324, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: 10.1145/237170.237270. URL http://doi.acm.org/10.1145/237170.237270.

[18] Hristo Lesev. Classification of global illumination algorithms. *University Press "Paisii Hilendarski", Plovdiv*, .:., 2010.

[19] Hector E. Medellin. Transformaciones en 3d. http://galia.fc.uaslp.mx/~medellin/Applets/Trans3D/transformaciones_en_3d.htm, 2007. Accessed on 04-06-15.

[20] Alessandro Mulloni, Daniele Nadalutti, and Luca Chittaro. Interactive walkthrough of large 3d models of buildings on mobile devices. In *Proceedings of the Twelfth International Conference on 3D Web Technology*, Web3D '07, pages 17–25, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-652-3. doi: 10.1145/1229390.1229393. URL http://doi.acm.org/10.1145/1229390.1229393.

[21] G. Papaioannou. Real-time diffuse global illumination using radiance hints. *High performance graphics 2011*, .:., 2011.

[22] Gabriel Peyre and Laurent Cohen. Geodesic computations on 3d meshes. http://www.cmap.polytechnique.fr/~peyre/images/test_remeshing.jpg, 2006. Accessed on 04-04-15.

[23] Eva Pietroni and Andrea Adami. Interacting with virtual reconstructions in museums: The etruscanning project. *J. Comput. Cult. Herit.*, 7(2):9:1–9:29, June 2014. ISSN 1556-4673. doi: 10.1145/2611375. URL http://doi.acm.org/10.1145/2611375.

[24] James Ferwerda Program and James A. Ferwerda. Three varieties of realism in computer graphics. In *In Proceedings SPIE Human Vision and Electronic Imaging '03*, pages 290–297, 2003.

[25] PS4Daily. Project morpheus vs htc vive vs oculus rift. http://ps4daily.com/2015/03/project-morpheus-vs-htc-vive-vs-oculus-rift/, 2015. Accessed on 04-04-15.

[26] J. Sans. F.; Esmitt Ramirez. Real-time diffuse global illumination based on voxelization. *Computing Conference*, .:1,12,7–11, 2013.

[27] Clifford So, George Baciu, and Hanqiu Sun. Reconstruction of 3d virtual buildings from 2d architectural floor plans. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '98, pages 17–23, New York, NY, USA, 1998. ACM. ISBN 1-58113-019-8. doi: 10.1145/293701.293704. URL http://doi.acm.org/10.1145/293701.293704.

[28] Robert F. Tobler. Radiosity - ray tracing. https://www.cg.tuwien.ac.at/research/rendering/rays-radio/, 1997. Accessed on 04-04-15.

[29] Oculus VR. Oculus rift dk2. https://www1.oculus.com/order/, 2014. Accessed on 04-04-15.