SAVONIA

THESIS – BACHELOR'S DEGREE PROGRAMME
TECHNOLOGY, COMMUNICATION AND TRANSPORT

INTERACTIVE ARCHITECTURAL VISUALIZATION USING UNREAL ENGINE

Creation of a Virtual Model

AUTHOR: Jaakko Susi

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Construction Architecture	
Author Jaakko Susi	
Title of Thesis Interactive Architectural Visualization Using Unreal Engine	
Date 3 April 2022	Pages/Appendices 40/7
Client Organisation /Partners Treedee Oy	

Abstract

The aim of the thesis was to map the creation and workflow of a playable, photorealistic, and interactive virtual model inside a game engine, Unreal Engine. Another goal was to produce a playable game for showcasing the exterior and the interior of a building. The project was commissioned by Treedee Oy and the building used in the final product was designed by them. The building is planned to be built in Hainan, China.

The theoretical part of the thesis focused on the topics and themes that had to be dealt with in order to produce a playable virtual model. Explanations of the topics and short instructions on how to use the basic tools of Unreal Engine were also included in the theoretical part. To better understand the software and the related topics, a virtual model was created. Apart from the base model, everything else was entirely done inside Unreal Engine.

As a result, the final product is a photorealistic, playable desktop version of a building with interactive elements. The interactive elements contain material and furniture changing during runtime as well as showing object info on hover-over of the mouse. Interactivity also included menus and user interface. The theoretical part of the thesis has instructions on how to use Unreal Engine for architectural demonstration and visualization purposes. The thesis was mostly written in the format of a user manual. Part of the thesis is confidential to the client and thus not included in the public version.

Keywords
Savonia University of Applied Sciences, construction architecture, Unreal Engine, virtual model, interactivity

PREFACE

This thesis and the virtual model related to it was an over a year long journey for me. I have learned an immense amount of information about gamification and creation of a virtual building model.

I want to thank Mika Leinonen from Treedee Oy for the commission and support throughout the project and Viljo Kuusela and Janne Repo for help, support and teaching. Also, a special thanks belongs to my family.

Tampere April 3, 2022
Jaakko Susi

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS AND GLOSSARY

UE4 = Unreal Engine 4

Unreal = Unreal Engine 4

PBR = Physically Based Rendering

AO = Ambient Occlusion

HDR(I) = High Dynamic Range (Imaging)

GI = Global Illumination

UV mapping = The process of displaying a 2D image on a 3D surface

Ray Tracing = The process of simulation and tracking a ray of light produced by a source of lighting

RTGI = Real Time Ray Tracing

1 INTRODUCTION

The project was introduced to me by Mika Leinonen from Treedee Oy. Treedee was looking for someone to map the workflow of creating a playable, interactive virtual model inside a game engine, Unreal Engine more specifically. I had been studying architectural visualization in offline-rendering engines for two years prior, so a project related to a real-time game engine was something I was really excited about.

Originally the assignment included creating an interactive VR-model version inside Unreal Engine. However, realizing how it took me nearly a thousand hours of studying, experimenting, and working on the model just to create a desktop version, the requirement for a VR-model version of the game was abandoned. And so, the result of the project is a playable desktop-version of the model.

Interactivity included elements such as changing materials and furniture during runtime and showing object info on mouse hover-over. Reaching photorealism was also one of the goals, which was achieved. The game was created using Unreal Engine version 4.27.

This thesis is for people who want to start learning about gamification of BIM-models inside Unreal Engine. The explanations of the concepts and instructions to use them are somewhat simplified. The thesis serves as a starting point for architecture offices to be able to create a basic scene and a game.

2 USES OF ARCHITECTURAL VISUALIZATION

In architecture, visualization presents the building in a way that can be easily grasped. It involves drawing or producing an image or a drawing that can be easily understood by the client. The main goal of architectural visualization is to illustrate and demonstrate how a building looks like before it is built.

3D-rendering has become an integral part of many industries. From retail to manufacturers, everyone is using 3D-rendering and visualization to improve their business. 3D-models and visualizations are often the most effective way to convey ideas to the client and users. The buildings are usually modeled utilizing BIM (Building Information Modeling) and therefore are also available as a 3D-model. A single 3D-model of the building and a rendering software can be used to practically produce an infinite number of interior and exterior visualization images, animations and a virtual reality setting from different angles and places the building.

FIGURE 1. A screenshot of an interior created in Unreal Engine (Susi 2021, CC BY-NC-ND)

The most immersive way a client or user can experience a design is through a game which can be played either on a desktop or in VR with a virtual reality headset, such as Oculus Rift. During the last several years, companies have come up with tools for real-time visualization. Software has been used to create videogames for decades but only in the recent years, architecture industry and designers have been able to use this technology to convey their ideas more easily.

2.1 Architectural visualization as a communication tool

Through architectural visualization, clients and architects can easily communicate about the details of a project. People who are not skilled or competent in interpreting architectural drawings and designs will not be able to communicate effectively with the architects. Although the use of mental imagery is beneficial, it should be kept at its minimum to avoid overloading the information receiver. Also, there is a risk for communication errors since both parties are creating different mental images. Lastly, visual presentation, such as a virtual model of the building or rendered images, can help reduce the amount of work involved in analyzing the data and decision making.

2.2 Architectural visualization as a marketing tool

FIGURE 2. Virtual Helsinki made by ZOAN in Unreal Engine (ZOAN 2020)

High-quality marketing visualizations are created to spark an emotional response to an image. They are typically made to a specific audience and are not intended to be accurately portray reality.

Animations are also a powerful way to show the building and architecture thoroughly. The main benefit of an animation is that it allows the viewer to visualize the full story of a project. However, they may not see all the details that the architect has paid attention to.

Already decades ago, it was envisioned that a real estate broker would be able to show their clients properties from the broker's office. This is now reality. In addition, it is now possible to show the building in a virtual reality setting before the building is built. At this point of the design, clients can easily affect if they want, for example, a part of the design changed.

The most powerful marketing tool is a gamified building model. This thesis walks the reader through the workflow of creating a simple game.

2.3 Unreal Engine

There are programs which allow the user to fly or walk through the building with a single click from the 3D-model- or BIM-software. And with a single click, the building is gamified. These simpler gamification programs have little flexibility when it comes to how much you can do inside the software. This is where Unreal Engine steps in.

Made by Epic Games, Unreal Engine is a powerful game development tool that enables developers to create 3D models, animations, linear film and television, training and simulation and architectural visualization.

Unreal Engine was first showcased in 1998. It has since been used to develop games in various genres, for example, first-person shooters, platformers and open-world games. The engine is written in C++, which supports a wide range of platforms.

Unreal Engine 5 is scheduled for release in 2022.

This thesis works as a starting point for people who want to learn Unreal Engine in architectural visualization and gamification.

3 ACHIEVING PHOTOREALISM IN UNREAL ENGINE

The creation of a gamified building model inside Unreal Engine starts with the preparation of the base model and geometry. After the preparation is done, the model is imported into Unreal Engine.

The most crucial components of a photorealistic scene in Unreal Engine are correctly made materials and lighting. Lighting in particular is something that can make or break the whole scene. The thesis focuses mainly on these topics.

Finally, if desired, interactivity can be built and programmed into the game.

3.1 Base model and geometry

Treedee Oy provided a Revit model which would act as a starting point inside Unreal Engine.

The 3D model was modeled in Autodesk Revit, version 2020.

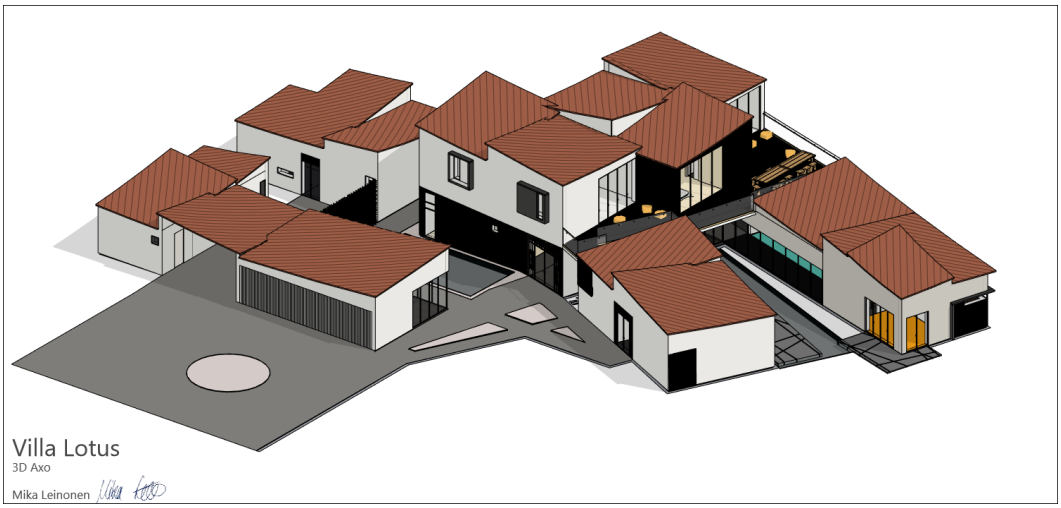

FIGURE 3. A screenshot of the base model and geometry inside Revit (Treedee Oy 2021)

A few modifications were made to the Revit model to prepare it for UE4. The surfaces that were painted in Revit are considered as their own sub-surfaces inside the mesh. This means that some of the interior surfaces had to be painted with different materials to allow certain materials to be applied to these surfaces inside Unreal Engine.

A bigger change or addition that had to be made was a ceiling so that there would be a single, unified surface in the first floor inside Unreal.

3.2 Importing models

The way Unreal handles imported objects and meshes is the following: when, for example, an fbx- or an obj-formatted file is imported into Unreal, Unreal will convert the file into uasset-format and save it automatically into the project's Content folder. Unreal will do this to all filetypes that are imported into the project.

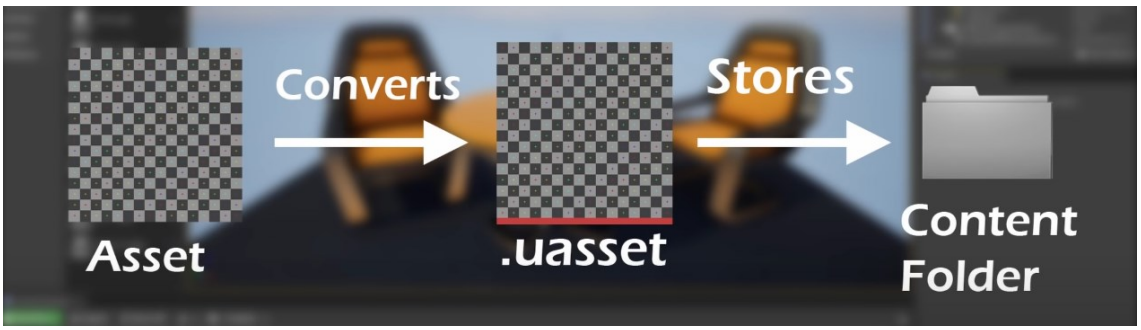

FIGURE 4. Asset migration to Unreal Engine's Content Folder (Unreal Engine documentation 2021)

The easiest way to import files into Unreal is to drag and drop them into the projects' Content folder.

Drop the file into a wanted location and folder inside the Content Browser.

When importing an fbx-file, Unreal will ask about the file's import options such as the materials and transform options. Usually, the default options are fine. Generally, you also want to make sure Skeletal Mesh is turned off.

3.2.1 Datasmith import

The most effective way to import a Revit-model to Unreal is through a tool called Datasmith.

Datasmith is a collection of tools which helps you bring content into Unreal Engine 4 from other software. It works seamlessly across various industries such as architecture, engineering, and construction. In short, it is an importing tool for bringing the data from an external design software into Unreal Engine.

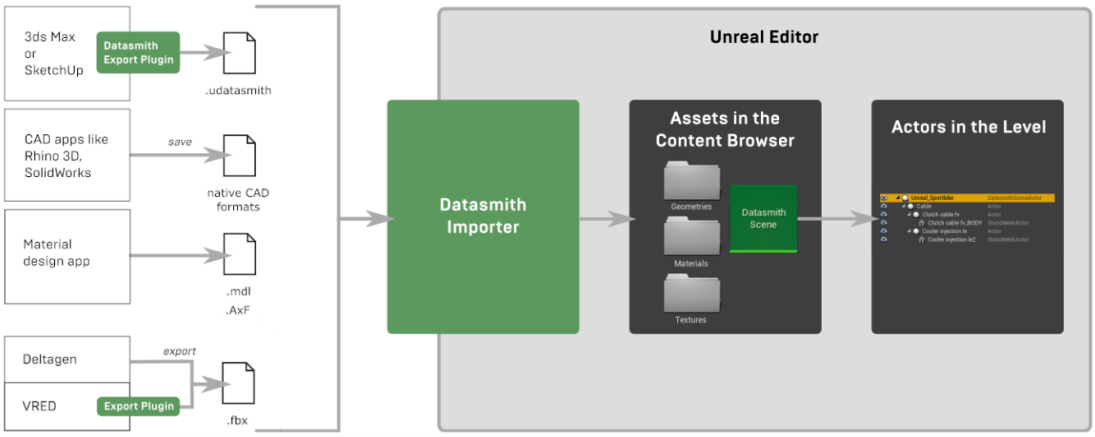

FIGURE 5. Datasmith import process (Unreal Engine documentation 2021)

Under Edit, select Plugins and enable Datasmith Plugin. Immediately after enabling it, Unreal will ask you to restart the engine in order to turn the plugin on. Do so.

After Unreal restarts, Datasmith button will appear to the ribbon. Click on Datasmith and find the udatasmith file you exported from Revit.

By default, under Process, everything is turned on. Min Lightmap Resolution should generally be half of what the Max Lightmap Resolution is. Set Min Lightmap Resolution to 256 and leave the Max Lightmap Resolution to 512. It can be increased later for individual Static Meshes.

Click Import.

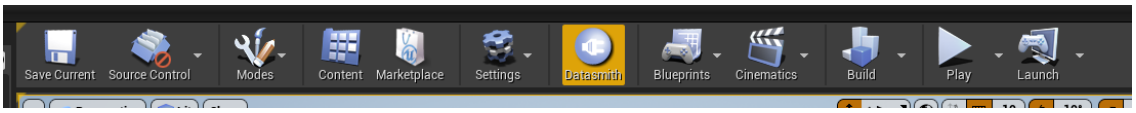

FIGURE 6. Datasmith button in the Ribbon (Susi 2021, CC BY-NC-ND)

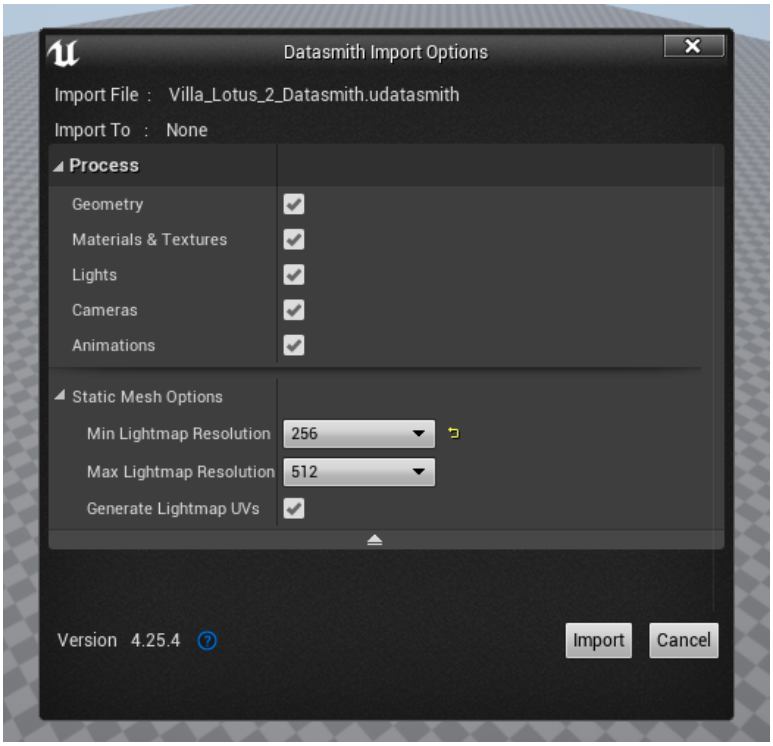

FIGURE 7. Datasmith Import Options (Susi 2021, CC BY-NC-ND)

3.3 Actors, geometry, and UV mapping

After importing the geometry, it is time to add additional geometry, such as furniture, to the model.

An Actor is an object that can be put into a Level, such as Static Meshes or lights. It should also be noted that Actors have a Mobility setting available. The setting controls whether the Actor will be allowed to be moved or changed in a certain way. Usually, this setting applies to Light Actors and Static Mesh Actors. There are three different states in the Mobility settings:

Static: The Actor is not intended to be moved or changed in any way during gameplay.

Stationary: The Actor is not intended to be moved but can be changed during gameplay.

Movable: The Actor is completely mobile. The Actor can be moved, changed or removed during gameplay.

To place an Actor into the Level, in the Place Actors panel, drag and drop an Actor of your choosing into the scene.

A UV Channel controls how a 2D texture is wrapped around a 3D object.

There are two purposes for UV Channels inside Unreal Engine:

When creating a Material that contains a texture map, you can specify a UV Channel to describe the area of the texture that should be used to shade the 3D model.

A UV channel is also used to store and apply Lightmaps. These are special textures that are used to store and manage lighting information for Static Meshes.

Since objects receive varying amounts of lighting, each part in the mesh must have its own UV coordinate value. Also, each triangle has to cover its own area in the texture.

The easiest way to handle UV mapping when importing geometry into Unreal Engine is to use Datasmith. Datasmith handles and automates the UV mapping during the import.

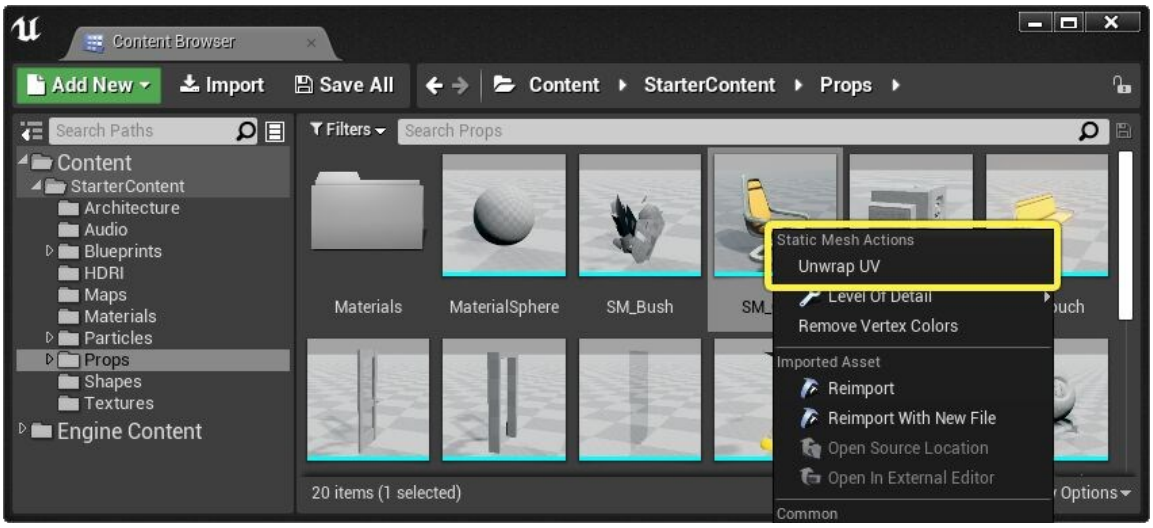

FIGURE 8. Unwrapping Mesh Geometry in Content Browser (Unreal Engine documentation 2021)

You can unwrap the UVs directly from Content Browser as shown in FIGURE 7. It is also possible to do it inside a Static Mesh Editor. Double click the mesh in the Content Browser. In the Ribbon/Toolbar, find select UV > Unwrap UVs as shown in FIGURE 8.

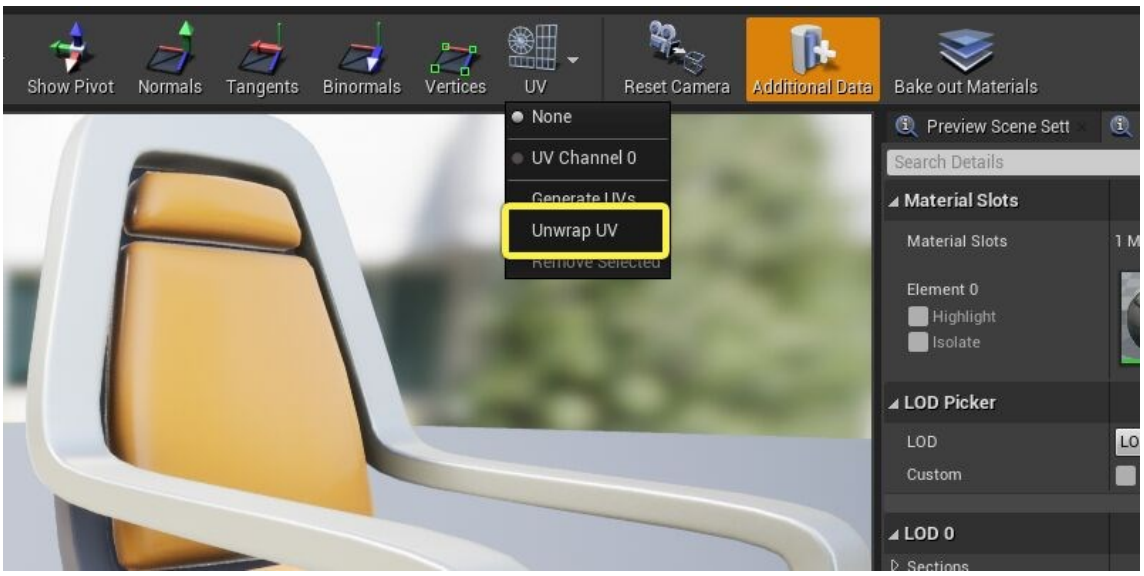

FIGURE 9. Unwrapping Mesh Geomery in Static Mesh Editor (Unreal Engine documentation 2021)

The 3D geometry of a specific Static Mesh can be projected into 2D texture space.

In the Toolbar of a Static Mesh Editor, select UV > Generate UVs.

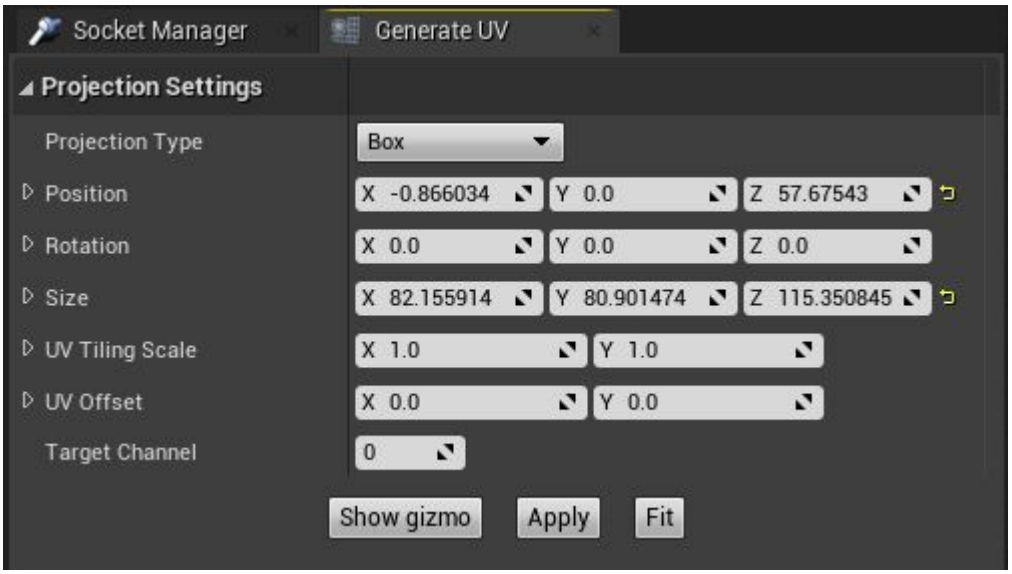

FIGURE 10. Generate UVs (Susi 2021, CC BY-NC-ND)

3.4 Materials

After all the desired geometry is placed into the scene, it is time to assign materials.

A Material in UE4 and other visualization programs can be used to control the visual appearance of an object. It is usually assumed that a material is the paint that is applied to an object.

In technical terms, a material is used to calculate the interaction between light and an object. This procedure is performed using incoming data that includes various math expressions and images.

3.4.1 Master material and material instancing

In Unreal, Materials are created inside the Graph Editor. The nodes are used to express different properties of a Material. These nodes are then connected to the Main Material node. The results of what is connected into the Main Material node is what is displayed when the Material is placed to meshes.

Unreal Engine makes use of a Master Material system. This enables the user to efficiently create essentially different materials with the Master Material's properties and parameters. In short, a Master Material is a material that is parameterized and then used to create Material Instances. Also, parameters can be added to the Master Material later, and the parameters will show in the Material Instances created from the Master Material in question.

It is recommended that Materials or similarly Master Materials are used as sparingly as possible. Instead, Material Instances should be used whenever possible.

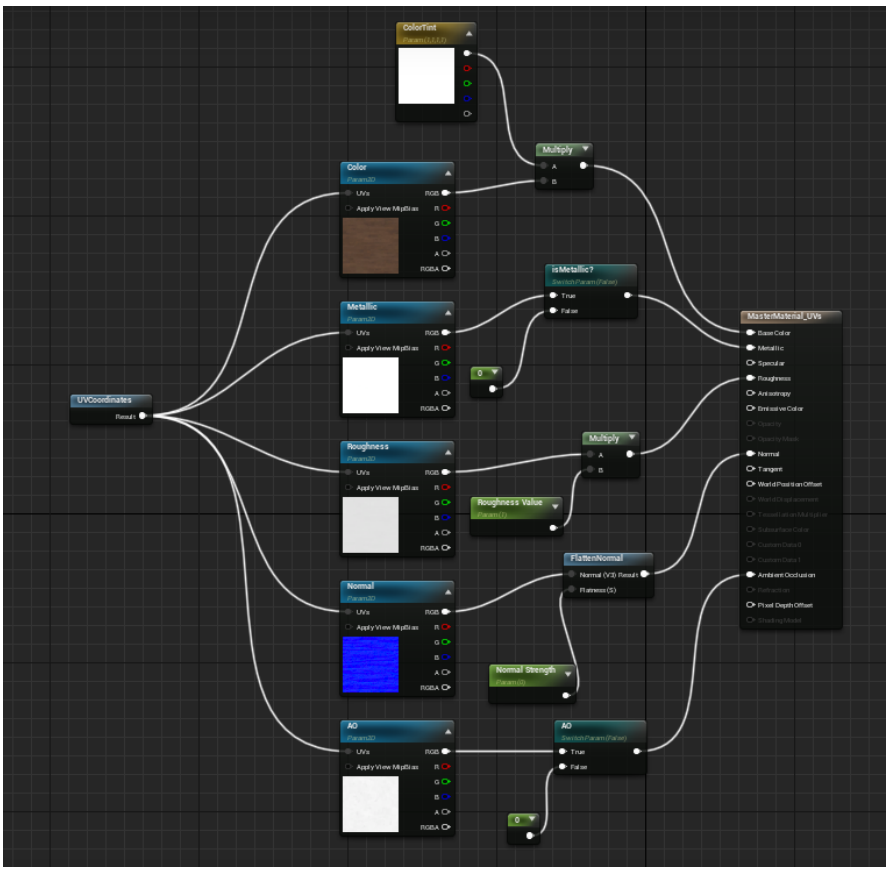

FIGURE 11. Master Material inside the Graph Editor (Susi 2021, CC BY-NC-ND)

3.4.2 Physically Based Rendering (PBR) in materials

Instead of trying to intuitively approximate what the light does, physically based shading allows us to approximate what the light actually does. This process produces a more natural looking and accurate rendering. This method is called Physically Based Rendering. The materials which are done using this method will work well in all lighting scenarios and environments.

3.4.2.1 PBR-material parameters

Base Color

Base Color is the overall color of a material, which is defined by a Vector3 value (RGB). It can be defined by taking in a color value of 0 to 1.

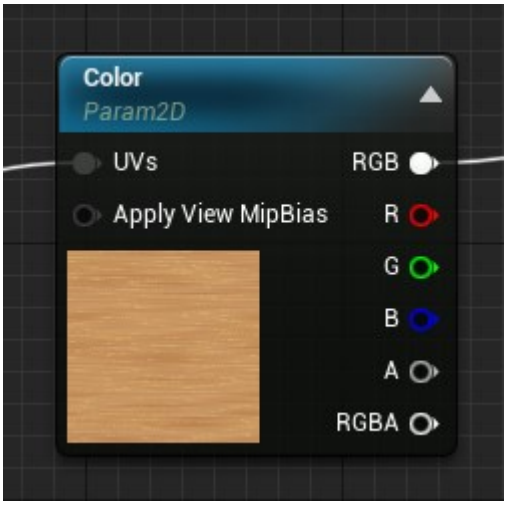

FIGURE 12. Parameterized Base Color node inside the Graph Editor (Susi 2021, CC BY-NC-ND)

Roughness

The Roughness input controls how rough the material's surface will look. Rougher materials, for example, a brick, will not reflect light and therefore the value should be closer to 1. For smooth materials, for example, marble, the value should be 0.

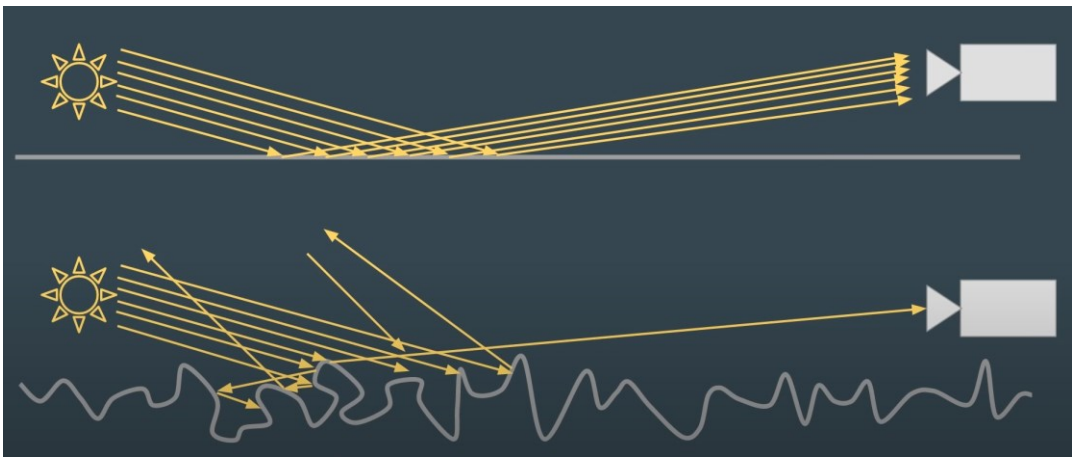

FIGURE 13. Blender Guru Roughness demonstration (Blender Guru 2016)

Metallic

The Metallic input controls how metal-like your surface will look. For non-metals, the value is 0. For pure metal surfaces, the value is 1. This value will always be either 0 or 1. In hybrid surfaces, for example, rusty metals, a value between 0 and 1 might be right.

Specular

The Specular property and value control the material's ability to reflect light. This property works only for non-metallic surfaces. The default value is 0.5, 0 being non-reflective and 1 being fully reflective.

Normal

The Normal input takes a normal map, which provides detailed information about the surface's facing direction. It coordinates the individual pixels with their normal faces.

The difference between a Normal Map and a Bump Map is that a Bump Map goes only in one direction. A Normal Map goes in two directions.

Ambient Occlusion (AO)

The Ambient Occlusion input serves as a component used to simulate the shadowing effect that occurs within the small cracks and crevices of a surface.

3.5 Lighting and illumination

Lighting is one of the most important things to get right in order to have a realistic scene. The most realistic lighting is achieved through built lighting from CPU-Based Lightmass or GPU-Based Lightmass. It is also possible to have a completely dynamic lighting, but it reduces the visual quality of the model. However, if dynamic lighting is chosen, there is no need to build lights and the time needed to see the changes in lighting is reduced to zero. All the changes appear instantly. In built lighting, every time a Static Mesh is moved, lighting needs to be rebuilt.

Actors that must be added to the scene to create basic lighting are as follows: Sky Atmosphere, Directional Light and Exponential Height Fog. Under DirectionalLight Actor's Details-panel, turn on Atmosphere/Fog Sun Light to bring light into the scene from the DirectionalLight Actor. To be noted, a Sky Light Actor gives the scene significantly more realistic shadows. And since the aforementioned light Actors are set to Static by default, lighting needs to be built. To build the lighting, click the arrow on the right side of Build on the Ribbon and select Build Lighting. Or alternatively, set the DirectionalLight Actor and Sky Light to Movable to avoid the need to build the lights. This reduces the lighting's visual quality but at the start of a project, it might be a good idea.

3.5.1 Global illumination

Global illumination simulates indirect lighting between materials. It uses the reflectiveness of the materials and geometry to create realistic lighting effects.

A photon is a particle that is responsible for the energy that is used to produce light. In a scene, a photon is responsible for producing the light that is in the render. When it encounters a surface, it carries its energy values and then continues to bounce off until it is absorbed.

There are two ways to light up 3d worlds inside Unreal Engine: real-time lighting methods can provide dynamic lighting effects, while precomputed lighting information is stored in textures which are applied to surfaces. These two ways can be combined and blended to optimize the lighting in the scene. They are not exclusive to one another.

Precomputed lighting in Unreal Engine is a feature that gives two different ways of accessing lighting data. It can be done on the GPU or the CPU. Precomputed lighting produces usually the highest quality results, but it cannot be changed dynamically.

3.5.1.1 CPU- and GPU-Based Lightmass

CPU-Based Lightmass

The CPU-Based Lightmass system utilizes a process called Unreal Swarm to generate and distribute lighting data. Due to the complexity of lightmap scenes, and the number of available threads, it is important that a build has multiple machines to generate the final results. If handled on a single machine, complex scenes will take a lot of time and power to complete.

GPU-Based Lightmass

The GPU-based Lightmass is a system that uses your machine's GPU with DirectX 12 and ray tracing capabilities to generate and store lighting data. It does not support distributed builds. A GPU-Based Lightmass is significantly faster to build and calculate than a CPU-Based Lightmass.

3.5.1.2 Screen Space Global Illumination

Screen Space Global Illumination is a lighting method that outputs dynamic lighting data from the camera's view. This method is limited by the objects and lighting that can be seen only within the

camera's view. SSGI is a good additive to be used in combination with existing Global Illumination methods.

3.5.1.3 Ray Tracing Global Illumination (RTGI)

Ray Tracing in Unreal Engine is a dynamically generated illumination method which utilizes Microsoft's DRX framework and NVIDIA™'s ray tracing capabilities to create lighting effects which are physically accurate.

Two methods of RTGI are supported:

Brute Force

The Brute Force is a method that emulates Path Tracer's method of rendering. It provides the most accurate global illumination, but in turn it is the most consuming for frame performance. At this point in time, even with a powerful PC and especially GPU, in most cases the frame performance is not enough to validate using this for real-time global illumination.

Final Gather

The Final Gather method uses a two-pass algorithm and a fixed number of samples for optimal performance. If Ray Tracing is utilized for a real-time project, Final Gather can be used and optimized to fit the needs.

To enable Ray Tracing, from the Main Menu find Edit menu to open the Project Settings. Under Platforms > Windows, press Default RHI dropdown and select DX12. Next, under Engine > Rendering, enable Ray Tracing.

3.5.2 Light types

As in Static Mesh Actors, light types also have mobility settings.

Static means that the light cannot be changed during runtime. Static Mobility setting is the fastest for rendering and it allows light baking. The Light properties menu offers tooltip-info when hovered over.

Stationary means that only the indirect illumination from the light source is baked into a Lightmass when you build the lighting. All other lighting is dynamic. The light's color or intensity can be changed during runtime, but it can't be moved.

Moveable means the light is completely mobile and dynamic. This method is also the slowest for rendering.

Directional Light

The Directional Light is commonly used as a direct sunlight or sun or as a primary light source.

Sky Light

Sky Light is used in conjunction with the Directional Light. Sky Light gives the scene ambient lighting. You can use the existing environment to be used with the Sky Light or you can use a HDRI. A HDRI cubemap gives more control over the settings.

Point Light

A Point Light is probably the most used light type in a scene. A Point Light could be compared to a real-world light bulb. It emits light in all directions equally from a single point.

Spot Light

A Spot Light emits light from a single point in a cone-like form.

Rect Light

A Rect Light or Rectangular Light emits light from a rectangular shape or plane. The dimensions of the plane can be modified.

The Light types used in this project were only Directional Light and Sky Light since the scene was rendered only to be during midday.

3.6 Reflections

The most common way to set up reflections in Unreal is through Screen Space Reflections (SSR). This effect is enabled by default. It is also the cheapest way for the engine to render the reflections. There are not many options or properties to change when it comes to SSR. The properties can be tweaked in the Post Process Volume's settings. By definition, Screen Space Reflections only reflect what is on your screen.

If higher quality reflections and more bounces are needed, it is possible to use Ray Traced Reflections (RTR). However, usually the performance and framerate will decrease when RTR is enabled.

3.6.1 Reflection Captures

Reflection Capture Actors are objects that are placed in the scene to provide data into the Reflection Environment. Currently, there are two types or shapes of reflection captures: a box and a sphere.

A Box Reflection Capture should only be used in rooms which are rectangular. Anywhere else where there are odd angles, a Sphere Reflection capture should be used.

For example in a room, Reflection Captures should be placed as shown in the picture. Smaller radius ones to capture the reflections in more specific areas inside the room and the bigger radius ones to capture the reflections on a larger scale. Unreal will then blend all the captures together. The more Reflection Captures there are overlapping in the level or scene, the more expensive it is to render the materials because the engine needs to average the reflections between all of them.

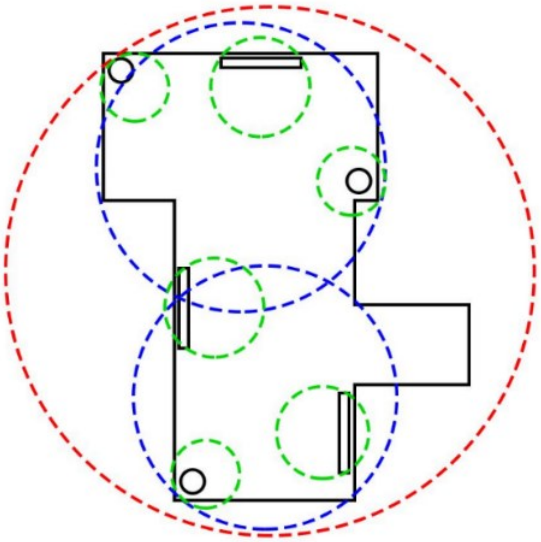

FIGURE 14. Reflection Capture Actors placement (Unreal Engine documentation 2021)

4 CREATING INTERACTIVITY AND VISUAL PROGRAMMING

As an architecture student, programming might be scary to start to learn. Unreal Engine uses C++ as a programming language, and it is considered to be in the harder end of the more common programming languages. Luckily, Unreal Engine provides Blueprint Visual Scripting system to enable the creation of gameplay elements. It makes the programming easier to do and understand. However, if you are already familiar with C++ or any other programming language, naturally Blueprints make the work considerably easier.

As a beginner in programming and Unreal Engine in general, it is recommended to start the project with a template that comes with a Player Controller and a Player Pawn. For example, first-person and third-person templates are templates that come with basic Player Controller and a Player Pawn.

4.1 Blueprints

The Blueprint Visual Scripting System uses Unreal Engine's Node-based interface to create game elements. It is commonly used to define object oriented classes and assets. The system allows designers to work with the tools and concepts typically used by programmers.

Blueprints are essentially visually scripted additions to the game, and they can be used to create interactive gameplay elements. They are graph-based constructs that can be used to implement various gameplay and object construction features.

Level Blueprint

Each level has its own Blueprint, which can be used to control cinematics, actors, and other level-related systems. This is what is called as a Level Blueprint. If and when the level has Blueprint Classes placed in it, these can be interacted with from the Level Blueprint. An example could be a custom trigger event.

Blueprint Class

Blueprint Classes are commonly used to create interactive assets such as doors and switches. They can also be used to make switchable objects, for example, furniture or floor materials.

Player Pawn

The first-person and third-person templates come with a Player Pawn, the mesh of the Player known as the SkeletalMeshComponent, and PlayerController.

Creating a new Character Blueprint allows you to customize the character’s behavior based on how you want it to be controlled and moved.

User Interface

Blueprints can also be used to create a user interface for a game, similar to how events and variables can be stored in a Blueprint Class. The user interface can be fully customized inside a widget.

4.2 PlayerPawn and PlayerController

A Pawn in Unreal Engine is a physical representation of a player. It can also be an AI entity withing the world. Since there is necessarily no need for multiplayer mode in architectural visualization and showcasing, the default Player Pawn that comes with the first-person and third-person templates can be used. The template's Player Pawn comes with a gun and shootable projectiles. Naturally, these must be removed from the Pawn.

There is also a connection between Pawns and Controllers. A PlayerController is the interface between the Pawn and the person controlling it.

SkeletalMeshComponent

Unlike Pawns, Character-derived classes have a SkeletalMeshComponent. This component can be used to enable advanced animations.

CapsuleComponent

The CapsuleComponent is used for a moving collision. It is assumed that the capsule is a vertically oriented structure. When testing the level you have created, there might be a case where the controllable player, for example, cannot fit through narrow places. In this case, the CapsuleComponent is what needs to be adjusted.

CharacterMovementComponent

The CharacterMovementComponent is used to allow players to move by swimming, running, jumping, and falling without the use of rigid body physics. It can also determine the speed at which a character can travel across land and air, and the gravity scale of their body.

4.3 Adjusting PlayerController

By default the character movement speed is too high to be used for architectural visualization purposes.

To change the character movement speed, open the FirstPersonCharacter Blueprint inside FirstPerson > FirstPersonBP > Blueprints-folder. On the left side of the screen under Components, you can find Character Movement (CharMoveComp) (Inherited). Clicking on it opens a details panel to the right side of the screen, where movement speed and other properties can be edited. If you want to edit only the movement speed, Max Walk Speed is the value you need to change.

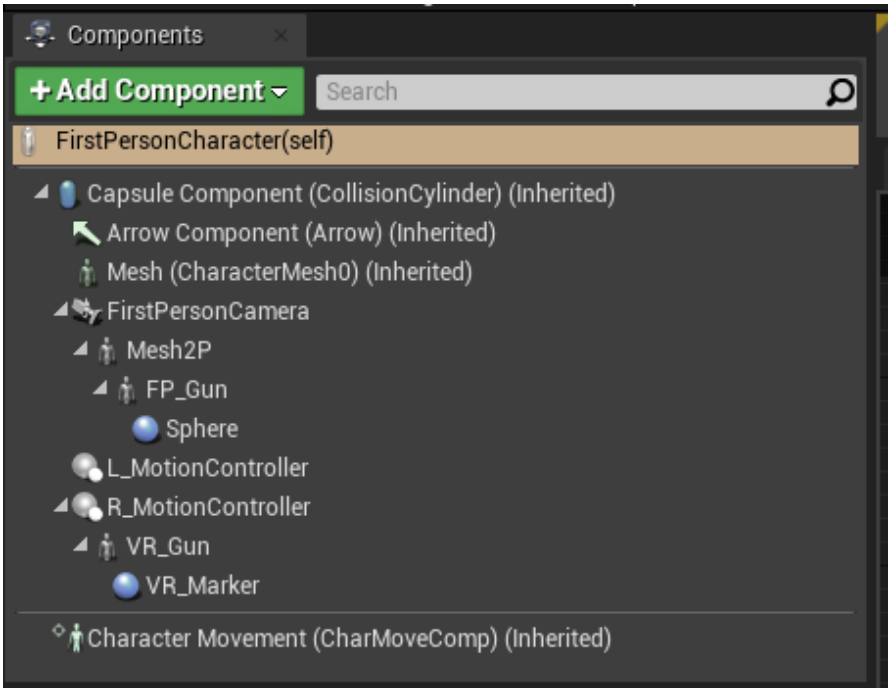

FIGURE 15. First Person Components (Susi 2021, CC BY-NC-ND)

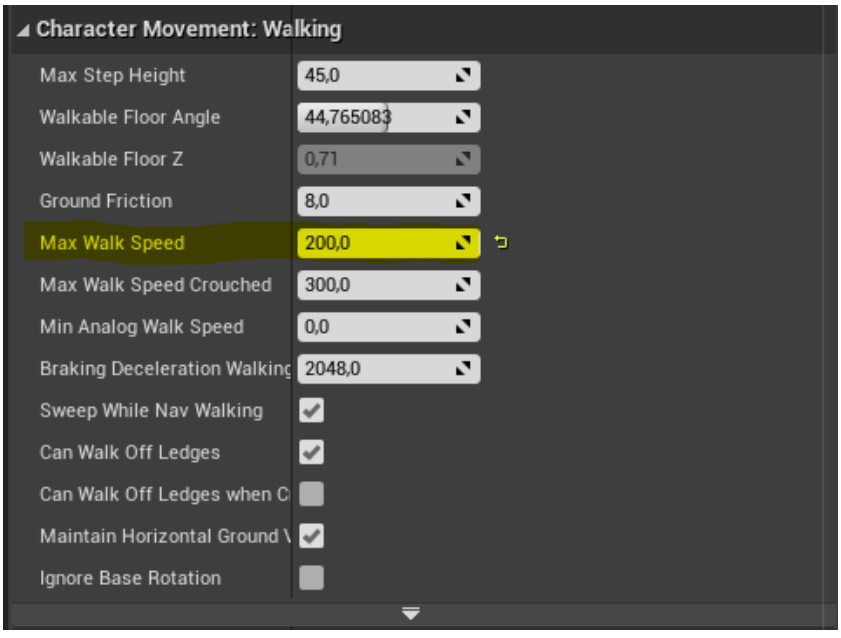

FIGURE 16. Adjusting Walk Speed (Susi 2021, CC BY-NC-ND)

*To decrease the mouse look speed, one way to do it is to add a float*float value of 0,5 between InputAxis Turn and Add Controller Yaw Input and InputAxis LookUp and Add Controller Pitch Input. Change the float*float value to your liking.*

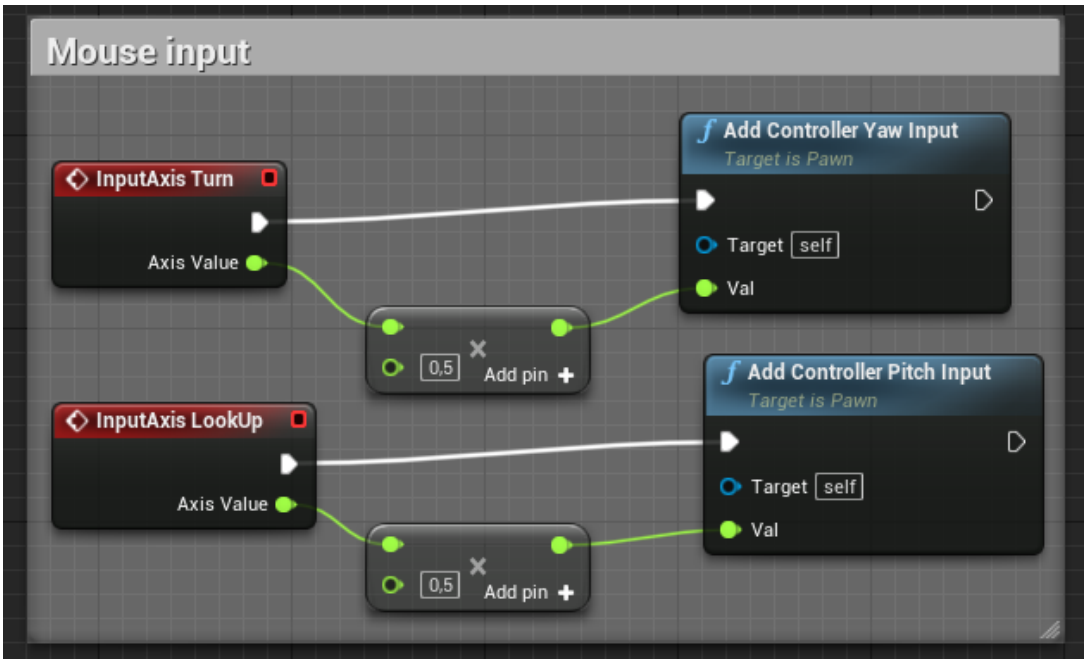

FIGURE 17. Adjusting mouse sensitivity (Susi 2021, CC BY-NC-ND)

4.4 User Interface

Inside a game, there must be a way to communicate with the player. This happens through User Interfaces (UI) and Heads-up Displays (HUD). There are multiple ways to create UIs and HUDs in Unreal Engine. The way used in this project is through Widgets. More about Widgets in 4.4.1.

The Heads-up-display, or HUD in short, is an information overlay on the screen during gameplay. In architectural visualization, you can display, for example, the name of the building and the designers. The HUD is usually non-interactive.

UIs are typically composed of various interactive elements that are typically drawn on the screen. In certain circumstances, they can be added to the game world. Some examples of UIs include the main menu and the pause menu.

4.4.1 Widgets

To create interactivity into the game, one of the first things to do is to create a Main Menu. Probably the easiest way to do this, is to create a Widget Blueprint.

Widget Component is a 3D-instance of a Widget Blueprint that you can interact with during runtime.

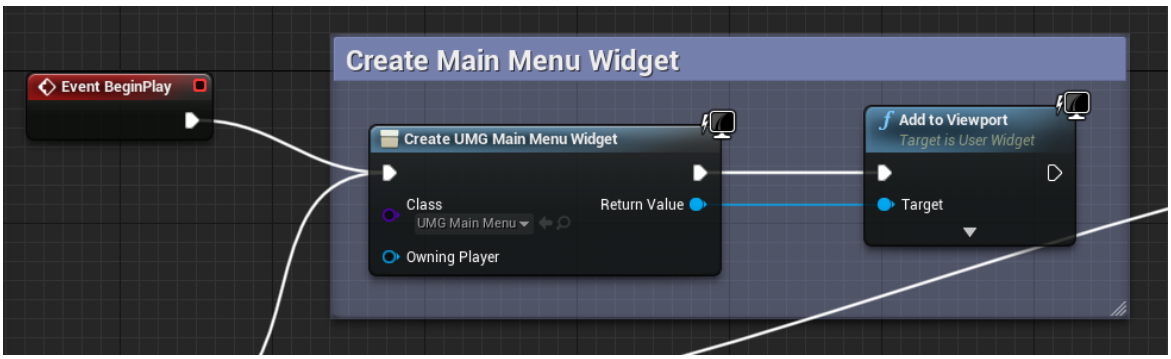

FIGURE 18. Main Menu inside the Level Blueprint (Susi 2021, CC BY-NC-ND)

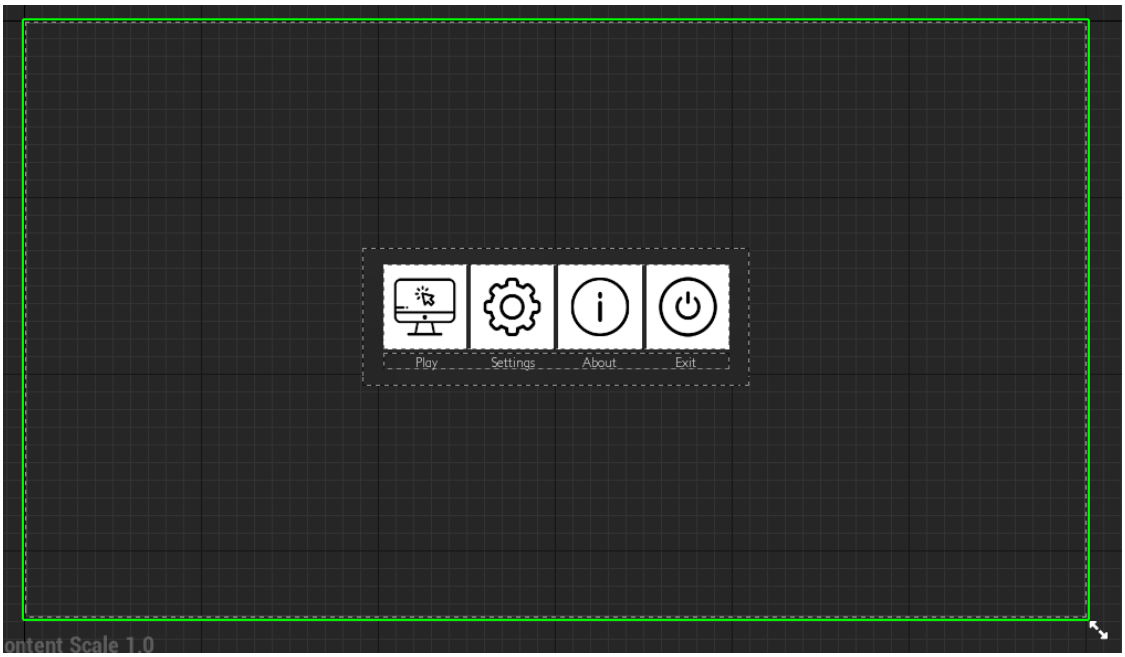

FIGURE 19. Main Menu inside the Blueprint Designer Mode (Susi 2021, CC BY-NC-ND)

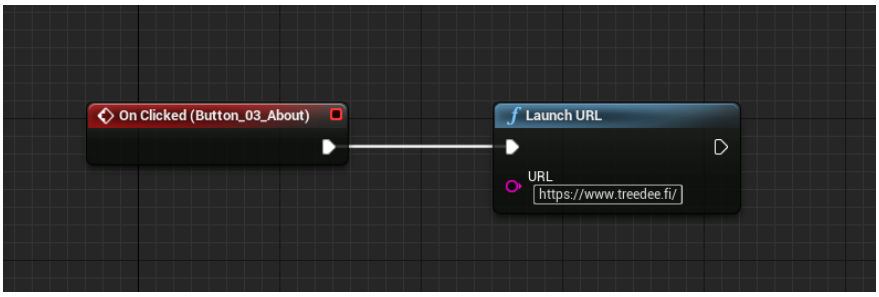

FIGURE 20. Button functionality example inside the Main Menu Widget (Susi 2021, CC BY-NC-ND)

Adding Widgets inside Widgets can be done by creating Parent-Child relationships. For example, graphics settings could be this kind of a nested Widget. However, in this project there is another, completely different, Widget for graphics settings when pressing the Settings button.

4.5 Adding functionality to Level Blueprint

A Level Blueprint is a type of Blueprint that shows events related to a specific level. Events happening in this level are controlled inside a Level Blueprint. Usually everything in architectural

games and models is on one level and therefore it is recommended to utilize Level Blueprint to its fullest. Each level has its own default Level Blueprint which can be edited.

To open the Level Blueprint, in the top ribbon, search for Blueprints > Level Blueprint.

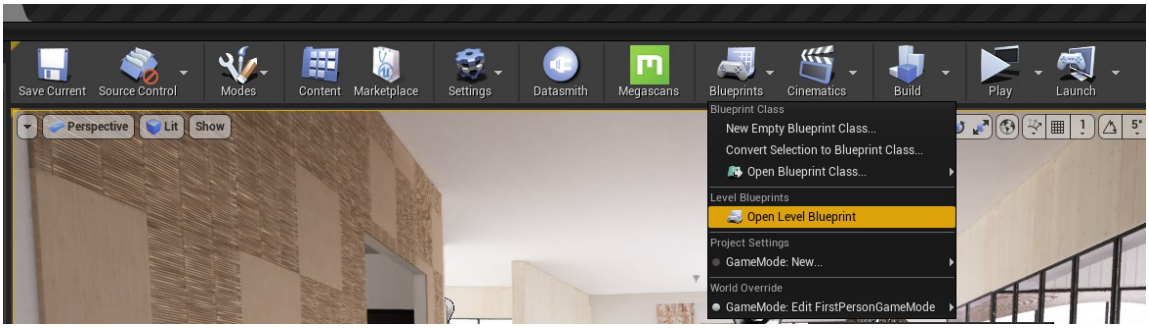

FIGURE 21. Opening the Level Blueprint (Susi 2021, CC BY-NC-ND)

For example, an event to create the Main Menu Widget referenced earlier, is done inside the Level Blueprint.

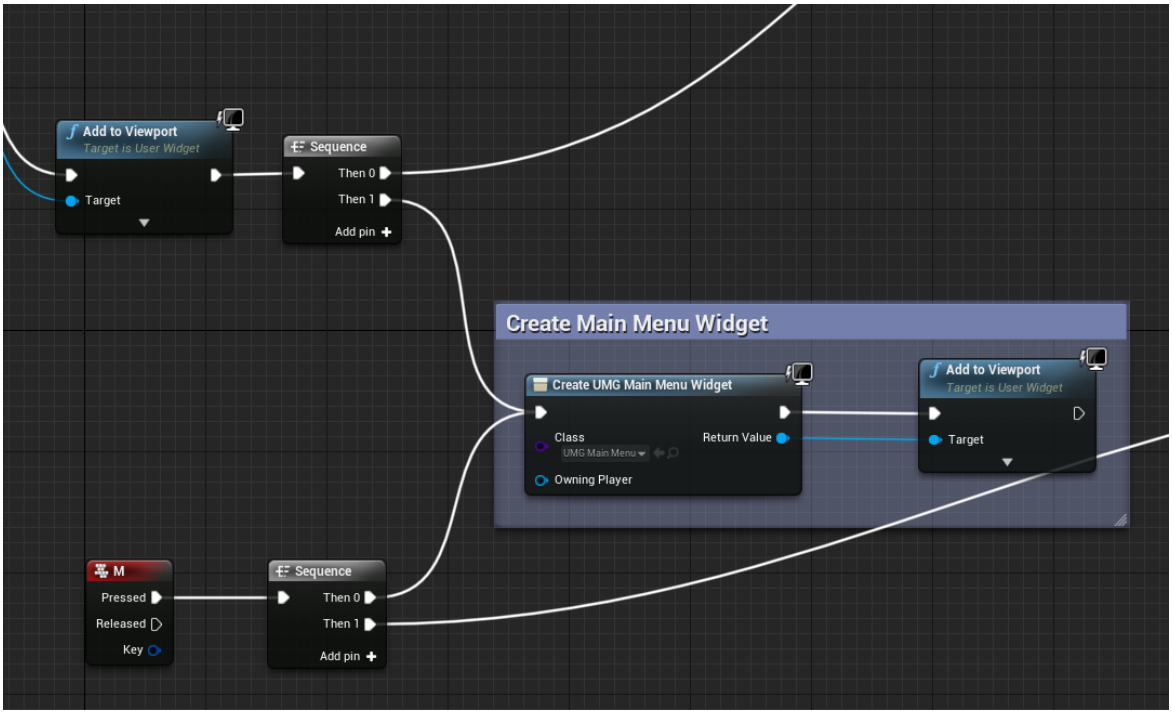

FIGURE 22. A part of the Event Graph of the Level Blueprint (Susi 2021, CC BY-NC-ND)

4.6 Exporting an exe file

Before a project can be distributed, it must be packaged properly. This ensures that all its code and content are up-to-date and formatted in the proper format.

To package a project, find an option called Package Project under the File-menu.

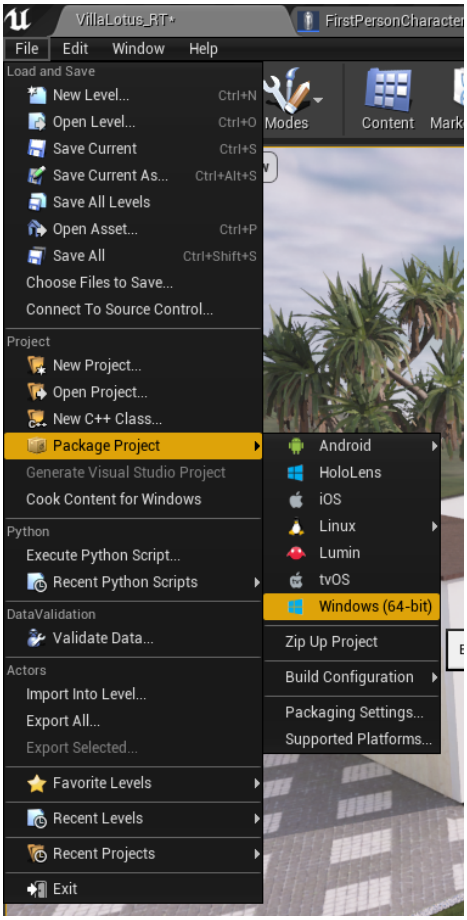

FIGURE 23. Packaging the project (Susi 2021, CC BY-NC-ND)

After choosing the platform, you will be presented with a dialog for selecting the target directory. After the packaging has completed, run the exe file to check that everything works.

5 CONCLUSIONS

The aim of the thesis and the project was to study and examine the uses of Unreal Engine in real-time examination of a building model. The work also included exploring the possibilities of interactivity and achieving photorealism with a game engine.

It was established at the start of the project that the scope of the work was large. The subject area was pruned, but these subject areas expanded during the making of the gamified model. For example, lighting experiments and studying took tens of hours if not hundreds in total. Majority of the working hours during this entire project was spent in creating and experimenting with the model and elements in Unreal Engine.

The programming part proved to be extremely challenging, although the visual scripting system was making the job easier. Some topics had to be eliminated, such as the presentation of Component Metadata during runtime, lighting scenarios, and most importantly VR. In the end, VR's functionality would have taken a huge amount of time to get it to work interactively. And therefore, the final product is a desktop version. The game works as a good base for further development and it now serves as a platform for a possible next thesis on a topic that would deal only with VR.

When we discussed about the VR topic, it was speculated how the larger, real-world adaption would look like in the future. At this point in time, there is a great amount of hype and visions around VR but the practical implementation is lacking.

All in all, I felt that I succeeded in the project. Photorealism was achieved and the model has several interactive elements and functions. I knew the work was going to take time and work, but I would not have imagined it to be so heavy overall. The project taught me much about working with Unreal Engine and the whole project was something I was really interested and deeply invested in.

As for the usability of Unreal Engine in architectural visualization and gamification, the time required to build a working game from scratch is not worth it, in my opinion. Maybe later, after building one functional game, when you already have many of the pieces built and ready to be used in a new project. You can then migrate them into a new project and therefore make the usability worth it in the new project. But creating these functional elements, and adding materials from scratch, will take hundreds of hours.

There also has to be a bigger cause to make building a game worth it. Just visualization rarely does it. There has to be the need for gamified elements. The project should also be a public building so that there would be several people utilizing the game. A school would be a case in point. For example, the students could use the game to change how their classrooms would be organized and decorated as.

Hopefully Unreal Engine 5 and the new Global Illumination system, Lumen, will reduce the time needed to build a photorealistic interactive game for architectural designers. I also suspect that Epic Games will continue to support and maybe even add resources to Unreal Engine regarding the architectural field.

6 REFERENCES

Unreal Engine 4.27 Documentation. 2021. Internet publication. https://docs.unrealengine.com/4.27/en-US/. Accessed between 3.2.2021-18.12.2021.

Selin, Jukka. 2021. Tietomallin pelillistäminen ja toiminnallisen suunnittelun menetelmä rakennusten suunnittelun apuna. Doctoral thesis. Tampere University. https://urn.fi/URN:ISBN:978-952-03-1888-8. Accessed 28.3.2021.

VR Division. 2021. Unreal Engine Master Class. Course.

Unreal Sensei. 2020. Unreal Engine 4 Beginner Tutorial - UE4 Start Course. Internet publication. https://www.youtube.com/watch?v=_a6kcSP8R1Y. Accessed 4.2.2021.

Unreal Sensei. 2020. The Secret to Realistic Landscapes in Unreal Engine - UE4 Tutorial. Internet publication. https://www.youtube.com/watch?v=UIycCZl4lYE. Accessed 15.3.2021.

TheRevitKid. 2021. From Revit to Unreal Engine - Step by Step. Internet publication. https://www.youtube.com/watch?v=arC1aUMgZyw. Accessed 17.3.2021.

Unreal Academy. 2019. UE4 Raytrace Optimization – Creating RT Toggle Widget, Setting Up RT Lights & Tips – Part 1. Internet publication. https://www.youtube.com/watch?v=kIxb1zVosP4. Accessed 22.5.2021.

Wolfe WOLF. 2018. Unreal Studio - Lightmass and Beyond. Internet publication. https://www.youtube.com/watch?v=TzqpyNb0998. Accessed 19.4.2021.

Ben Cloward. 2020. Ray Traced Reflections - UE4 Materials 101 - Episode 41. Internet publication. https://www.youtube.com/watch?v=airQR0BstNA. Accessed 22.5.2021.

Ben Cloward. 2020. Optimizing Ray Traced Reflections - UE4 Materials 101 - Episode 44. Internet publication. https://www.youtube.com/watch?v=quynkx1dTkI. Accessed 22.5.2021.

Polygon University. 2019. How to Get High Quality Reflections in UE4: Quality and Performance Tutorial. Internet publication. https://www.youtube.com/watch?v=Fqw5c6Ow1xA. Accessed 23.5.2021.

3darchstuffs. 2019. How to Rotate Texture in Unreal Engine using Material Editor. Internet publication. https://www.youtube.com/watch?v=XnbujOHlZ9A. Accessed 6.3.2021.

Ryan Manning. 2019. UE4 Interior Lighting Series (Part 1). Internet publication. https://www.youtube.com/watch?v=371YAnQF73I. Accessed 12.3.2021.

Ryan Manning. 2019. UE4 Interior Lighting Series (Part 2). Internet publication. https://www.youtube.com/watch?v=dQ8bqHUg5To. Accessed 12.3.2021.

Ryan Manning. 2021. UE4 - Lighting Exterior Scenes (The Basics). Internet publication. https://www.youtube.com/watch?v=tD3HYY67XpY. Accessed 10.3.2021.

William Faucher. 2021. Demystifying the Skylight [Unreal Engine 4 & 5]. Internet publication. https://www.youtube.com/watch?v=BGoaPyfZlYg. Accessed 23.6.2021.

William Faucher. 2021. Raytraced Glass, Translucency, & Refraction (Unreal Engine 4) + New Raytracing Features Coming Soon. Internet publication. https://www.youtube.com/watch?v=jwZi0KVebo4. Accessed 12.6.2021.

Fabrice Bourrelly. 2020. Unreal for Architecture Raytracing and Lightmass Hybrid method tutorial. Internet publication. https://www.youtube.com/watch?v=daK7wpQ5UEI. Accessed 1.7.2021.

Fabrice Bourrelly. 2020. Photorealism in Unreal Engine for Architecture using Raytracing. Internet publication. https://www.youtube.com/watch?v=EyVAL46uE_I. Accessed 14.6.2021.

Fabrice Bourrelly. 2020. How I work with Raytracing noise in Unreal Engine for Architecture. Internet publication. https://www.youtube.com/watch?v=ZS6co7LPRYQ. Accessed 22.6.2021.

On Mars 3D. 2021. Unreal Engine Lighting Workshop: Lightmass and Post Processing. Internet publication. https://www.youtube.com/watch?v=bRL_LmkF-W8. Accessed 2.4.2021.

Understanding Global Illumination. 2013. Internet publication. https://www.pluralsight.com/blog/film-games/understanding-global-illumination. Accessed 4.9.2021.

Unreal Sensei. 2021. How to Make Unreal Engine 4 Render Lighting Faster! - UE4 GPU Lightmass Tutorial. Internet publication. https://www.youtube.com/watch?v=H3PEVnRGA4s. Accessed 13.5.2021.

Architectural Visualization. 2020. How to use Backdrop plugin to change light and background in Unreal Engine | Unreal Engine tutorial. Internet publication. https://www.youtube.com/watch?v=jwZi0KVebo4. Accessed 30.3.2021.

GDi4K. 2021. Unreal Engine Landscape Lighting Essentials. Internet publication. https://www.youtube.com/watch?v=nLFTStnuPaI. Accessed 2.4.2021.

THE IMAGENEERS. 2020. Unreal Engine 4 Arch-viz Tutorial Course part 3-2 Lighting with HDRI & shifting to GPU. Internet publication. https://www.youtube.com/watch?v=6eP_I0X9SBo. Accessed 22.4.2021.

THE IMAGENEERS. 2020. Request 4 part 1: How To Create a 3D Widget. Internet publication. https://www.youtube.com/watch?v=1rfHC3Xz6OM. Accessed 18.9.2021.

Wessel Huizenga. Publish date unknown. Creating ArchViz Inside Unreal Engine. Internet publication. https://www.artstation.com/learning/courses/l9m/creating-archviz-inside-unreal-engine/chapters/9z8/lighting-setup. Accessed 29.5.2021,

Mark Drew. 2017. UE4 Displaying actor information in UI widgets. Internet publication. https://www.youtube.com/watch?v=ZdueRgtuinQ. Accessed 4.10.2021.

Matt Aspland. 2020. Packaging And Exporting Your Game - Unreal Engine 4 Tutorial. Internet publication. https://www.youtube.com/watch?v=MatkFIvCKMY. Accessed 9.6.2021.

Unreal Engine. 2019. Interactive Architectural Visualization in Unreal Engine | Autodesk University 2019 | Unreal Engine. Internet publication. https://www.youtube.com/watch?v=o_2Q5y1AXVI. Accessed 22.8.2021.

Unreal Engine. 2020. Achieving Higher Visual Fidelity with Ray Tracing | Unreal Fest Online 2020. Internet publication. https://www.youtube.com/watch?v=wv5UeSZST5w. Accessed 14.8.2021.

APPENDICE 1: EXTERIOR RENDER 1

FIGURE 24. Exterior Render 1 (Susi 2021, CC BY-NC-ND)

APPENDICE 2: EXTERIOR RENDER 2

FIGURE 25. Exterior Render 2 (Susi 2021, CC BY-NC-ND)

APPENDICE 3: EXTERIOR RENDER 3

FIGURE 26. Exterior Render 3 (Susi 2021, CC BY-NC-ND)

APPENDICE 4: EXTERIOR RENDER 4

FIGURE 27. Exterior Render 4 (Susi 2021, CC BY-NC-ND)

APPENDICE 5: INTERIOR RENDER 1

FIGURE 28. Interior Render 1 (Susi 2021, CC BY-NC-ND)

APPENDICE 6: INTERIOR RENDER 2

FIGURE 29. Interior Render 2 (Susi 2021, CC BY-NC-ND)

APPENDICE 7: INTERIOR RENDER 3

FIGURE 30. Interior Render 3 (Susi 2021, CC BY-NC-ND)