

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης
Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος:

Development of a learning materials digitization Platform

Δημητρης Ιωσηφίδης Χοκμετίδης (Α.Μ: 4366)

Επιβλέπων καθηγητής: Βιδάκη Νικόλαος

Ηράκλειο - Απρίλης 2021

Acknowledgments

Abstract

The visualization of data was always one of the keyways to communicate ideas in the modern world. Most visualization models occur when big companies or even small groups of people want to view the problem in a different manner from the ordinary. Visualization is one of the most common ways to express and view those problems and solve it with the best and cost-effective solutions. Aside from metrics, graphics visualization has evolved to be one of the most fast-growing trends in terms of expressing an idea. 3d models used in different ways have proved to be invaluable to many sectors like education, manufacturing, health, and entertainment. In the last decade real time engines have improved the capabilities and introduced more dynamic experiences providing the user with a complete picture of the task ahead.

Πίνακας Περιεχομένων

Abstract 3

Πίνακας Περιεχομένων 4

Λίστα Εικόνων 17

Παραπομπές 18

- **Introduction..... 5**
  - *Introduction 5*
  - *Motivation-Initial problem statement…… 5*
  - *Outline…… 5*

- **Methodology…… 7**
- *Methodology..... 7*
- *The role of the Environment artist..... 7*
- *Environment art..... 7*

- **Pipeline 8**

- **Idea 9**
  - *Idea 9*
  - *Concept art 10*
  - *Photo reference 12*

- **Asset planning 14**
  - *Asset planning 14*

- **Blockout 16**
  - *Blockout 16*
- **Modelling 18**
  - *Modelling 18*
  - *Foliage 20*
  - *Photogrammetry 22*
  - *UV mapping 24*
  - *Materials 29*
- **Placement 30**
  - *Placement 30*
  - *Grouping 30*
  - *Mobility 30*
- **Districts 33**
- **Lighting 33**
  - *Lighting 35*
  - *HDRI 36*
  - *Cubemaps 39*
  - *Volumetric fog 39*
  - *Reflections 41*
  - *Screen space reflection 42*
  - *Ray-traced reflections…… 43*
  - *Mesh Distance fields 44*

- **Optimization 45**

  Optimization 45

  Instancing 47

  LOD 47

  Culling 48

  Forward shading 49

- **Navigation 50**

  Navigation 50

  Collisions 51

  Core navigation 53

- **Weather system 55**

Introduction

Ideas are the centerpiece of human interaction. In business a well-formed idea can be the decisive factor that can save a lot of money and communicate a single vision. The ability to visualize is the key to solving many problems in the workspace. Modelling something like a product that only one person thinks about can communicate to everyone what that product looks like and its function. That is why real-time engines have proved to help many teams create different interactive scenarios. Those interactive scenarios appear more believable than most with use of many methods giving us back a better view of the result we seek.

This project describes visualization with the literal meaning of the world by creating a real time environment of a real-life location. The techniques used to make the environment are part of real-life production pipeline meaning that they are industry ready. Unreal engine is a key program used to realize the digital environment because it provides us with advanced lighting algorithms along with a high polycount budget allowing for more detail and believability.

Motivation

The main subject of my thesis involves graphics and most of all photorealism, which was an intriguing part of the industry and a big part of the entertainment products I experienced as a kid and on a greater scale as an adult. By creating an environment from scratch, I am willing to learn the basic techniques used from the industry pipeline and create a believable-playable environment targeting state of the art graphic fidelity.

Initial problem statement

The project involves the idea of a real but not accessible environment. By creating a digital copy, a team of small or large size can plan their project with big accuracy best on real life data. This problem is very apparent in film production that they want to plan things in advance for a shot.

A virtual environment can provide information that will lead to better planning making it a cost-effective solution. In the games industry digital environments can be used to recreate past time periods with accuracy or fantastic ones that cannot be seen or photographed creating a unique experience for the player.

Outline

The project starts with a brief introduction to basic concepts of video game-film techniques and

their history. Chapter 2 will explore environment art from the past along with some of the tech used to create them. Then in Chapter 3 we will see how those techniques along with new ones are used to create models that will later populate our scene. Chapter 3 will follow the creation of a mood board and research of reference images. Stepping on chapter 3 Chapter 4 will create additional mood boards for creating a variation of the real location. Chapter 5 will be an in depth look at the Unreal engine and forming our scene based on what we made before with close reference to the real location. Chapter 6 concentrates on lighting techniques that will improve the quality of the environment overall. Chapter 7 will concentrate on optimization and navigation systems.

Methodology

An environment is created with a certain pipeline in mind. Environment design requires a good use of reference images to define the vision and the visual style. Also reference images help with lighting parameters and object placement to create a more believable result. There are variations of 3D software like 3DS max that enable the creation of 3d models, photoshop for creating textures and Unreal that utilize all the above for creating the environment.

The creation of the environment follows a certain pipeline to ensure the best creation times along with good design choices. The design choices are often dictated by the story or reference images giving it a certain aesthetic or style.

The role of the Environment artist

Certain roles are as crucial to the industry as the environment artist. Responsible for creating backgrounds, layouts and environments for film, television, and games the environment artist uses 3d tools to realize his vision. The environment artist responsibilities include the creation and use of models and textures, working closely with the art director, the tone and esthetic of the environment and doing an extended research about the subject.

Environment art also requires the use of other skills relative to art. Some of those skills are strong knowledge of geography, architecture, and the laws of physics, drawing skills, color theory etc. Those are an integral part of the process and will have a crucial role in decision making throughout the process.

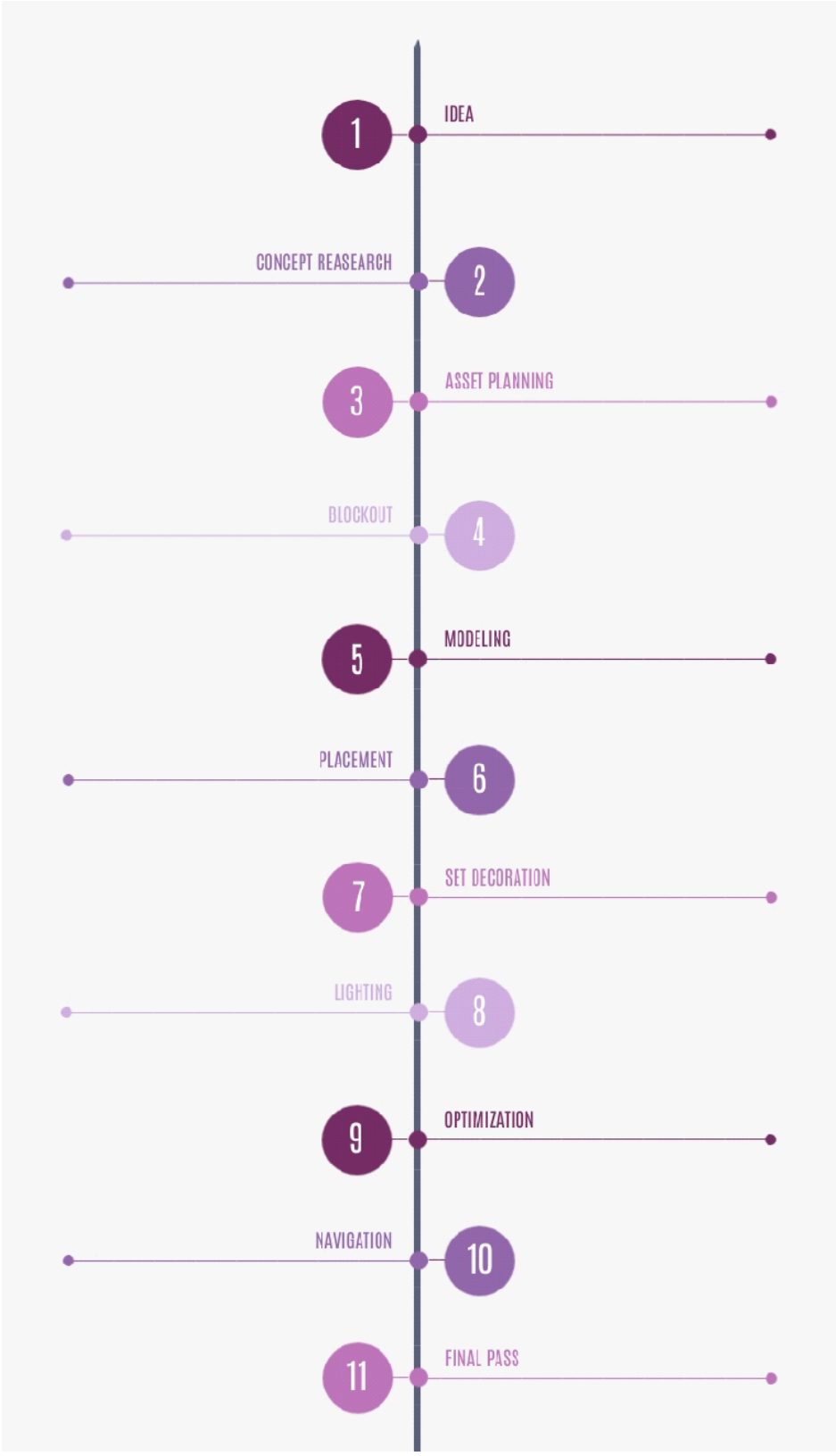
Environment art (3d)

Environment art is the total sum of models, textures, composition, and lighting that form the digital environment. All digital environments have boundaries for a certain area that the environment artists must populate with assets by forming the environment needed. The player area is commonly referred to as a ‘Sandbox’ which is the indicated player area.

Environments can vary from project to project, some environments require a greater scale like a landscape and others require less space but more detail like an urban area. Each one also requires a different approach in lighting mesh creation, navigation and most certainly optimization.

Environment art also depends most on the given reference art or image and from there can take on many different paths such as realistic recreation of a city or based on concept art a fantastical or futuristic perspective on the same subject.

Pipeline



Idea

The core of the project revolves around the idea of reimagining an existing urban landscape. This

means that all building, roads must be converted in such a way that the environment will feel real and authentic. Research is key to ensuring the authenticity of the environment along with the pre-existing location, this helps in terms of scale and decision making about what to keep or what not.

The scale of the environment will be no bigger than a small neighborhood of Paris, France. Meshes populating the environment along with other elements will populate this neighborhood according to the concept art and photo research. Since no concept art can be created (because this requires a different skill set) we will find it online and then we will combine them to create a mood board which is the visual guideline of our project.

Since we cannot Paris, we will have to find real reference and then combine it with research found on movies, videogames and webpages that illustrate certain aspects of our project such as architecture, meshes and other things that ensure the authenticity of the environment. Some video games illustrate the sense of a world and help us get a unique look to those locations.

DreamWorks Home





Concept art

Concept art are several images that translate our idea in the best possible way, they do not have to be specific or visually similar, they only must translate our idea visually. Concept art is the early process of an environment with the focus on art. The concept artist's job is to draw it and after that the environment artist must pick and choose which elements of the drawing he/she will keep so that

navigation and optimization will be feasible.



The above example illustrates the city of Paris and gives us a good reference about the buildings, lighting and objects that populate the scene. This image can give us enough information about the parameters we want the rest of our project to be and can guide us to what types of images we can search in the future.



Searching for specific concept art can sometimes be very difficult especially when we narrow down options. For this project we had two main filters that had to apply to get the best results. The one is the city itself, Paris which is essential to get because of the unique city architecture and the post-apocalyptic which is essential to getting the aesthetic that we need. Both are important but they can be limited resources that’s why photo editing even with simple steps can be invaluable.





A simple desaturation can turn an image that is irrelevant to our aesthetic to an image that is closer to our main concept art. Those are simple methods that can be used to find a bigger number of images fitting our project and giving a better grasp on how we can create our environment.

This technique can prove even more beneficial if advanced imaging programs come into place like Photoshop. Photoshop allows us to place in an existing image objects or people we do not have in the pre-existing material. This helps a lot if we have a specific image or environment in our mind. By creating multiple images, we already have a good sense of the environment we want to create.

Photo reference

Photo reference is a visual practice created by looking at photographs so that we can have a better sense of objects, architecture, and mesh placement. The difference between concept art and photo reference simply is that photo reference is a real photo of the place we want to recreate and concept art is not, it is simply a piece of art trying to evoke the atmosphere and esthetic of the piece.

Photo reference can be found again by browsing on the internet for pictures of specific aspects of our environment such as buildings, mesh materials etc. Photos can be much easier to gather than concept art, especially if the place we want to create is the capital of a country, in this case Paris, France. Google maps can also be a great tool for research since it provides us with a street view anywhere on the planet. An extension of google maps is google earth that can provide 3d map data meaning that we can view the size of buildings in the street and get a greater sense of scale for our scene.

For more accurate photo reference a physical digital camera is a must since most photos will be taken by the environment artist. This appears to be a common tactic among game studios since most of them will send missions to other countries to collect photos for several projects. The advantage of taking the photos on your own is that you can choose the composition, angle, lighting allowing you for a better postproduction in an imaging software like photoshop since you have the raw files, allowing for greater flexibility.

Architecture:



City streets:

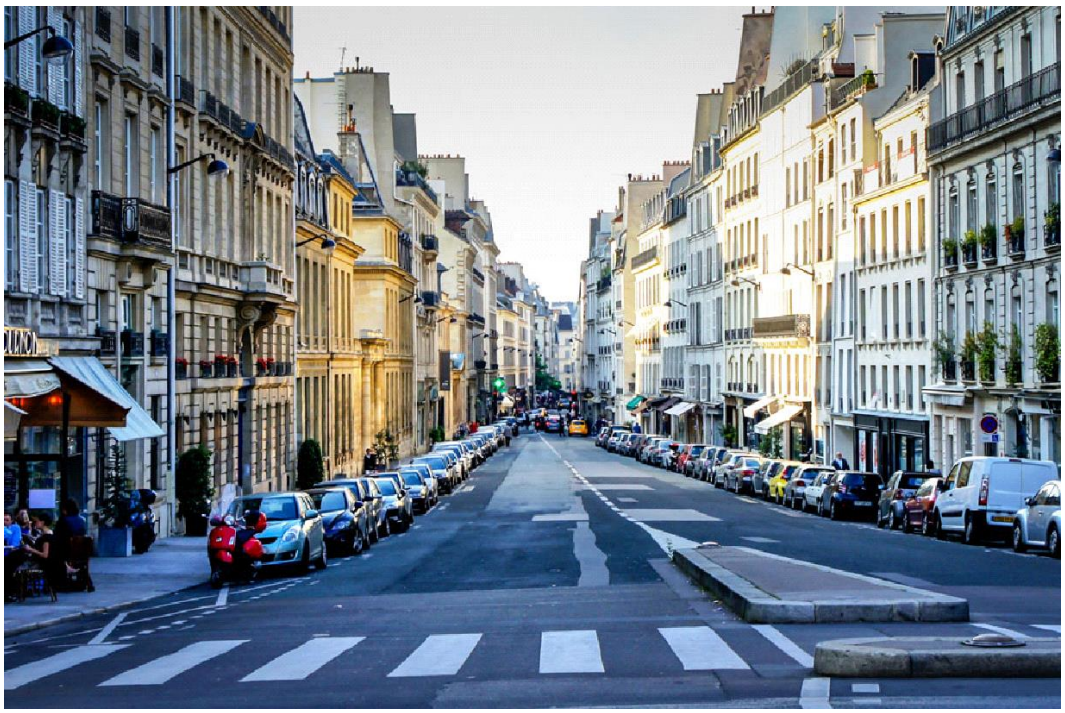


Photo reference usually comes first of the concept art since the art directors see a collection of images to produce the re-edited environments. Later years we see that environment artists can start working even before concept art is finalized with the help of photo reference, building the initial town/city and then changing it according to concept art. Below you can see an example.



Asset planning

Asset planning is the simple act of choosing which assets we are going to need to build our environment from scratch. Assets also vary in size. A building must be built in parts so that we can create variations of building by simply changing individual pieces. This happens because the player's eye mostly focuses on big objects rather than small, detailed objects making the importance of variation critical to the project.

Smaller assets like trash bins or signs on the street have a straighter forward modelling process since all of them are about the same size and color. Another use of smaller projects like pebbles can be placed using instancing. Instancing is a technique that produces multiple asset copies from a single 3D mesh (object) . This is a valuable technique for placing a great number of objects (like the rubble in the picture to the left) without compromising performance.

Scale is the main component that brings everything together. Scale refers to the ratio of the distance in our environment to the corresponding actual distance in the real world. This is a crucial step to ensure how realistic our environment is. The most apparent example is that a building can be just half the scale of a bin in the street, this makes the object seem fake and unrealistic and certainly in no close relation to the player navigating a road.

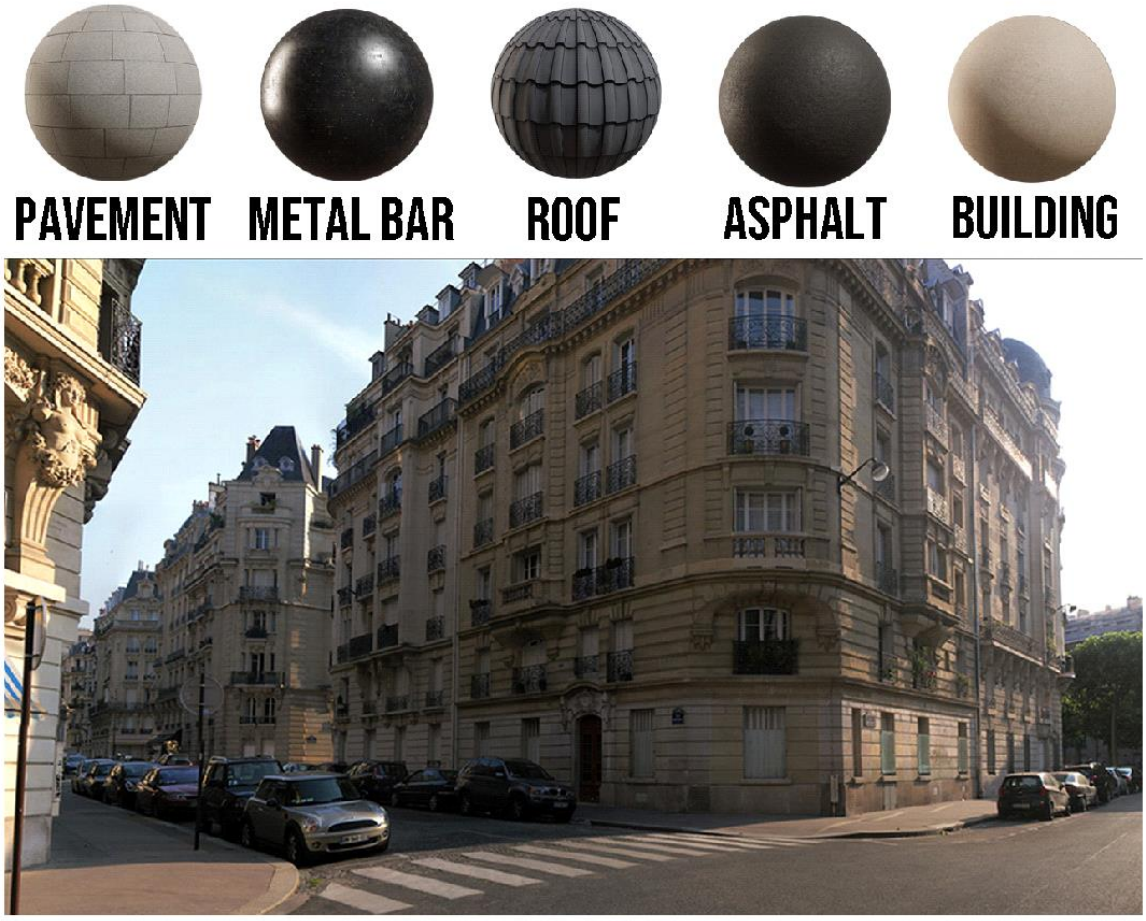
The main reason for organizing our meshes is based on importance which has close relation to what the player experiences. This means that it's important to pay closer attention to meshes that can be seen by the player in greater detail such as objects placed on the road along with vegetation or big buildings that have enough mass, enough to capture the player's eye.

With that said, here is a board with specific attributes to each object, mark A is top priority based on player experience versus B which mainly involves objects outside the player's view.

NAME	SCALE	IMPORTANCE
Building lower floors	Big	A
Building upper floor	Big	B
Roads	Big	B
Cars	Medium	A
Debre	Small/Tiny	B
Vegetation	Big/Medium/Small	A
Street objects	Medium/Small	A

Asset planning can also translate to material planning based on reference and the targeted meshes you want to create. Taking a reference image and then doing material tagging can prove very beneficial to an industry workflow. Material tagging is trying to match the material seen in an image like concept art or real-life reference photos so that we can focus later creating each one specifically for this reason. This is a process we will discuss more in the modelling section of this paper since modeling and UV formation can play a critical role on how our texture/material is applied on a surface that we modelled.

An example of material tagging can be seen in the picture below.



As you can see after doing material tagging, we have a better idea for specific materials and their importance to the scene. Note that in bigger projects materials won't be that specific because the material artist has to find a better long-term solution for each object of their scene. By creating a parent material, we can create many variations of color and texture to suit many needs. Take the mentioned metal bar material, its parent material is a material with a plain metal texture. From this point on we can modify it in any way we want like changing its color to black and adding micro details like scratches and dirt to evoke a sense of realism. The same material can later be used as a metal framing for a window, always created by the master material (parent).

Blockout

Blockout is one of the first phases of level design and environment creation. Blockout the part where we use geometric shapes to block-in your level designs, game environments and game art assets. The result is to blocking-out the layout and establish shape, size, scale and composition of the level and keep in mind of creating a playable level along with a playable space,to begin playtesting. Blockout's purpose is not the finished environment and will change along the way. This phase often is very rough and messy and must be flexible to be able to change things around working the whole map, determining all variables of your scene.

Key characteristics of blockouts are

- Simple design
- Construction is mainly geometric shapes
- The focus is on size, scale, proportion and composition.
- Sometimes demo lighting can be applied.
- The focus is on the entire scene and not a particular object.

In terms of a small environment blockouts are easy and cost effective but the problem in our case is the huge amount of data a city block can have. Another reason that a real-life city is very difficult to recreate as a blockout is the formation of the roads, rivers, and buildings in general specific to that area. For a human to recreate an accurate blockout replica it would take many hours of carefully planning based on satellite images. For that reason, the best possible way to recreate a city block as accurately as possible with the most cost-efficient way is with the help of 3d data collected from google earth and most specifically with the use of Unreal's Twinmotion technology.

Twinmotion: Twinmotion is a mature viz tool used by architects and visualizers alike to create architectural scenes in a virtual environment. Made for architecture, visualisation, construction and urban planning simulation, Twinmotion combines an easy to use interface along with Unreal Engine. Twinmotion is easy to learn, regardless of size or complexity of each project. Enables linking with Archicad and enable users to move from the in-house modeler.



In the above image you can see the Twinmotion navigation window with the selected area we want to sample. Twinmotion will draw data from this highlighted area and will create our blockout scene based on the real road and structure of the city.

*In this process the attitude cannot be calculated this means that for towns built on mountains this process will be different. For this case Paris often comes across a couple of hills but for the most part it is built on a flat area.



After Twinmotion has collected all the map data we see the result. This would take probably a couple of days for an artist to assemble assuming that he will have to spend time researching and shaping each building outline individually in 3d software of his choice. For this project we will recreate this park along with some of the buildings circling it.

First, we will create the close to reality version of Paris and then we will work on changing that and creating a fiction version based on reference and artistic ideas that we gathered in advance. Building first the base will help us make less changes and have an already good base for shaping our environment in the long run.

After creating our blockout the first thing we need to do is to port it in the engine so we can then import models and meshed that we will create in 3Ds Max that will eventually replace the blockout versions.



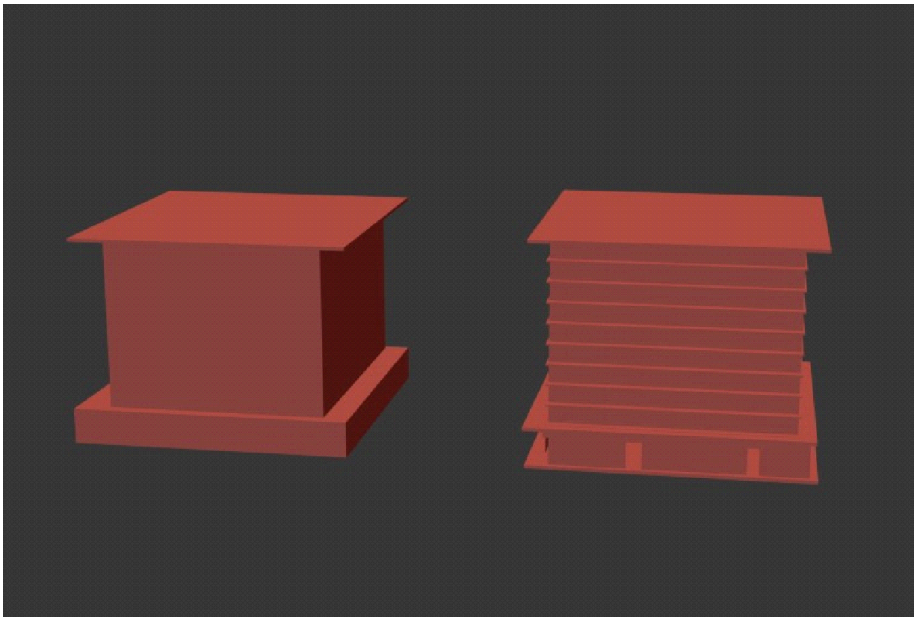
Modelling

Modeling is a major part of each project and the start of a project taking shape. 3D modeling is the process representing an object in three dimensions via software. The product is called a 3D model. Three-dimensional models display a physical body using a series of points in in three dimensional space, fused by multiple geometric shapes as triangles, lines, surfaces, etc. Assembled by a series of data, 3D models can be from the user, procedurally, or by scanning the real model. In this case since we cannot scan because of the scale we will have to model manually the objects we want for our scenes.

Starting with buildings we will have to follow our reference and slowly work our way to a complete model. The 3d modelling process starts with building our desired object with the use of primitive objects. Geometric primitives are objects that exist in the real world such as balls, cones, barrels and pyramids. In 3ds Max, we can model many objects of similar geometry using a primitives. Primitives can also be combined into more complex objects further refining them with 3ds max modifiers.

For the first step we will have to create the basic characteristics of a building such as shape and size, with the use of simple box shapes and then with the use of the scaling tool we will adjust their dimensions to get the desired proportion we want for our final object. After a simple face of our model has been created, we move to face two which is finding the characteristics of our model and applying them to the one we just created.

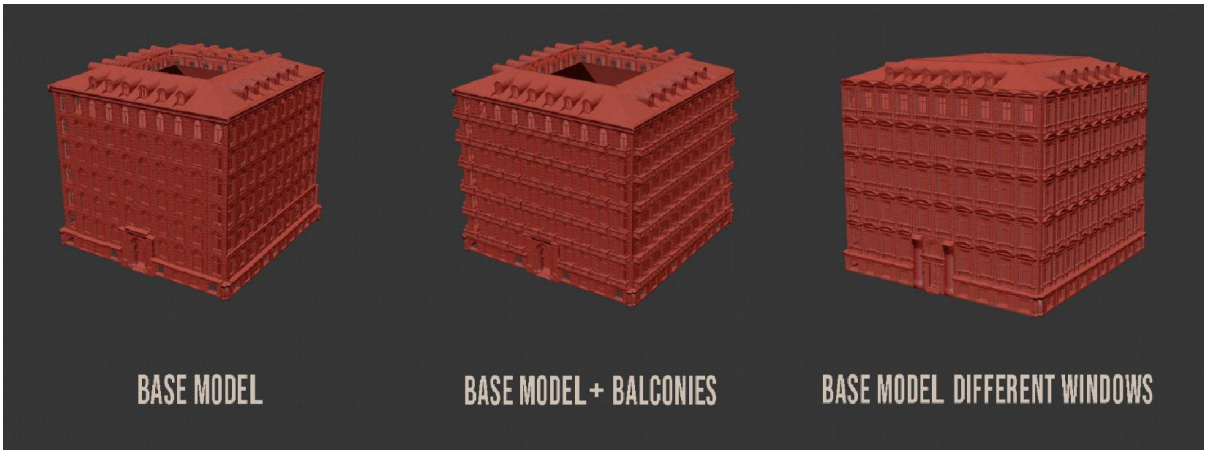
Buildings have several secondary characteristics as individual shapes on top of the basic box shape commonly referred to as floors. To create them we will have to use 3ds max modifiers. Modifiers are a way to sculpt and change objects. They can modify the geometry and its elements. The main modifier we will have to use in this example is the Editable Poly modifier that allows us editing tools for the selected object: vertices, edges, borders, polygons, and elements. The other is the Extrude modifier which adds depth and shape to a shape object and makes it a parametric object.



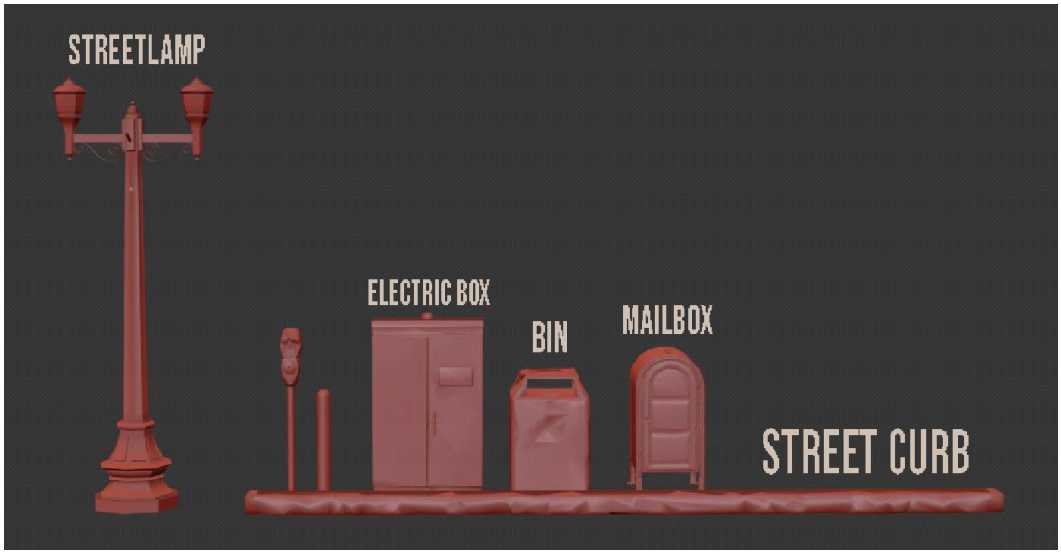
The process is like other objects, the more complex the objects we want it will require more time and more iterations to complete. Simple street objects like a street bin will take less time to create because it is much closer to 3ds max primitive shapes and more complex objects like buildings will have to be shaped many times to come closer to our reference images. This process can take a lot of time and many changes can be done, so for the part of the paper we will stick with the final models since the main subject is real time engines.

Below you can see the complete model of one of the buildings. The building block can be divided into windows, roof, walls, and entrance. This is our base model and from this one we will create more variations to protect our scene from repetition. Adding simple assets can create a different mesh in the eyes of the player. The second model uses balconies to cover aspects of the base model. If we want to cover up more of the base model then more adjustments must be made like windows, entrance. Keep in mind that the roof is not something that needs change since the play area is in the lower parts of the buildings.

This is the same principle we mentioned in asset planning. The most important aspect is what the player sees, and this is something that will always determine our action even in modeling. Changes in the lower parts of the buildings up to the second floor must be made so that we do not break our environment's immersion.



For the rest of the city, we will need even more meshes including all we mentioned in the asset planning list. This includes meshes that the player will come across on the street like street lamps, garbage bins and road blockade. Even though those assets are small they interact with the player so we will need to render them at full quality in the engine.

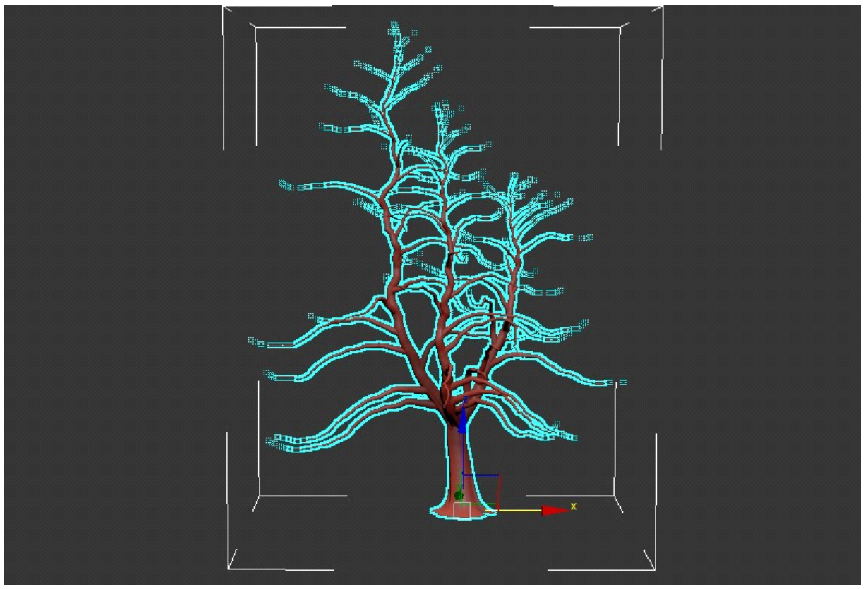


Foliage

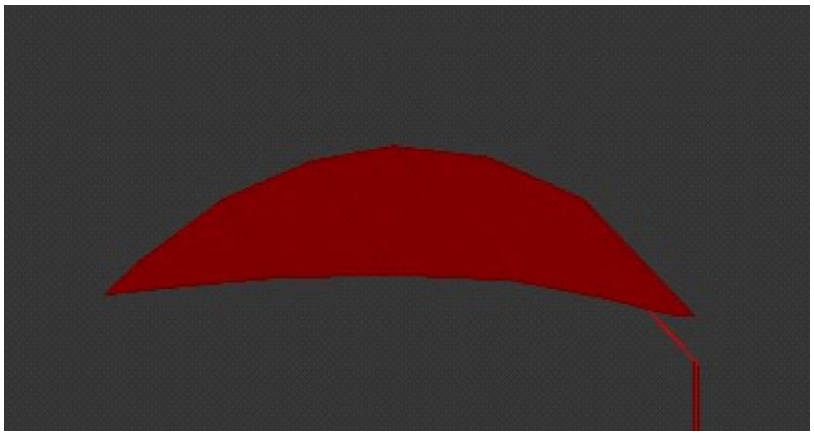
Modelling trees and vegetation has a slightly different workflow from hard surface modelling mainly because of the model's complication. Scanning also is out of the picture for reasons we will talk at the photogrammetry section of the paper. A tree is divided mainly into two parts: the bark (including the branches) and leaves.

Both bark and branches are formed using splines in 3ds Max. First, we will shape the main body of the tree and after that individual branches. After tracing each line, we need to thicken them

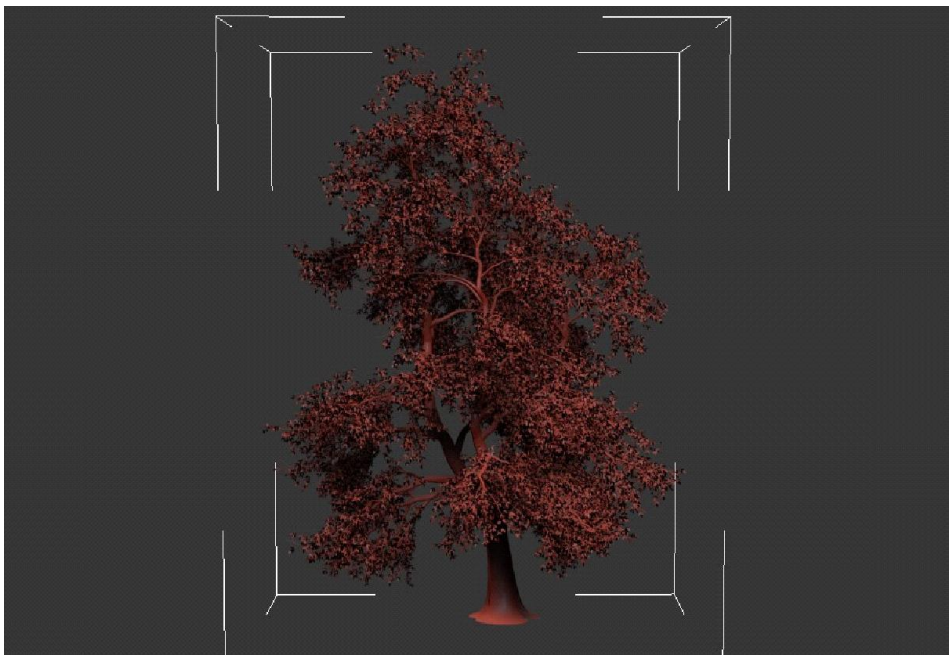
first the bark and then the branches. This can be done using the shell modifier which gives thickness to an object by adding an extra set of faces facing the opposite direction of existing faces, plus edges connecting the inner and outer surfaces wherever faces are missing in the original object. Now that our shapes have volume, we will need to close the edges by using the edit poly modifier and linking each vertex to create edges on the tree's branches.



After creating the tree's main body, we need to populate each branch with leaves. For this step we will have to create one leaf first and then populate each branch with the object paint function. The leaf can be recreated by an image plane first or from an imported file from a image editing software such as Photoshop or GIMP.



The Object Paint tool enables you to paint objects freehand anywhere in the scene or onto specific object surfaces. You can also "fill" selected edges automatically with paint objects. You can paint with multiple objects in a specific order or at random and change the scale as you paint. So, after creating a single object we can create a random seed. A random seed is a random placement based on size, rotation producing variant results.



The same workflow can be done with any vegetation object like bushes or small trees. The only thing to consider in each process is the anatomy of the plant created. A good example is a fern. Ferns leaves differ in shape from conventional branches too. So, to correctly recreate a plant, we need to first have a reference image to know how we will approach the modelling process based on the plant's anatomy. After creating the main body and the first leaf is the same process as before. Note that vegetation like grass with small screen time will not need to be modelled in a 3d and can be done by merging 2d planes. This happens mainly because of time and optimization further down the line.

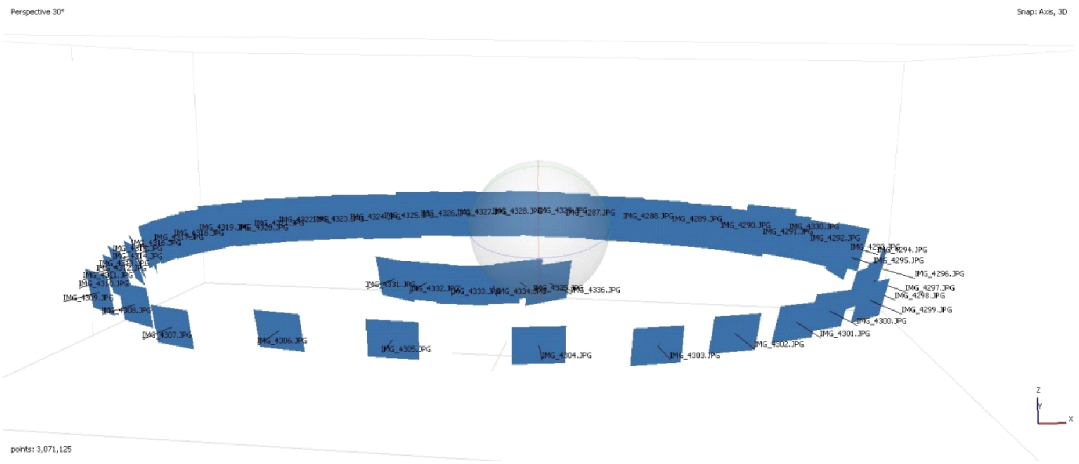
Photogrammetry

Photogrammetry is the method of gaining data about real objects in the environments through the process of calculating, and processing photographs. Photogrammetry uses methods including optics and projective geometry. Digital photography and photogrammetry process includes multiple steps that enable 2D or 3D data of the given object. The data shows what type of information that goes to the finished model.

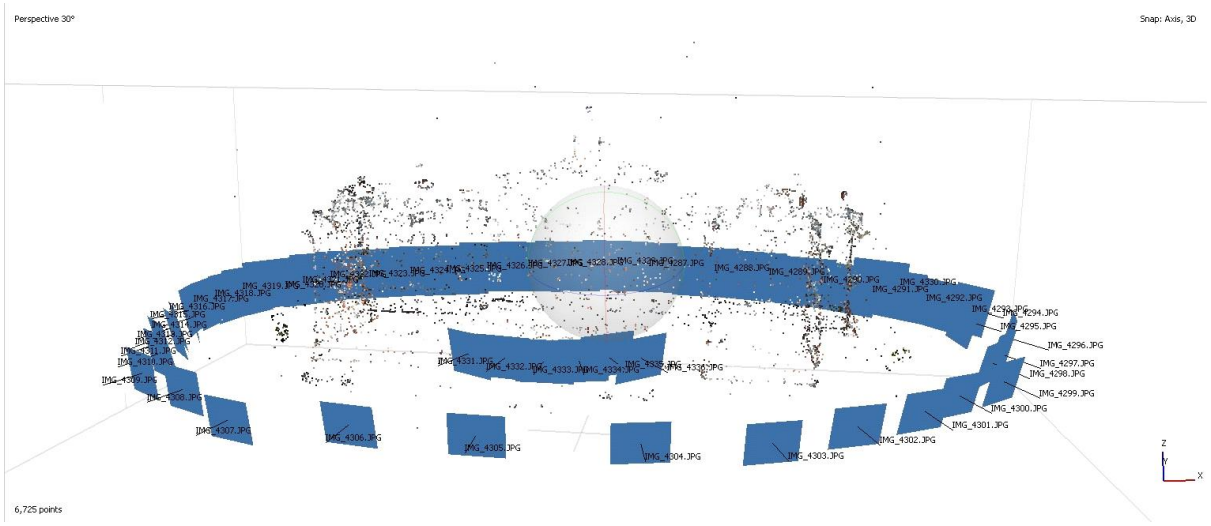
The given data show us the orientation of the photographs taken in the 3D space. The coordinates exact image locations on the camera's metadata. The rotation and tilt of a camera defines its position. The orientation shows the parameters of the imaging process. Those parameters are the lenses along with further characteristics like the distortion or image effects. Further observations play a critical role: With scale bars, basically a known distance of two points in a certain location.

For the sake of this paper, we will use the free trial version of Airsoft photo scanning program. But before we start with the process in software, we will have to photograph a real object outside the street. The process of photographing can be difficult always depending on the object. Matte material surfaces are easier to photograph-scan than glossy or reflective ones. Essentially the photograph is the information, scale, and material surface that we want to recreate, in glossy or reflective surfaces in sunny days areas of the object can reflect and destroy the material information. So, an ideal day for scanning objects is a cloudy day and a good planning of the material properties of each object we want to scan. Objects like mirrors or transparent glass must be recreated in a 3d software instead of scanning it in the real world.

Photogrammetry is often used in organic objects such as rocks, tree barks and other natural elements mainly because the complexion is vast and most of the time they do not come in predetermined shape, size or form. Using google maps we can capture buildings from the roads that circle it and gather all the photos we want. The photos ideally must be captured from three different heights: lower angle, angle at half of the building's height and height above it. Unfortunately for us this is impossible since google maps capture photos on the ground level from a car top so we will gather as many photos as possible from ground level.

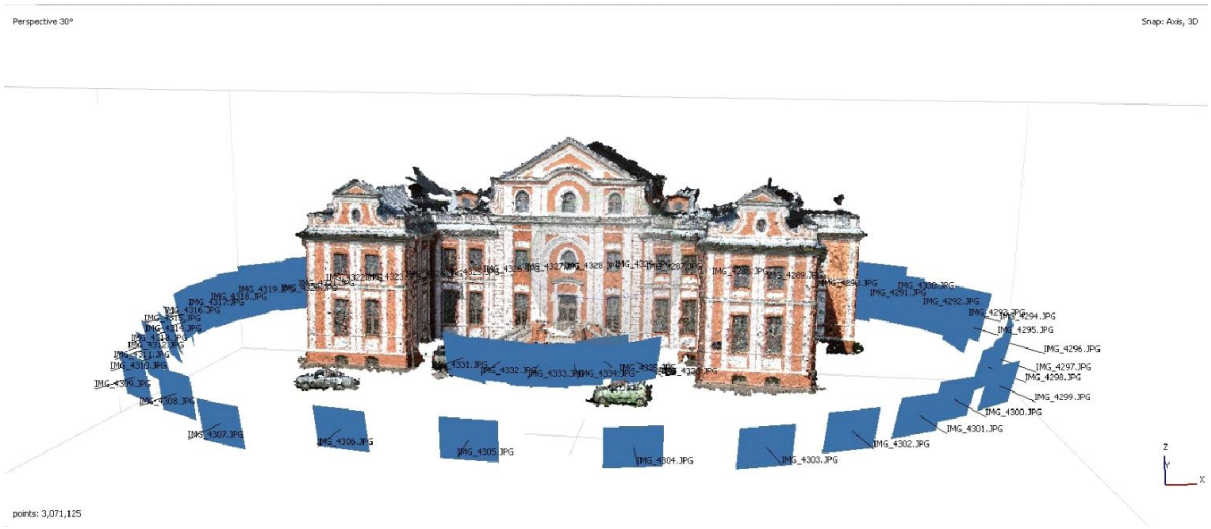


This is a screenshot from Agisoft software showing the angle of each photo we collected in a ring formation. Besides the photos located in the ring there are some photos located in the center and this is because some parts of the buildings like stairs must be photographed separately for accurate depiction. Following the photogrammetry process, we will have to create a point cloud first and then proceed building dense clouds before we complete the process by creating our mesh. Note that sometimes human error in photographing can create a problem in the process. This is because humans can be inconsistent in terms of evenly photographing an object from the same height or angle. Agisoft provides a photo alignment function that can align each photo based on reference point for more accurate results.

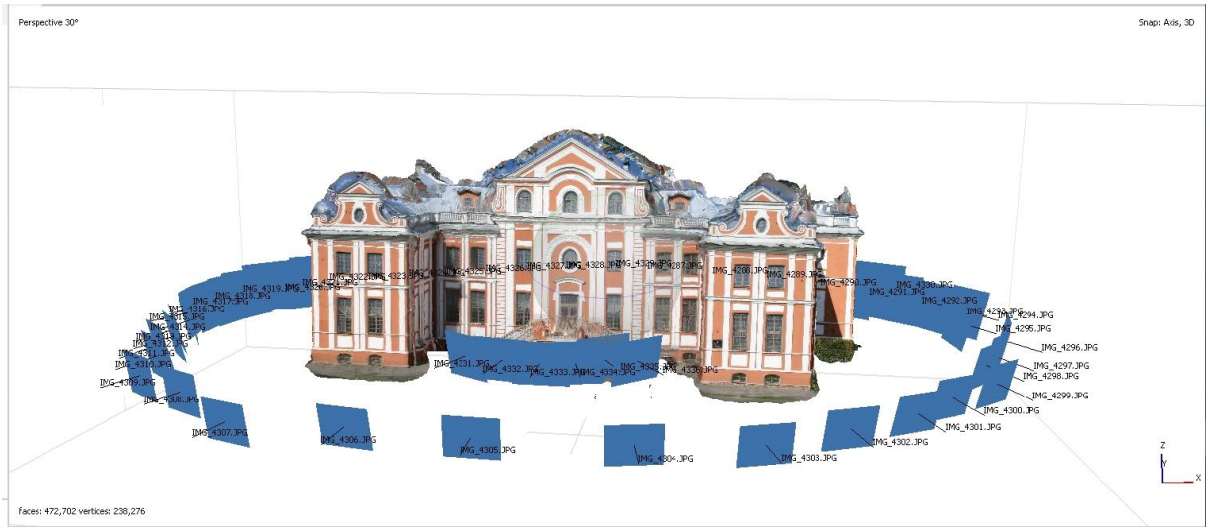


This image shows the point cloud generated from the images. Point cloud data coordinate points in 3d space. These dots represent the object in space. Those points are mapped in the X, Y and Z axis. The point cloud is created scanners or from the photogrammetry method, which is how many points are in the exterior of the object that is been captured.

The result of the scanning process are point clouds that are used for many uses, used to generate CAD models for industrial parts, for visualization, animation, and custom applications with complex models.



After creating a dense cloud, we can finally see some of the results of our photogrammetry. Dense cloud is a more compressed and well-formed version of the point cloud but still has some information that we will not need for our project like the cars in front of the building. Miscellaneous static objects like cars and trees can sometimes block our object especially if it is a building. Therefore, before building our mesh, we will need to clean our scene of any objects that we don't need to be fully rendered.

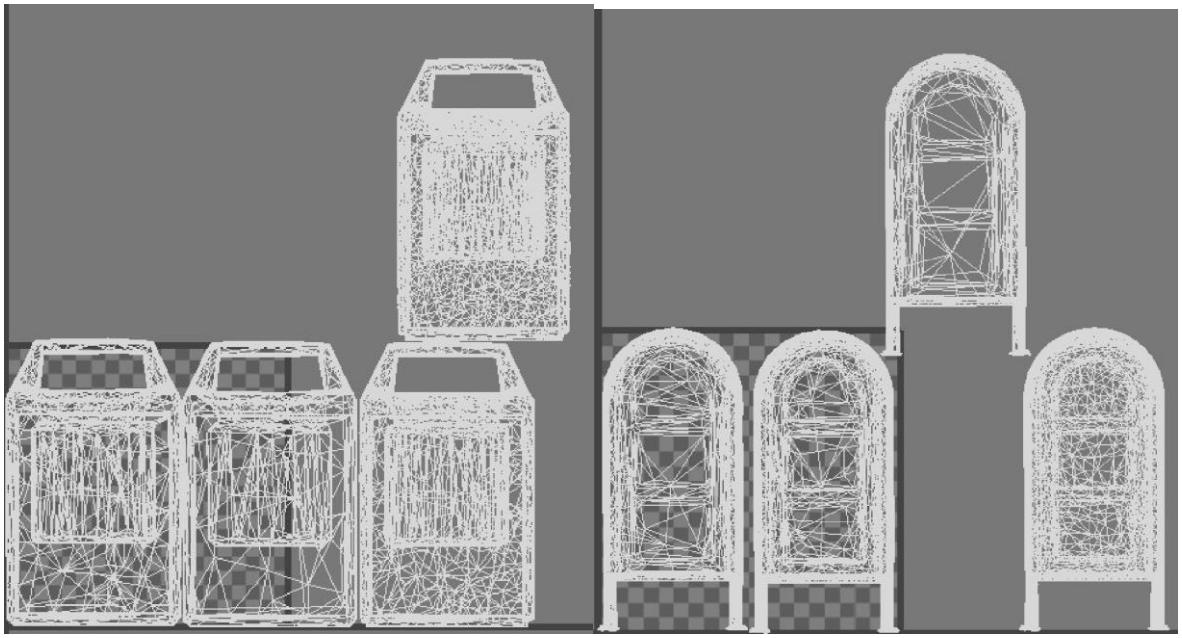


After removing the cars and all unwanted information we build our mesh. This process refines the pre-existing dense cloud and builds the final faces of our mesh. For importing it in engine we will have to import it first in 3ds max to clean it and then we will proceed exporting it in-engine.

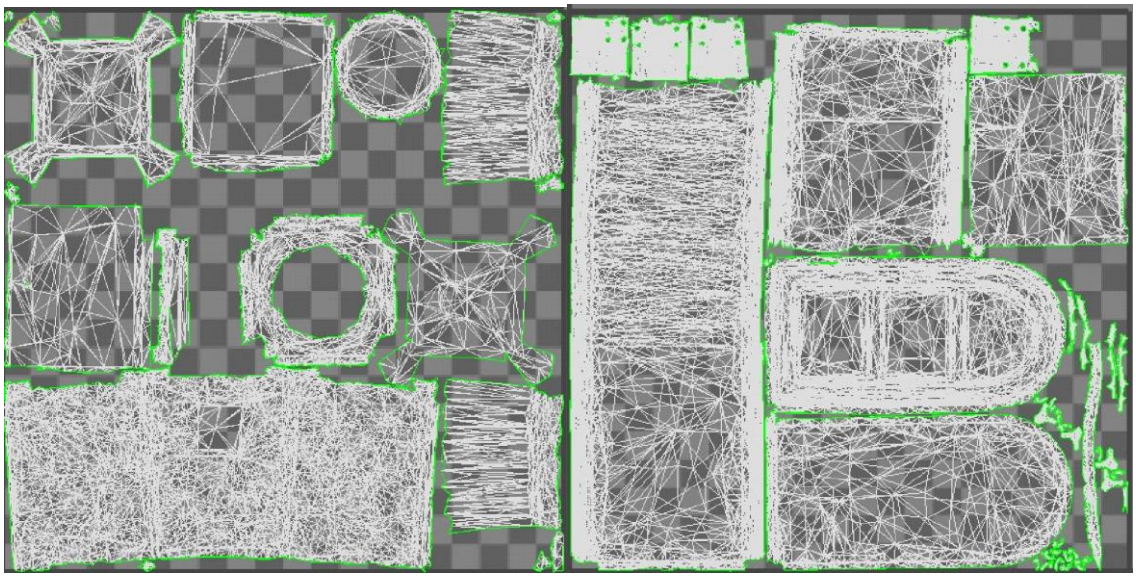
UV mapping

UV mapping is the projection an image to a 3D surface do a texture can be applied after. UV polygons then are painted with color from a texture image. The UV mapping includes assigning pixels in the image to the surface, this happens by triangling the image into a map and pasting it onto an image on the object. UV works like projection mapping stores a texture in space rather than into the geometry of the object. In the rendering the coordinates indicate the direction of the paint based on three axis.

For the objects in our scene, we will once again use one of the modifiers from the modifier list of 3ds Max to unwrap the uv of some of our objects. Adding mapping coordinates to a mesh, the modifier used is the UVW map and controls how the material is wrapped and appears on the surface of an object. With mapping cords to a mesh, the modifier indicates how the material will appear on the object.

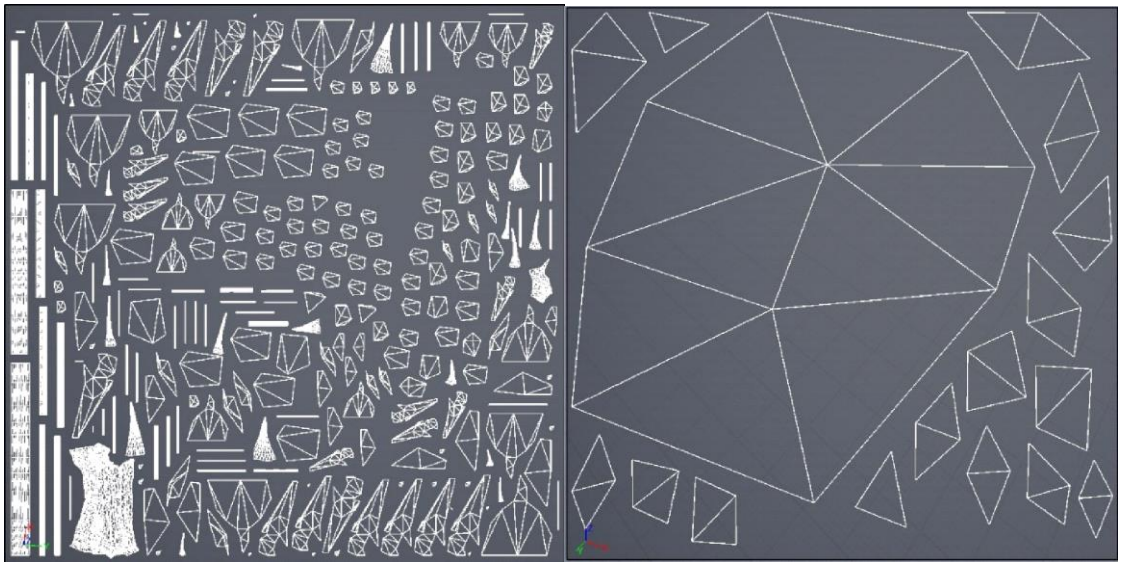


The above images show the wrapped surfaces of both the mailbox and the garbage bin. Since our objects have multiple surfaces around, we will need to break them down to simple shapes so that our materials later in the process can wrap evenly around them. This process must be done with every single object we have, and the material applied. UV mapping is essential for meshes most of the time. We will see how large surfaces like pavements and roads can have the same lighting effects and scaling options only though the applied material tilling settings.



Above are displayed both previous meshes and unwrapped forms. Looking back and forth at the wrapped and unwrapped versions we can see that the unwrapped version has selectively cut each surface of the object from the bottom of the bin to the sides of the mailbox. Also, even though object-like handles are very small in comparison and can be difficult to unwrap, we can replace them with a simple primitive shape like a tube and then unwrap it instead of the original shape. This is very helpful and cost-effective because primitive objects such as spheres and boxes have mapping coordinates, as do loft objects and NURBS surfaces. Scanned, imported, or hand-constructed polygonal or patch models do not have mapping coordinates until a UVW Map modifier is applied.

Vegetation uv can be more complex because of the physics of the objects. Complex multi surface models can be very difficult to do a uv unwrap to each leaf and each twig. So, the only possible way to unwrap a tree of any vegetation is part by part. First the bark and then leaves, if we have modelled one leaf like we mentioned in the foliage part of the paper we can unwrap essentially the one leaf we use with the random seed essentially unwrapping every leaf in the tree.



Materials

Materials are essentially the asset that we use to give color and sometimes shape to a given object. A material can have many properties that will define the look of our object and will dress it differently based on lighting conditions. Well structured materials can dramatically improve the look of a scene and give a better sense of realism or art based on the given content.

In more technical terms, a material will react based on the light source or sources bouncing on it and will determine the look of the object. The calculations are done based on a sum of images called maps or commonly referred as textures and math expressions, as well as variables based on connected nodes.

A material has certain attributes that define it and based on the input will produce more complex and better-looking results.

- **Base color-Albedo**: Base Color defines the overall color of the Material. The albedo is the color of the given material, wood is brown so the albedo will have the same color variation as the material in the real world.

- **Roughness:** The roughness of the material is first based on the real physical attributes of the object (marble will be less rough than a rock material). Roughness will determine also how light scatters on an object; a material with little to no roughness will appear shiny in contrast with a material with more roughness that will scatter less light on its surface.

- **Metallic:** Metallic like roughness is a scalar parameter that will indicate how much a material appears more like metal or less. A metal object will have a metallic value of 1, other materials based on their appearance will have values between 0 and 1.

- **Normals:** The normal map or bump map will give our materials the physical dimensions or properties of a given object. Normal maps help reduce a material's flatness and make it realistic.

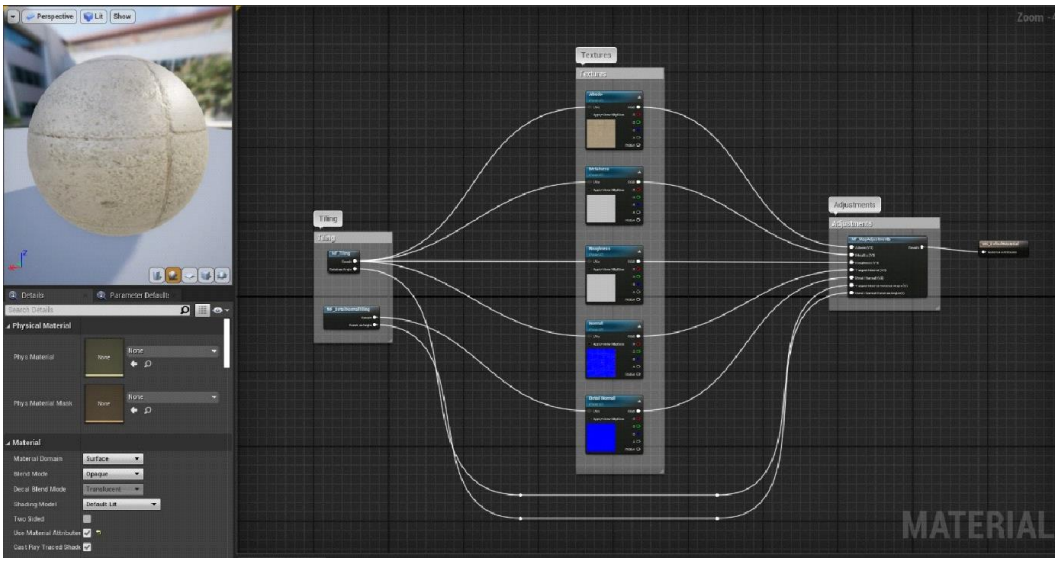
- **Specular:** A non metallic material will have more reflection and the map that controls that reflection is called a specular map. A specular map controls the specular reflection of a material making it appear less or more mirror-like.

- **Tessellation:** Tessellation helps a material to appear elevated by extruding its vertices based on the given map. This helps scenes with complex geometry like physical materials soil, rocks, etc.



Those are four of the material maps essential for creating believable photoreal materials in the engine. Materials with such properties are called PBR (Physical based rendering). Physically Based Rendering (PBR) is a method of shading and rendering that provides a more accurate representation of how light interacts with surfaces. It can be referred to as Physically Based Rendering (PBR) or Physically Based Shading (PBS).

To create a material in the engine we will have to import those maps and structure the material in the material editor. The Material Editor is a node-based graph interface that enables you to create shaders that can be applied to your geometry, such as Static and Skeletal Meshes, or with other systems such as Cascade to create interesting materials.

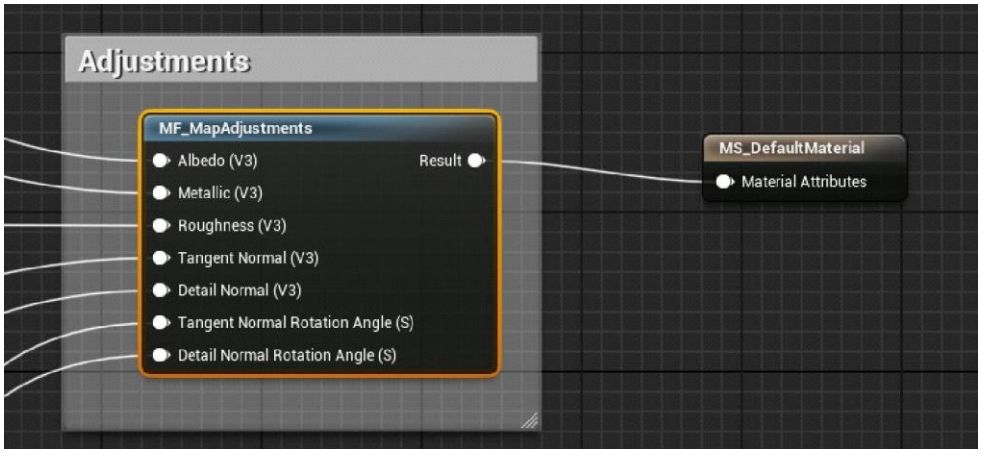


Each material node corresponds to a function that will allow us to link the given maps

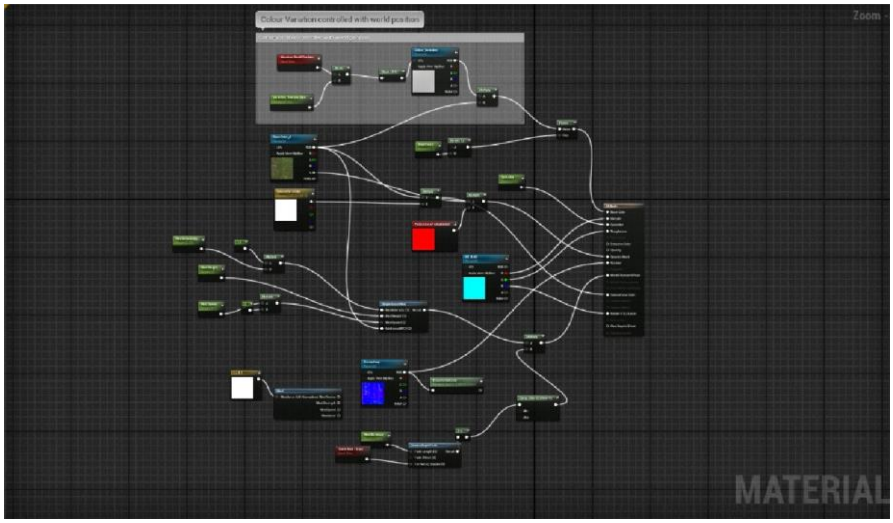
and create a material. Material Functions are parts of the material editor which include parts of code that can be reused on a material to give it different functions. By combining different material functions we can create new ones. Size of a function can vary based on how many functions are used for a given task.

Function like material nodes can be edited but in contrast they have strict restrictions to ensure basic function. A well crafted node tree can be beneficial on the scenes optimization later down the pipeline.

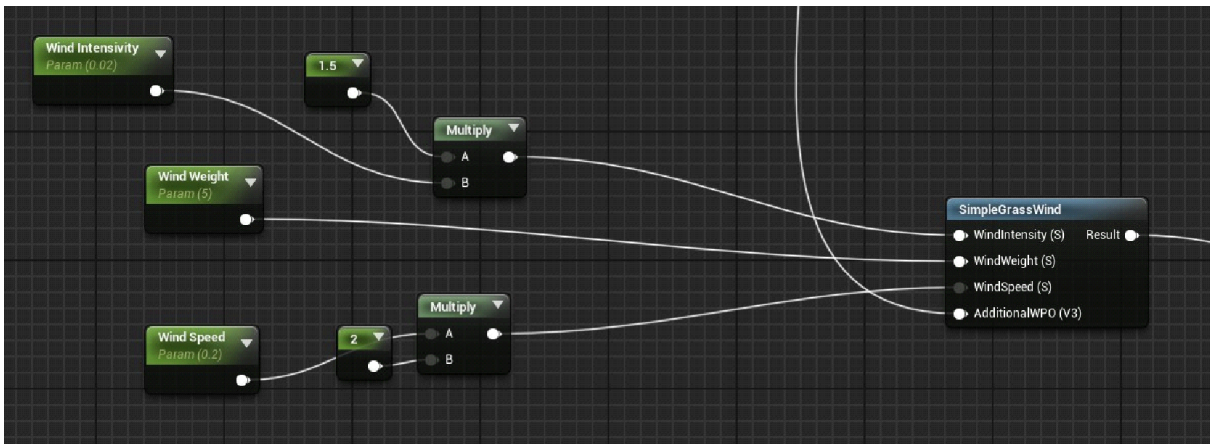
First, we will use two material function calls for tilling our material maps. One for our base color and roughness and one for our Normals. This will allow our material tilling to change in the future based on the applied surface. After connecting all given maps, we will again use a material function to connect all previous nodes to one function call before using it on a material.



The above material's low complexity allows its easy design in the material editor, but not all materials are as simple as simple tiling materials. Materials can also have physics, color changes and deformation. Foliage materials are some of those materials because of their physical properties. Furthermore, they have more variables for correct implementation on the photorealism side.

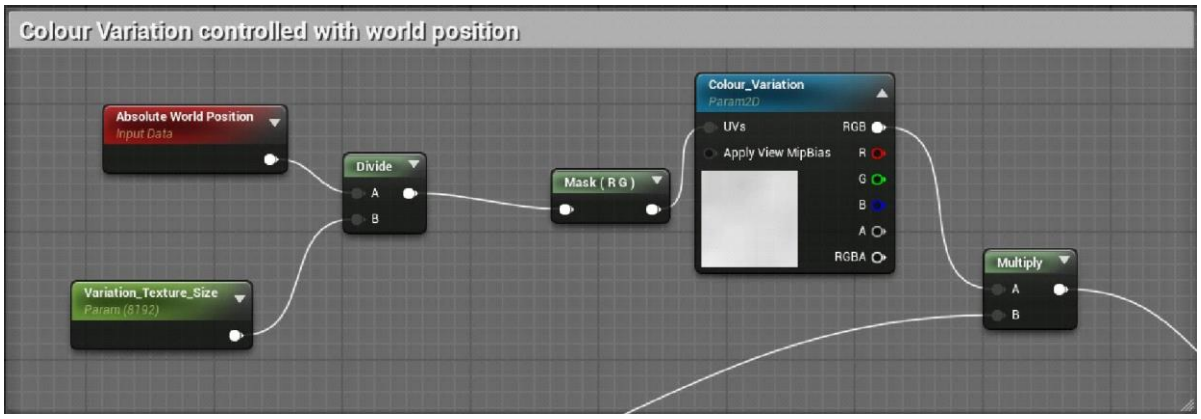


Based on the previous material node overview we can see that this foliage material is much more complex and has more material functions. The biggest changes are the physics-based function like wind.



GrassWind function will allow us to control an animation based on previous nodes which control the wind's intensity, weight, and speed. Those attributes relate to a constant or a value that can be decreased or increased based on what we want as a result. All the values then are connected to the main material function who triggers the event. Keep in mind that changing the material parameters from the material editor will be applied to every one of the objects using the same material.

If we want a material to apply only to a certain object, we will have to make a material instance only for the object we want. Material instancing is used to change the appearance of a Material without recomputing existing data. Typical Materials cannot be edited or changed without recompiling existing data, an instanced Material can be made to change without taking compute power from our system. Some instanced Materials can change during gameplay based from in-game events . This allows for visual flexibility in your artistic projects where we want to change materials in real time to help immersion.



Color variation is a different section on our material node panel that will allow our material color to adapt to the environment in every lighting situation without the need of a static light source.

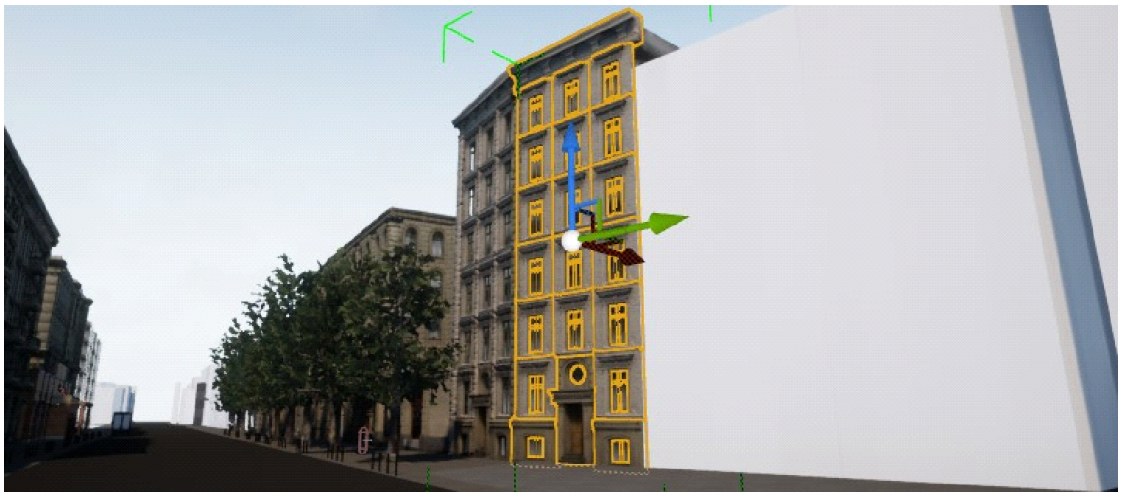
Placement

An **Actor** is any Object that can be placed in a level. When a new project is created, the level that is loaded it usually is blank with no object inside. To build the desired scene or populate the world, we must place **Actors** in the level. This can be done with two different ways, both use the **Content Browser** and results in a new instance of the Actor being created. Each actor instance in the level can be moved or have its properties modified.

Grouping

In the Unreal , we will use the grouping function to create big chunks of geometry that we will then use to build our scene faster. Grouping helps us not only to build the scene faster but change it later in development by saving time from changing every single piece of geometry. Grouping works in all geometry from little meshes to foliage actors and can be very beneficial in the optimization of a large scale environment.

A group can be assembled by different actors like a house with all internal furniture and meshes. This approach however can be difficult to change in the future and is good to use only if the objects have a definitive position in the level. Things such as street lamps with certain placement between one another.



Mobility

The **Mobility** setting controls the movement of an Actor when in gameplay. This applies to Static Meshes, Actors and Lights. When available based on type, the Mobility property has 3 states:

For all three stages terms such as light mass, lightmaps, indirect lighting, volumetrics and shadow maps will be mentioned. Those terms will be thoroughly analyzed in the lighting section of the paper since they involve lighting.

Static:

This mobility type for Actors that move or update when in gameplay.

- Static Mesh Actors will cast shadows that affect the already calculated lightmap using unreal engine's Lightmass. This mobility function makes

them ideal for architectural or decoration purposes and won't change during gameplay.

- Light Actors will contribute to calculated lightmaps like static mesh actors. They will illuminate with indirect lighting methods to illuminate dynamic objects.

Stationary:

This movement type of Actors or lights that will change in playtime and won't move.

- Static Actors, change but not move. They don't affect pre-existing lightmaps and behave like Movable Actors on the lighting part. When a Movable Light is present in the scene, they'll use a stored Shadow Maps for the following frame when the light source is stationary, this improves performance when using dynamic lighting sources.

- Light Actors can change during gameplay by changing their color or the intensity of emitted light to be brighter or softer . Stationary lighting can affect pre-existing lightmaps but they form dynamic shadows for most dynamic movable objects in a scene.

Movable:

The mobility type for Actors that they need to change during gameplay.

- Static Mesh Actors will generate a dynamic shadow that won't cast any pre-existing shadows. When lit, they will use indirect lighting. For Stationary lights or Movable mobility, they will only use real-time shadow.

- Light Actors can now cast dynamic shadows. Furthermore, being able to animate the light when in gameplay and change color or intensity as well.

Districts

To distinguish the level to multiple areas, like in real life, we will dedicate each area with a specific architecture. This like street meshes will not change since signs are universal no matter the location.



District 1 will have different architecture in terms of buildings layout and color while district 2 will have a more conservative color palette. Architecture and character are the main border between area 1 and 2 and it is the main transitioning point between them. The difference between the two in terms of style is one of the many practices an art director guides the player through and environment, making each area more distinct and more memorable.

The same treatment will apply to the park. Each Park is divided in parts. Playground areas, areas people can eat, bench areas and table games areas. Studying each environment and all the options that can create realistic solutions for variation making the environment grounded and not boring in terms of gameplay.

Placing vegetation

For creating the park portion of the map, we will need to place vegetation around the park area, this can prove a difficult task since vegetation has more variation and it is in larger numbers than simple static meshes. Also, optimization is a key component that will allow the environment to function properly so lodding and instancing is something we will use but analyze in the optimization section.

For achieving easy and fast placement on the environment we will use the foliage tab in unreal engine. The image below shows the active foliage groups in this case three different types of trees. Each one of those groups have certain attributes like density that it will apply to the environment when we paint over it.

Density depending on the type of foliage can be at high numbers or in lower. Higher numbers of density are used in grass assets since grass covers a big surface of the ground. In the other end of the spectrum lower density is used for assets of big scale like trees. This happens first for reasons we see and for optimization.

Also, for variation purposes a random rotation and random scale need to be applied to create a landscape with variation. Fifty



Lighting

In this unreal project we will use certain lighting techniques to give a sense of realism to the environment. This depends not only by correctly placing lighting but also on the lighting effects properties that filter lighting in certain ways like the atmosphere.

At first the main three types of lights used for external environments are three.

Directional light (Sun)

The **Directional Light** is a light that is being emitted from a source that is far away. All shadows will be parallel, making this a perfect for simulating sunlight.



As seen, directional light (sunlight) creates hard shadows. Directional lights have variable properties that will produce different results .

- Intensity: Total value of the emitted light.
- Light Color: The color value of the emitted light .
- Atmosphere Sun Light: this Light exists to define the location of the sunsphere in the sky.
- Affects World: Will disable the emission completely.
- Casts Shadows: If the light in the scene casts shadows.
- Indirect Lighting Intensity: Increases the light's contribution in the scene.
- Min Roughness: Is used for softening specular highlights.
- Shadow Bias: Is the accuracy of the shadows from this light..
- Shadow Sharpen: It sharpens shadow for this light.
- Translucent Shadows is the ability of the light to cast dynamic shadows through objects with low opacity like glass, water etc.
- Dynamic Indirect Lighting: If the light must be added into the Light Propagation Volume.
- Static Shadows: If the light casts static shadows.
- Dynamic Shadows: If this light casts dynamic shadows.
- Translucent Lighting: If the light affects translucency.

Skylight (Ambient sunlight)

The Sky Light captures parts of the level and applies that to the scene as a light in the form of ambient lighting. This means the sky's appearance and its lighting and reflections will match the scene, this method includes skies with different lighting conditions. A cubemap can also be used to light the environment.

Compared to other lighting types (directional lights, spot lights) skylights are much more complex structures since they simulate many lighting conditions happening in real environments. That is why the creation of skyboxes can happen though multiple ways.

Stationary skylight:

A Sky Light with Stationary Mobility behaves like a stationary light baking shadows

from Lightmass. Once placed in a Stationary SkyLight, light must rebuild again to bake shadows. We can change the sky light as we want without rebuilding the lighting. The Sky Light bakes store directional occlusion information in what is called a cubemap. A cubemap is an image that includes lighting information that will be applied to the skylight and to the environment.

Movable skylight:

A movable Sky Light doesn't use any computation before hand. It will capture lights of any mobility in the scene.

Precomputed skylight:

When using baked lighting with a Static or Stationary SkyLight, the light's and shadow's data will be stored in lightmaps. The parts below discuss some of these features.

HDRI:

A High Dynamic Range (HDR) image (360 image) is a technique used in the past for pre-rendered purposes and now in real time to simulate realistic lighting and reflection caused by the environment. This map is used in a backdrop format in the scene and will automatically determine the position of the directional light (sun) and ambient light caused by the skylight. The plane under the object will work as the ground and act as a shadow catcher to prevent lightning from bouncing under our scene.



Each type of light is different, options like HDRI are more accurate since its captured from real life environments but are cost intensive to the engine. Simple skyboxes are more efficient but to replicate lighting like in the HDRI it will take much more testing not only in turns of testing but it the Lightmass building.

Skyboxes or HDRI is one of the most crucial pieces of lighting if photorealism is taken into consideration. It is important not only for filling the scene with lighting but also providing reflectivity on all surfaces.

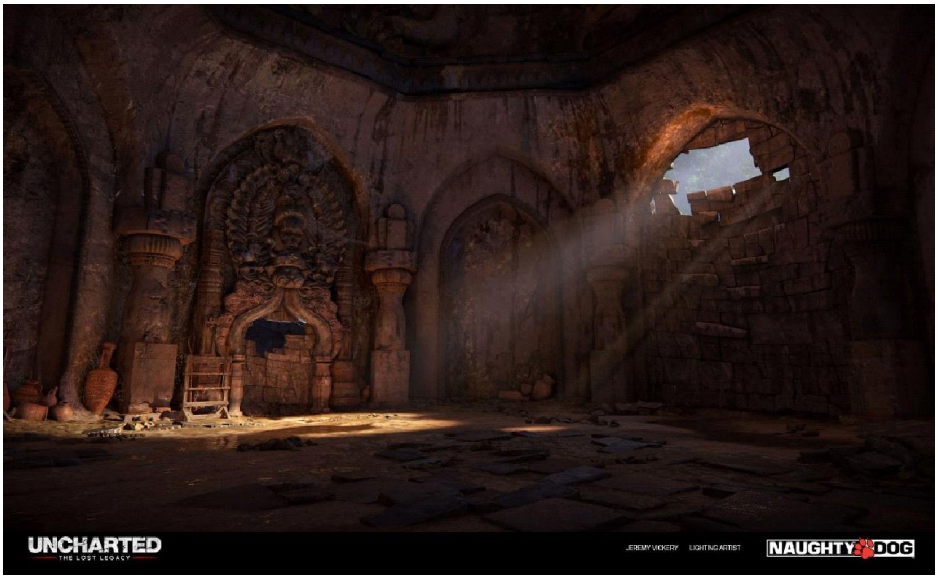
Spotlight

A Spotlight produces light from a single point and emits it in a to cone shape. There are two cones to shape the light - the Inner Cone and Outer Cone. In the Inner Cone, light emits bright light. The inner radius extends from the Outer Cone, a falloff takes place, creating a soft-edged cycle, around the Spotlight's disc. The radius of the light will define the length of the cone. Simply, this will work like a stage light.



Spotlights are commonly used in night scenes and interiors. For this paper using a spotlight will be kept at a minimum since for each spotlight we place our lighting budget will get even higher. The use of spotlights in exterior scenes are in areas that our directional light or skylight are unable to light such as tunnels or under bridges.

Other uses of spotlights are to hint the players eye to follow a certain path. This can be seen in contemporary video games. The picture below shows the player where the need to go but hinting it with light.



After exploring all types of lighting, we will explain basic concepts of lighting techniques and lighting properties on meshes. Furthermore, we will analyze state of the art in engine techniques.

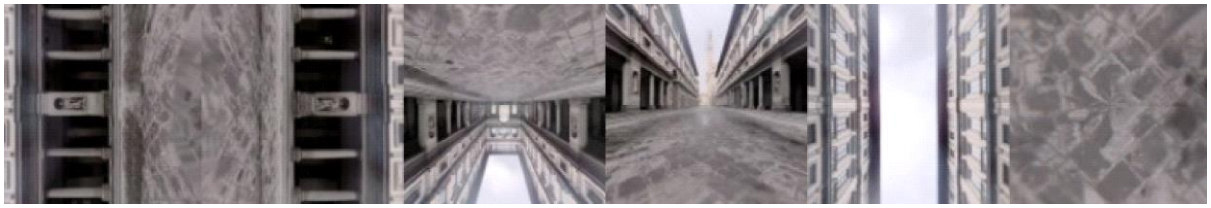
Cubemap

Cubemaps is a simple method to map environments, elements like distant scenery a common example are skies and surrounding environments like trees can work as a panoramic texture. Then these textures will store as a series of images wrapped to the inside faces of a cube.

Creating a Cubemap is a series of six pictures that can be taken or generated. They represent the view of each direction of the X, Y, and Z axis. These images after taken will need to be properly assembled.



Images are rotated straight up and down. Some need to be rotated to line up. In order for the cubemap to work each image has to be rotated in 90 degree increments clockwise. This happens because the orientation of an object such as a human can be different between the view and the orientation of the body. The body can be laying down but the view of the person can be looking up.



Volumetric fog

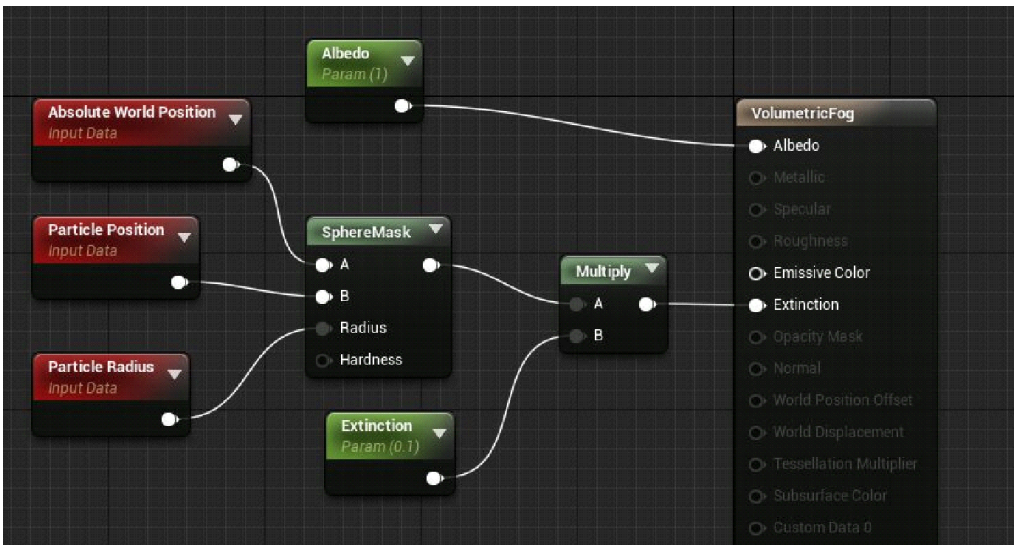
Volumetric Fog is a fog element that will affect the scene visually and in terms of lighting, volumetric fog can be controlled globally or locally in your scene. Those controls enable controlling of the fog for the entire scene. Local controls enable you to control fog from a particle system in areas where they are spawning particles.

Characteristics of volumetric fog:

- **Scattering Distribution:** This variable changes the direction of the volumetric scattering, meaning that light scatters equally in the scene, while a value of 0 will cause light to remain the same meaning that it won't change it's characteristics in the scene, values closer to 1 will scatter light.

- **View Distance:** This is the distance that the Volumetric Fog will compute. The texture volume for the fog is limited to the Z-axis and also depends on the distance. Increasing the distance will under-sample and cause visual artifacts. The number of Z slices will increase the quality, but will be more expensive.

Making a volumetric fog is simple since no material maps are required. Albedo ranges from [0-1], while emission and extinction densities have a value greater than 0.



The image below is showing before and after of the environment with and without fog. As seen, the environment with fog has a much more realistic feel than the one without it. Volumetrics must be used with caution mainly because volumetrics are GPU (Graphics processing unit) intensive.

Volumetric atmosphere fog



As seen in the images the use of volumetric fog fills the air creating a biggest spectrum of contrast between surfaces facing the sun and those placed in shade.

Reflections

The reflection capture system works by capturing points of reference or objects of the environment and then reflecting into a nearby surface or material. In the real world reflection will appear on reflective materials in the scene like water and glass, mirrors are excluded however and will require planar reflections to work. To capture the environment's reflections we will need to use a reflection capture sphere to indicate the radius of the environment we want to capture. In this paper we will be mentioning all types of reflections like screen space reflection and ray traced reflections.

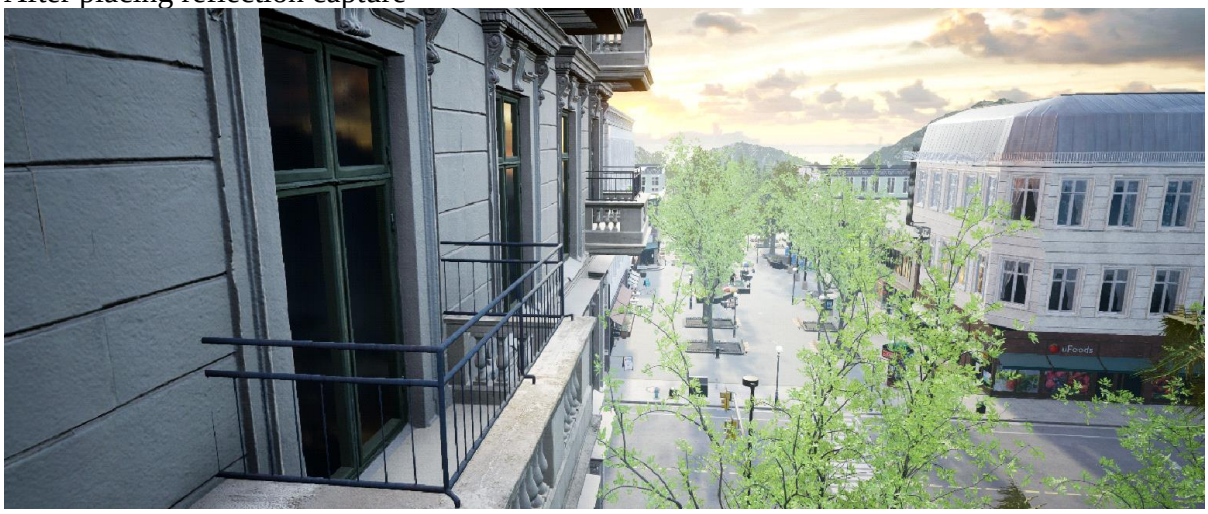
Before reflection capture



Reflection capture actors will capture all surrounding objects and cast their image onto a reflective surface. Each reflection capture actor has a radius that will indicate the objects we want to capture, the placement of reflection capture actors has to be precise in order for the right object to appear in the right reflection.

If we place a large reflection capture in an environment then what happens is that reflections will display objects that are far away from this scene breaking the immersion. By placing multiple reflection captures in the right spots however will help the environment reflect the right parts in specific reflections, keep in mind that too many reflection capture actors can cause optimisation problems.

After placing reflection capture



These are some rules to keep in mind when using reflection capture actors:

- **Screen Space** - Each of the Reflection Capture will cost more depending on how much of the screen is covered by its radius. This way, they behave like dynamic lights. This requires the rdii of those actors to be kept at a minimum.

- **Overlap** - Reflection Captur can overlap, this will increase the cost of reflections caused. This increases the cost even more if the rdii of screen space

is large, this will cause the environment to become performance intensive to cover your scene with high-radius Reflection Capture Actors.

Screen space reflections

Screen space reflections are a function that will aid grounding of objects on flat reflective surfaces like walls or the ground and blend with the results of the Reflection Environment.

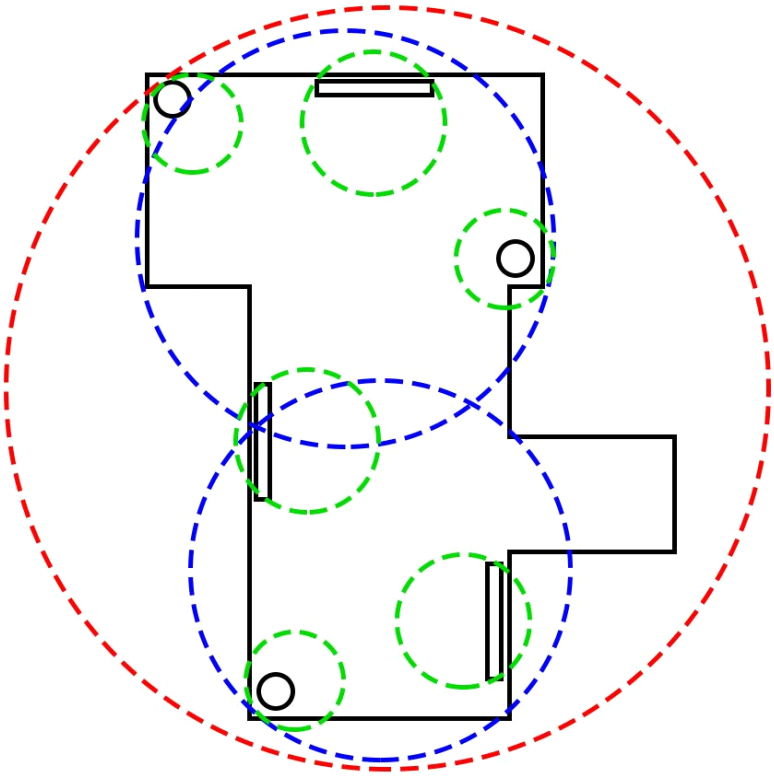
Reflections tend to be more pronounced and apparent in areas where there is a strong presence of light and shadow contrast. Exactly like the real world, reflections will tend to "pop" in environments that are darker in and of themselves but they have harsh lighting conditions. Therefore, the street scene used in this example was selected. Reflections can be more visible in areas where we see water paddles in the ground especially when a shadow overlaps it.

To get reflections right is to place Reflection Capture Actors across your scene. A common method is to scatter Actors across your scene until all are bound to a nearby reflective surface. Rules of placing those actors are:

- **Screen Space** - Each of the Reflection Capture Actors will affect the scene based on how much covers the viewport by its radius. They behave similar to particles or dynamic lights. That’s why keeping a good radius value on reflection captures can save compute power.

- **Overlap** - Reflection Actor can overlap each other. This will increase the cost of reflections caused by the Actors. In relation to screen space, it may become performance intensive to cover the scene with big-rdii Reflection Actors.

- **Hierarchical Placement** - Saving resources and having a nice order of Reflection Actors requires the use of hierarchical layout. In this project there are multiple little reflection capture actors in order to get right reflections of each element in the environment. The figure below shows the same example in a room.



We will briefly touch on the latest ray-tracing lighting algorithms. Even though the engine gives the option to apply ray tracing to the environment we cannot apply it since ray tracing features are supported only by NVIDIA Gpus.

Ray traced reflections

Ray Traced reflective surfaces are produced from accurate environment reflection bounces. The image below displays a single bounce of ray tracing compared to more bounces. Using more than one bounces will create real-time reflections between surfaces that are reflective.



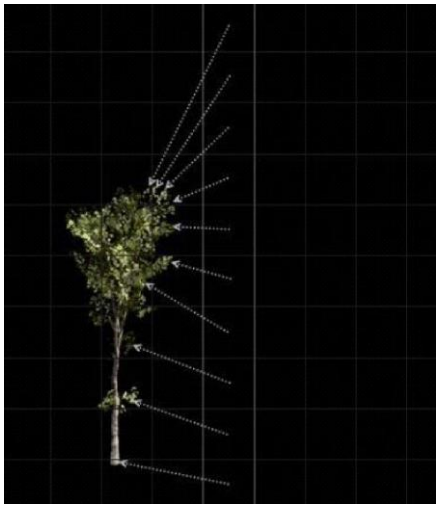


The difference between Screen Space, Planar, or Reflection Actors won't produce reflections dynamically and doesn't has a limit on reflection methods.

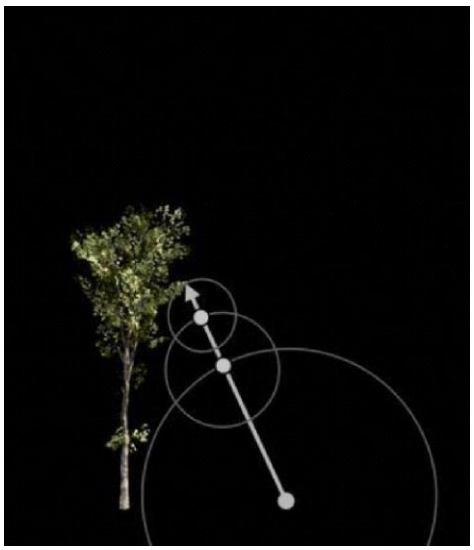
In those images ray Traced Reflections cast multiple bounces and have no limit to what we can see the sides of the book, the wall behind the camera provides additional lighting bouncing through the window.

Mesh Distance fields

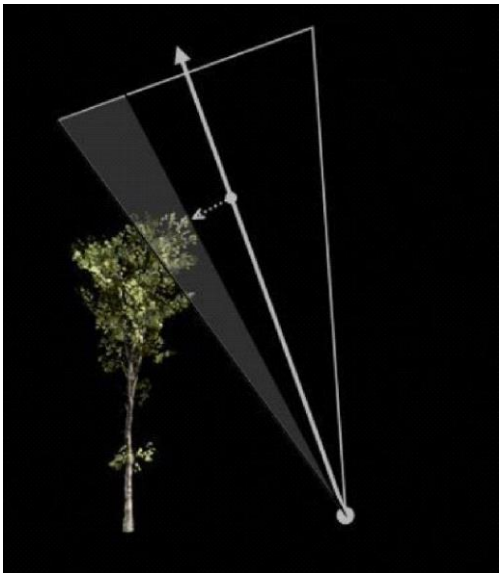
For Static Mesh's surfaces we use Distance Field. A distance field stores the distance to the nearest surface as a volume texture then every point on the exterior of the mesh is considered positive and any point inside is a negative.



The first property of tracing a ray's path is that we can skip through empty space relative to the distance of the nearest surface to the plane and the intersections are to be determined with a small number of steps.



The second property of Distance Fields is that when we trace a ray path we do it by tracking the closest ray that passes from inside an occluding object, the intersection can be calculated with no additional cost. This allows it to produce soft area shadows along with sky occlusion using distance fields. This function is instrumental to DF Ambient Occlusion a small number can compute an entire hemisphere from a receiver point.



We use mesh distance fields to improve the quality of shading in our scene. Enabling mesh distance fields helps with the contrast levels along with more detailed shading. Building mesh distance fields is a function mainly used in ray-traced scenes and can be very expensive on the GPU side.

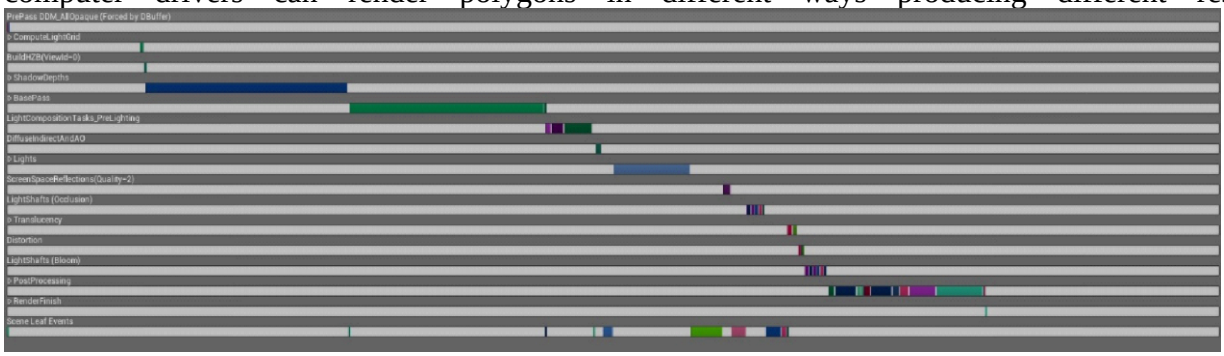
Optimization

Before going into the engine, we will explore general concepts of real time engines, specifically Unreal engine 4.26. Those techniques will later help us at the optimization of our scene so that it can be playable by every user.

To understand how much we need to optimize our environment-level we will use UE4 profiling tools.

GPU visualizer

The ProfileGPU allows us to identify the Graphic Processor Unit cost of passes, down to draw calls. The data are time stamped in the Gpu with accurate results. Optimization in the scene can drastically increase and decrease latency statistics in the dashboard. This is of course not always the case, some computer drivers can render polygons in different ways producing different results.



The above tab shows us each part of our scene that takes resources from our GPU, each one of those sections are crucial to be kept at a minimum to prevent the scene from poor framerate performance.



The GPU profiler informs with real-time statistics all rendering categories. The stats are cumulative, so we can see the most important categories without going down through a tree of events. Shadow Projection accumulation all the projections for all lights in the scene. Their on-screen GPU stats gives us a breakdown of how the GPU loads when we run the scene. Measurements also change instantaneously when we change console variables, also modifying materials and re-compiling shaders. The GPU will store a file the scene analysis at the end of the runtime.



The above graph shows us how our scene behaves in real time. The more even the graph, the better optimized it is. This does not mean that the graph will always show us an even pulse, some scenes have vegetation in which case the graph will spike. This is something that in the context of the scene makes sense since vegetation is heavy on the gpu.

Instancing

Instancing, a useful feature of unreal, is also referred to as proxies or instances in other programs. In short, instancing produces multiple cloned meshe, the layout is controlled by the creator. These instances appear fully at render time but far the clones are just dummies in the number of resources it uses on your computer.

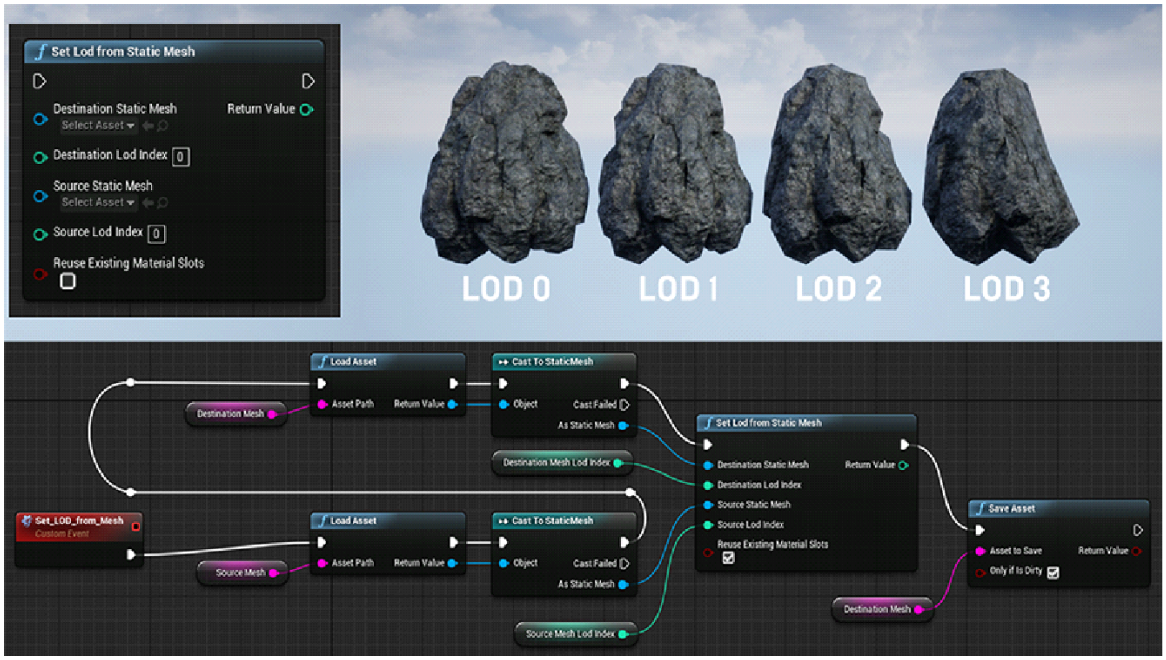
Objects with high complexity such as large areas of foliage can be rendered if we were rendering a single tree mesh. Instancing characteristics can provide benefits to an artist both in efficiency in scene and render time after building the environment, as it minimizes the load on GPU.

An instance needs a part of the RAM and other compute resources that an object requires. Render times will reduce drastically when we use instances in contrast to duplicate meshes, as rendered only must load a mesh only one time, this allows the main job is to calculate and display the light characteristics of a scene faster.

LODS

When close to a Static Mesh we placed in your scene, ideally the object looks detailed. Even though the mesh doesn't need to be detailed and complex if the player moves further from the mesh. No reason for having a complex and detailed mesh if screen percentage is too not enough to support the level of detail.

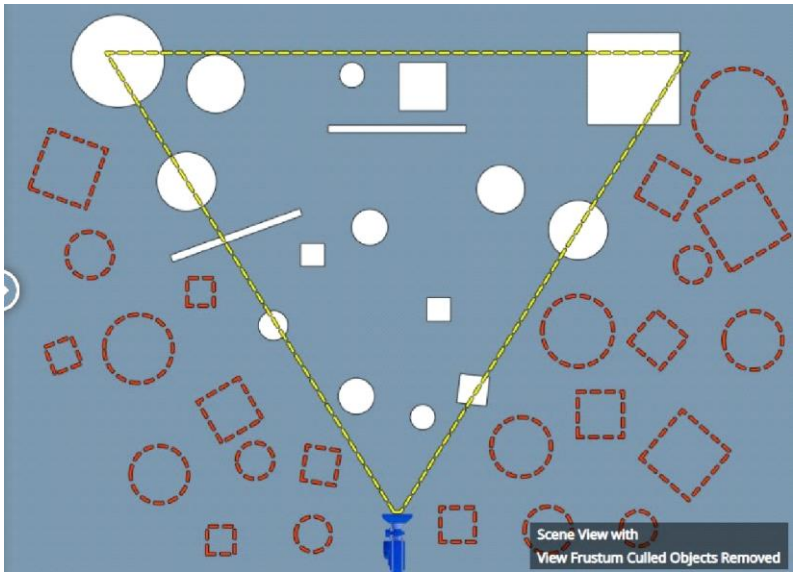
Furthermore, it needs to have high resolution so the player is close to the mesh and can see it in all detail. In Unreal, we can place an object in our scene that switches to a less complex mesh with fewer polygons as the player gets further away from it to make the scene perform even better. This is done using Level of Details or LOD for short**.** This mesh can seamlessly swap from one to the other as the player moves back and forth.



In most LOD is applied to geometry. LOD methodologies include shader management to manage the scene's complexity. A level of detail is applied to textures called mipmapping and also provides higher fidelity.

Culling

The idea of visibility or occlusion culling is reducing all objects at any time in order to achieve better performance. We start with what field of view the camera can see from its position, there are few objects visible, which is not the case because there are many objects that make up the scene that are not visible from this position.



Objects that cull outside the camera's FOV view frustum don't render, leaving only some of objects within this view that are related to objects to be checked for visibility. When this happens a series of data will be sent to the GPU to test for visibility of those specific objects. Those that are occluded by another are culled from view.

Culling works with object bounds. Each Actor is placed in a Level bounds using a box wireframe along with a sphere. Specifically to test what is visible and what not. The bounds are made of two parts: a sphere and a box. The sphere is used from the engine for fast collisions by testing the distance and it is larger than the object in it. The bounding box however is similar in shape to the object and provides accurate results.

Culling methods are divided into:

Distance culling: Distance culling methods per-Actor settings and Cull Distance Volumes, will render actors based on their distance from the camera.

- The Minimum Draw Distance is the minimum distance an actor can be rendered from the camera.

- The Maximum Draw Distance an Actor is the distance an actor can be rendered further away from the camera.

Forward shading

UE4 will use a Deferred Renderer because it provides flexibility and uses multiple rendering features. Forward Rendering enables more efficient base render, with faster/efficient rendering passes, which helps for better performance on platforms with low GPU specs. Forward Rendering also provides better anti-aliasing solutions than the Deferred Renderer, which leads to better visual fidelity.

A forward renderer is faster and can render r for most content. The improved performance comes from features disabled for each material. Only nearby reflections captured will not display parallax correction if the materials allow High Quality Reflections.

Forward Rendering works by culling lighting and reflection onto a space grid. Each pixel in the forward pass then changes over the lights/reflections affecting it with the material. Dynamic Shadows for lights that are stationary are computed and mapped to channels of a shadow mask, taking advantage of the existing limit of overlapping Lights.

All those element-techniques will provide a stable framerate along with an art quality of graphics. There are also small actions that will save time in terms of organizing the assets and saving core resources such ways are:

For Artistic purposes:

- Reduce the number of elements per mesh.

- Combine meshes to have an optimal number of triangles per element .

- Opaque materials are fastest ,masked materials slower, and low opacity

materials are slowest.

    - Limit UV plus hard edges as it leads to more vertices for the GPU. High polycount meshes with hard edges lead in more vertices in modelling applications.

- Large meshes can be divided for better culling.

- Smaller texture formats result in faster materials .

- Low resolution texture maps will render faster.

For Level design purposes:

- Reduce the count of Stationary or Dynamic Lights.

- Area lights are expensive and it is good practice to avoid them when the scene allows it.

- Adjusting the draw distance when the object on smaller objects to allow for better culling.

- Try to combine lights with a similar origin.

- Static lights are the fastest, Stationary lights are less optimal, and Dynamic lights are the costliest and difficult to optimize.

- Limit the radius and light cone angle to the minimum.

Navigation

Navigation is related to the movement of the player inside the world-environment. Essential for correct navigation are first collisions in objects. Buildings, trees, and objects that in real life can be traversed need to have collision boundaries, others like small plants do not need since a real human could walk through a small bush.

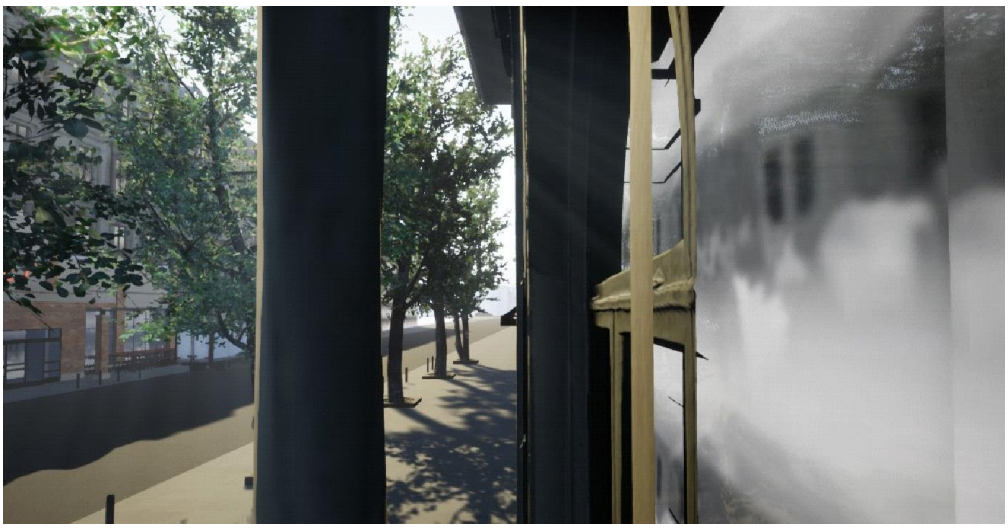
Collisions:

Collision Responses are the main way of how Unreal deals with collisions during run time. Every object that collides gets a series of responses that indicate how it will interact with other types of actors. When overlapping objects involved in the collision can be set to collide or to be affected by the collision, overlap, or ignore each other. Trace Responses work the same way, except the trace itself will be defined by the Trace type, giving Actors the ability to either block or ignore it based on the Trace Responses.

Objects with collision:



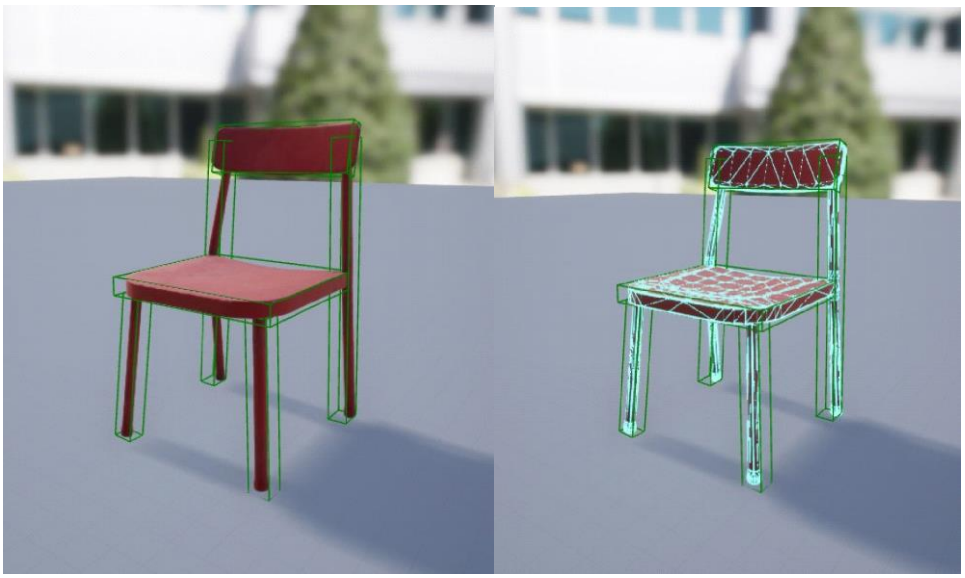
Object without collision (Clipping):



There are some rules to how collisions are implemented in the scene:

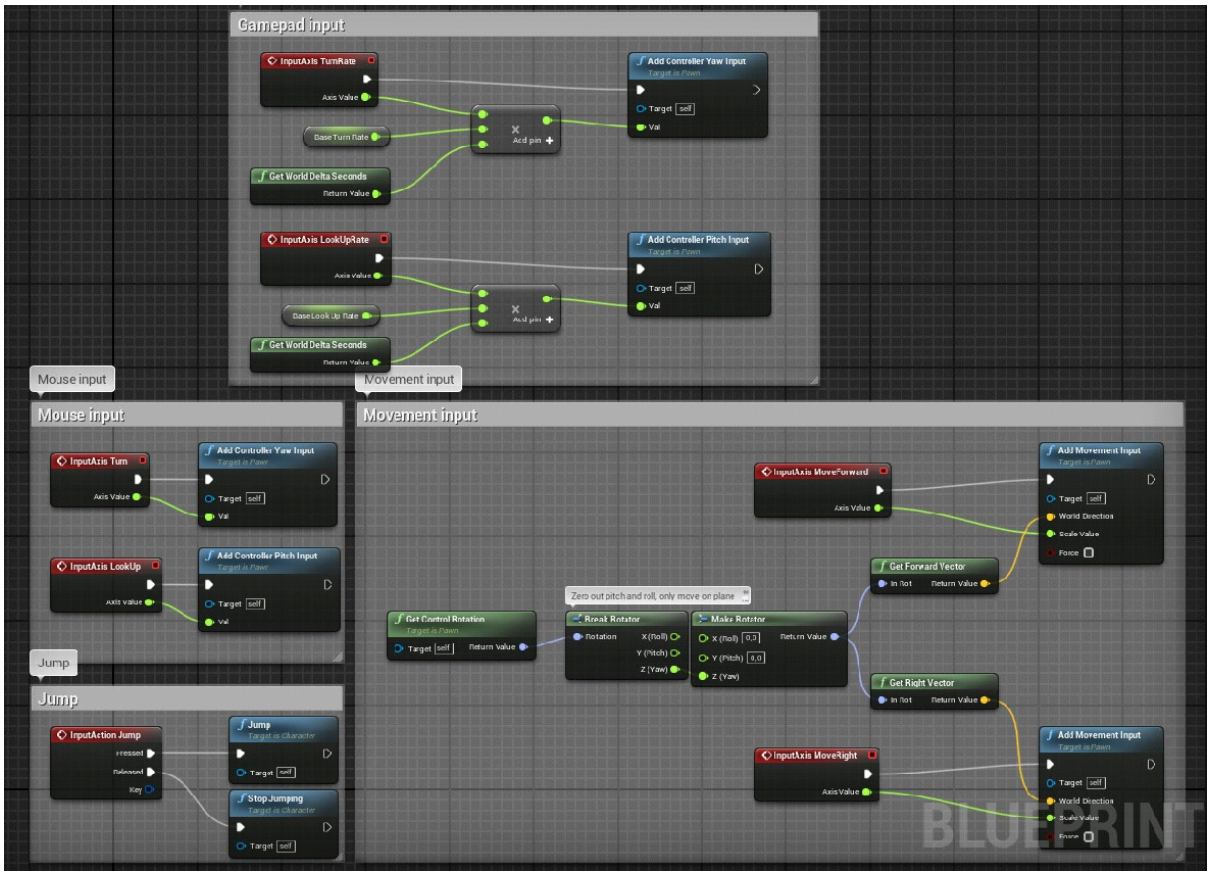
- Blocking will happen between Actors assigned to Block.
- Actors set to Overlap will ignore one another, without Generating Overlapping Events.
- For more than one simulating objects to block, both need to be set to block their respective objects.
- For two mehes being simulated: if one is overlapping an object, and the second object is blocking the other, the overlap will happen but it will not block.
- Overlaps can generate even if an object is set to block the other, especially high speeds occur.
- It is a bad practice for an object to have both collision and overlap.
- If an object is ignored and the other set to overlap, no overlaps will occur.

Collision bounds can be seen in unreal engines through the static mesh menu. Collision is divided into simple collisions and complex ones. Simple collisions can be applied to objects that cannot be interacted with by the player. Complex collisions are mainly used when we need to apply physics to objects that the player interacts with for more accurate results.



Core navigation:

Core navigation is referring to the main way of navigating through the level-environment. For this project we will convert a 3d person player to a first-person perspective camera. First to create a movable character we need to set some navigation blueprints so that keyboard input can be implemented in character movement.



In the above image we see four sections that are used to move the player. Mouse input will take in account things such as camera rotation. Jump is triggered by the space bar but for the need of the project we may need to remove it. Jump functions can be difficult to control and can give players the ability to pass though boundaries, intersect meshes and so on, that's why jumps functions are not used in navigation projects.

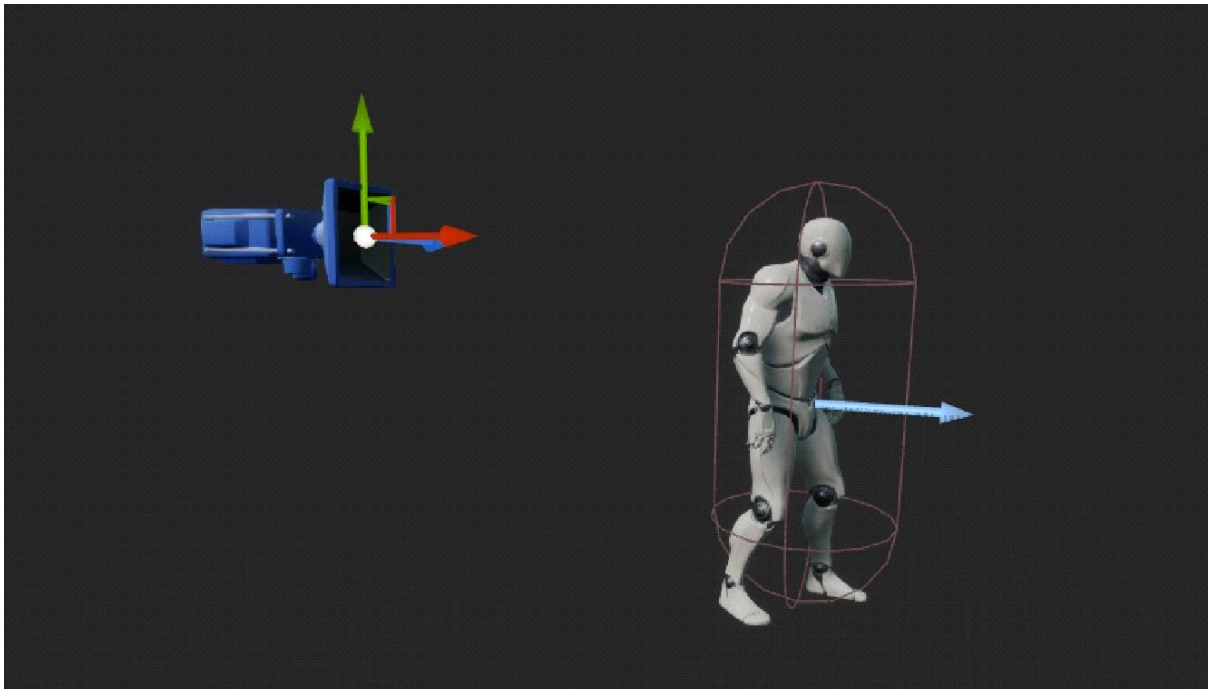
Movement input is the main way that the player moves through the level, mainly the WSDA keys for each direction forward, back, right, left accordingly. In movement we also need to consider movement outside the four axes to diagonal direction so if pressing the WD keys the player must move to forward-right position.

Gamepad input on the other hand uses the same principles as player movement and mouse input on a different topology. Gamepads movement and camera rotation are controlled by thumbsticks that do not have a key press like the keyboard. So, to move the player, we will use variables with increasing value to indicate the direction and the rotation of each stick. One of the cameras and one of the movements.

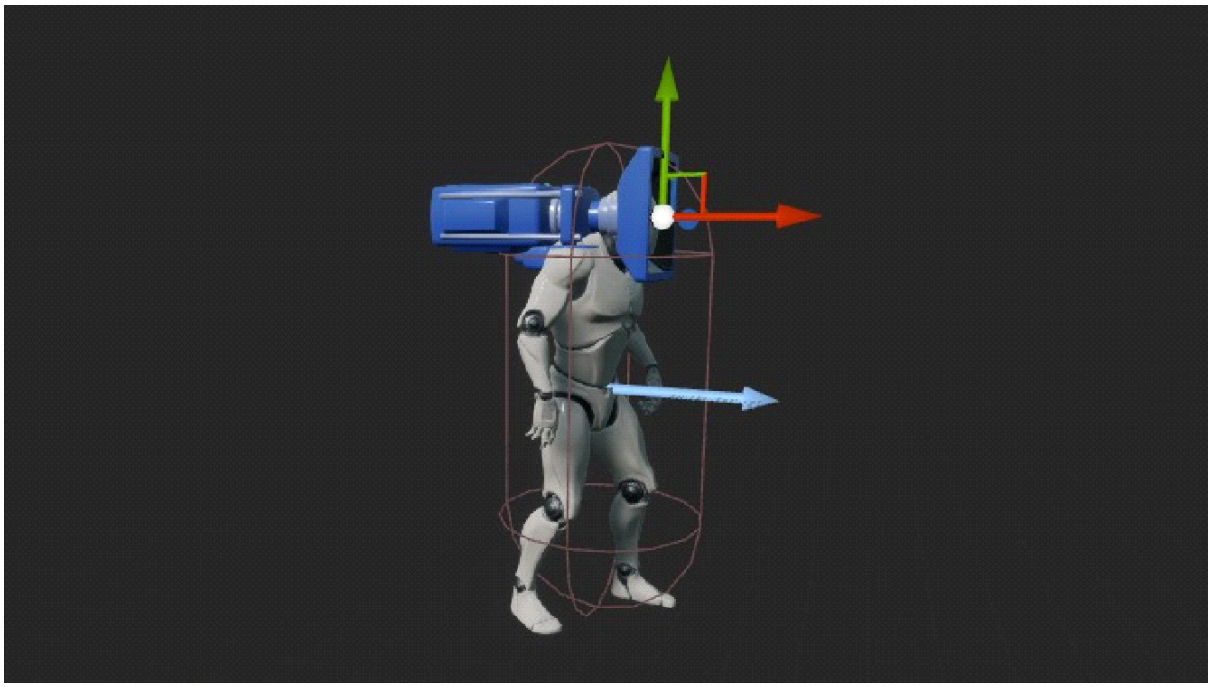
After we are finished with the blueprints menu, we will move to the camera placement on our character model. This step is essential if we want to accurately recreate the first-person perspective. Note that the first-person perspective we will use is not the first-person perspective we see in mainstream video games. The main difference is that to recreate a first-person perspective we will need to be able to see parts of the body like we do in the real world. This effect can be recreated from the steps below.

First, we will need to move the camera to a position we want. The first-person perspective is a camera on the head of the model so we will move it inside of our model. Because of clipping issues, we need to see if the camera is blocked by the head, this will take a few tests but after that we will be able to view the entire body without worrying about the head.

Camera at third person perspective:



Camera from third person to first-person:



After we set up the camera position, we will need to track it on the player's head. This means that the camera will follow and adapt to the model's movement and will not stay static during the playing state. For that to happen we will set the parent socket to the limb of our choice which is the head. The camera can be also attached in any other limb we choose, creating a new perspective.

After we successfully set up our character, we will take in consideration the acceleration that

he/she can traverse the environment. This happens due to memory constraints. Having a fast acceleration will most likely result in a buffer overload. This means that the engine was not able to load all necessary assets in the given time between point A and point B (start and finish point of the player walking).

Weather system

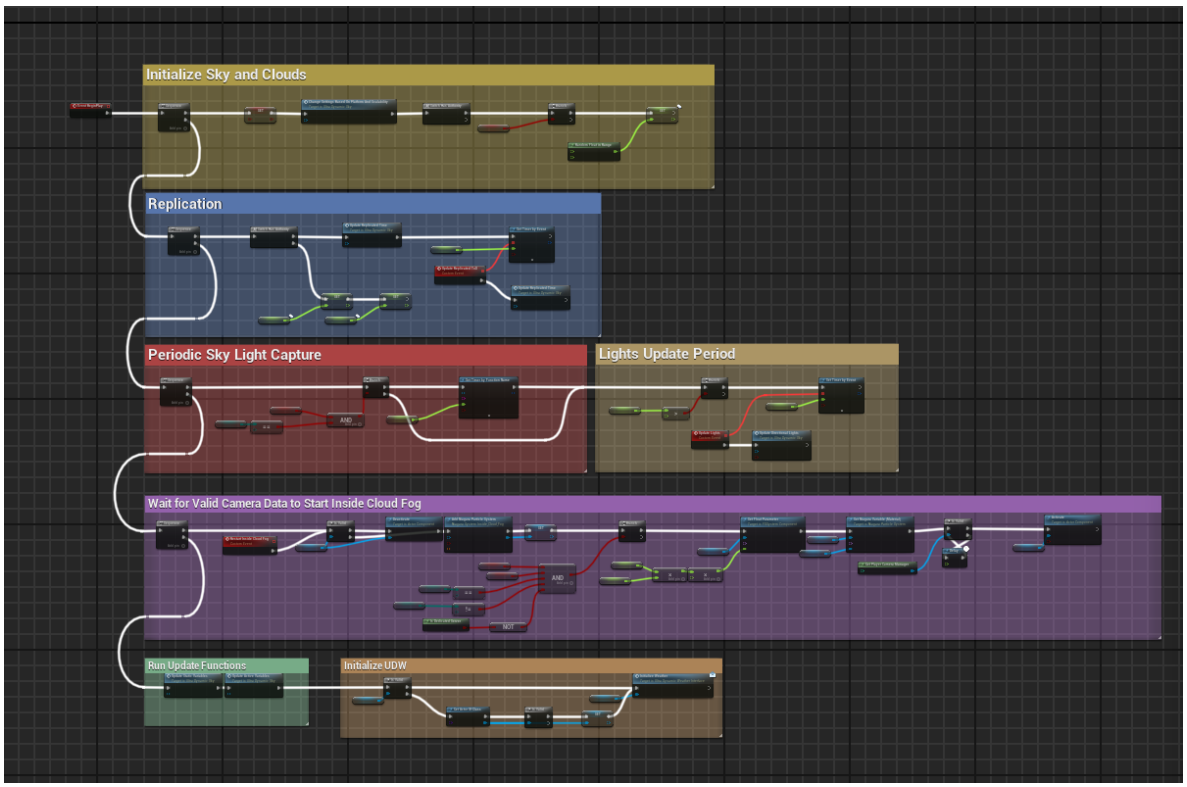
The goal of our weather system is to take advantage of pre-existing systems like unreal directional light and skylight and add additional elements to create weather patterns so we can have multiple weather conditions like rain, snow. Having those things are essential but also making them optimizable is crucial to the environment's playability.

In order to achieve good results and good optimization we will utilize unreal's Niagra system to create emitters for multiple particle systems. Those divide into snow and rain. Other weather conditions like foggy skies, clear or overcast weather can be created with skylight, directional light and volumetric cloud systems.

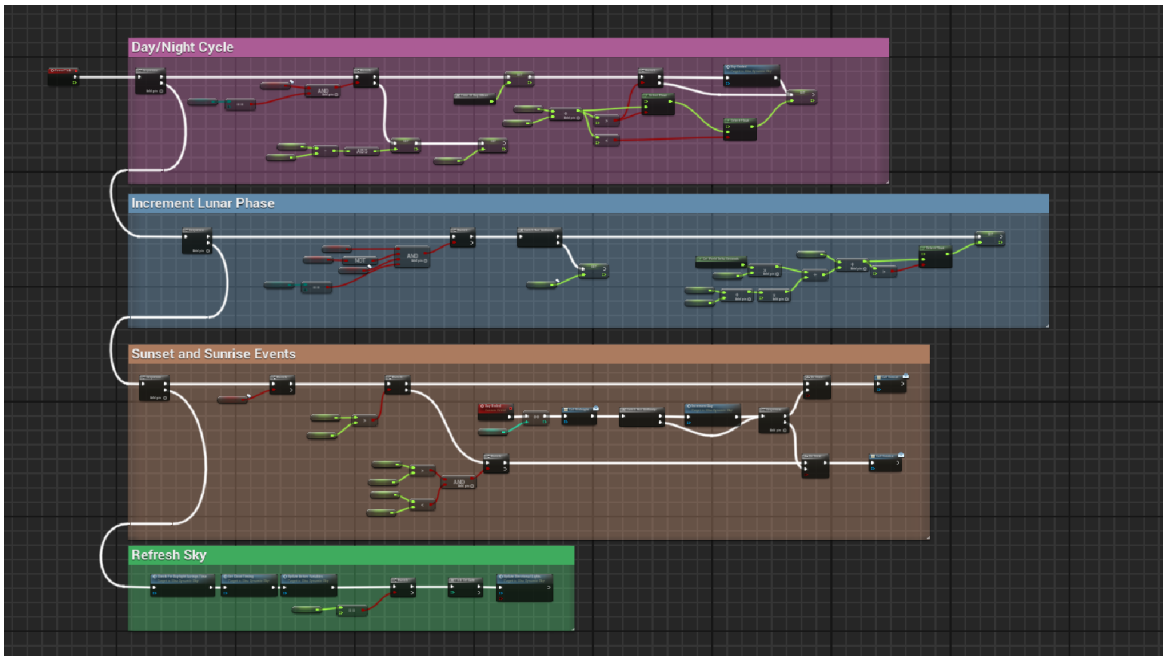
For existing functions in unreal (skylight, directional light and volumetric cloud) we create a blueprint called Sky and for rain and snow a blueprint called weather. Each blueprint is structured like a pyramid starting with simple systems and adding new functions on top of them.

Starting with the Sky blueprint we need to keep a few things in mind. Replication is one of those things that will extend our environment across the world. Sky captures in between changing weather conditions so that the transition of lighting is changed evenly without performance issues. Last but not least sun's phases like sunrise,sunset and night must be built first in order to co-exist with clouds and fog.

- **Initializing sky** and clouds are the first step since they are the base components of our sky.
- **Scalability** parameter so it's optimized for each platform that we will use this blueprint.
- **Replication** that we talked about previously is to extend the bounds of our sky to the ends of the playable world.
- **Periodic sky light capture** exists to recapture new lighting conditions that occur in the scene when we change it.
- **Lights update** period is the timing of the light capture update. Works like a timer for our lighting update.
- **Starting inside cloud fog** will create fog in the environment in many variants based on the preset we choose. This parameter is crucial to get lighting scattering right since fog will soften directional light sources.

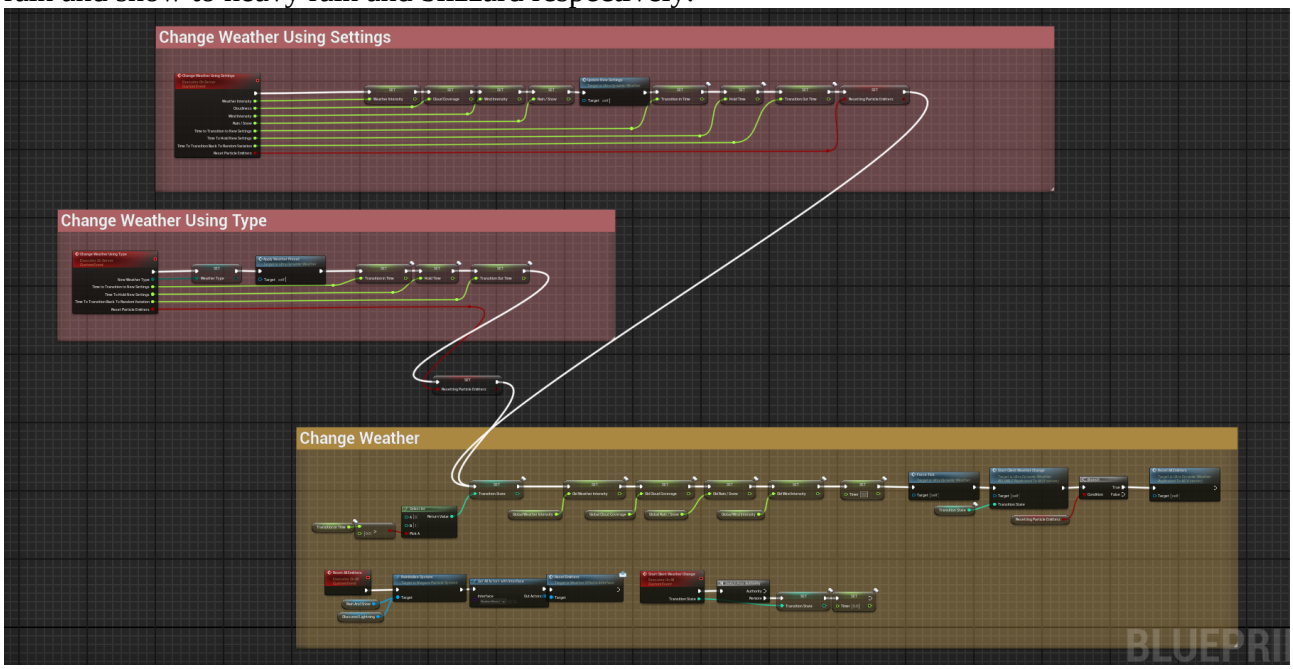


- **Day and night cycle** includes all trantintions of the directional light; the rotation of the sky based on the location of the level will contribute to the intensity and color of light.
- **Increment lunar phase** will appear if the sun is in 180 degree rotation to the level (night) and will display stars in the sky.
- **Sunset and Sunrise** events are displayed when the rotation of the sky is parallel to 90 degrees rotated from the level.
- **Refresh sky** works like a periodic light capture in order for colors and intensity of the light to be rendered in objects in the scene and also clouds.

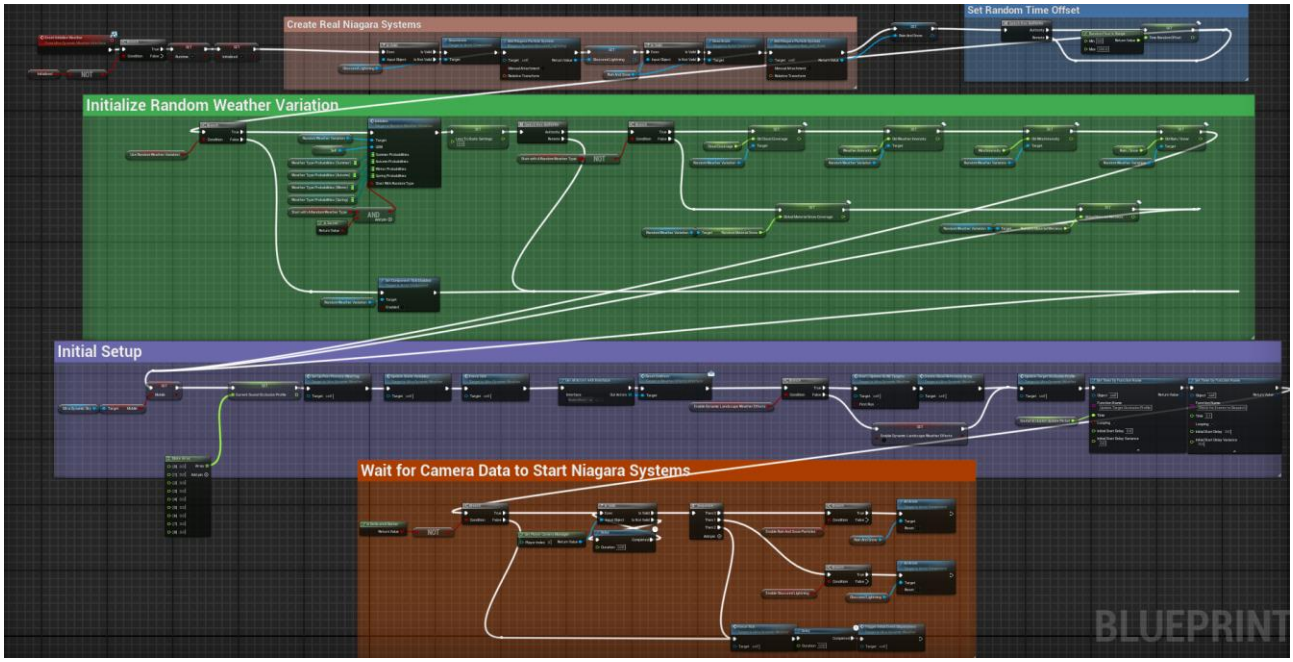


The second blueprint we have to create will work on top of our sky blueprint and will be named Weather. The weather blueprint will include all variables of the Sky we created and add new weather patterns such as snow and rain. In addition a wind variable will be created to double the effects of

rain and snow to heavy rain and blizzard respectively.



- **Changing weather using settings/type** are two ways to change weather patterns. Using settings is a more precise way to change the weather since each setting is bound to a scalar parameter. Using type is an automated way to produce a standard result of the given condition. Depending on how custom we want our scene we go back and forth between the two options.
- **Changing the weather** section helps the engine transition from old weather conditions to new ones. This happens in a few seconds transitioning from the old emitter to the new one if we use rain or snow.



Create Niagra systems will create our niagra spawners all over the scene to create the given weather position.

Random weather variation creates patterns based on weather conditions; this happens with branch function in between each one of them creating a complex system of interchangeable weather patterns.

Initial setup includes the wind variable mentioned previously in this paper and its the main variable that initialises at the start of the simulation, after that come all the niagra particle systems.