Cambodia in UE4

Mark Wiggers | Bachelor 2019

6th Semester
Truemax Academy Copenhagen
25/05/2019

Table of contents

Abstract 2

The birth of an idea 3

Referencing 4

Pipeline structure 5

Blocking 5

Terrain 7

Water 9

Running in real-time 10

Procedural methods 10

Workflows 12

Assets and texturing 12

This is Marvelous! 16

Trees 18

Lighting 19

Performance and Profiling 20

Menu 21

Cinematic 23

Packaging and Testing 23

What now? 23

Abstract

This dissertation examines the role of which an environment encapsulates the essence of what we perceive when we roam a virtual digital space. It will enlighten the phases that are necessary to create a breathing environment, that fulfills a balancing act between optimization and detail. Only through the meticulous work ethic of strict workflow and constant realignment with better and faster solutions, will the project take its form.

This dissertation will serve as an unfolding of my process throughout my different workflows as well as a dissection of what faults may lie within them. It will cover the project from an idea to its current state and also discuss the further possibilities to expand and evolve the environment through procedural methods.

The birth of an idea

In the initial phases of the project I made certain statements to shape an idea.

- I want to make a game environment.
- I want to run it **real-time**, in first-person view and with the ability to walk around and experience the nature of surroundings.
- I want it to reflect a detailed environment with both organic and non-organic entities.
- I want to achieve at least a constant of **30 FPS** on a normal to high-end PC.
- Ultimately I want to achieve a quality as close as possible to a **AAA** production.

First and foremost I wanted to create an environment with authenticity, something to be discovered and something to be felt.

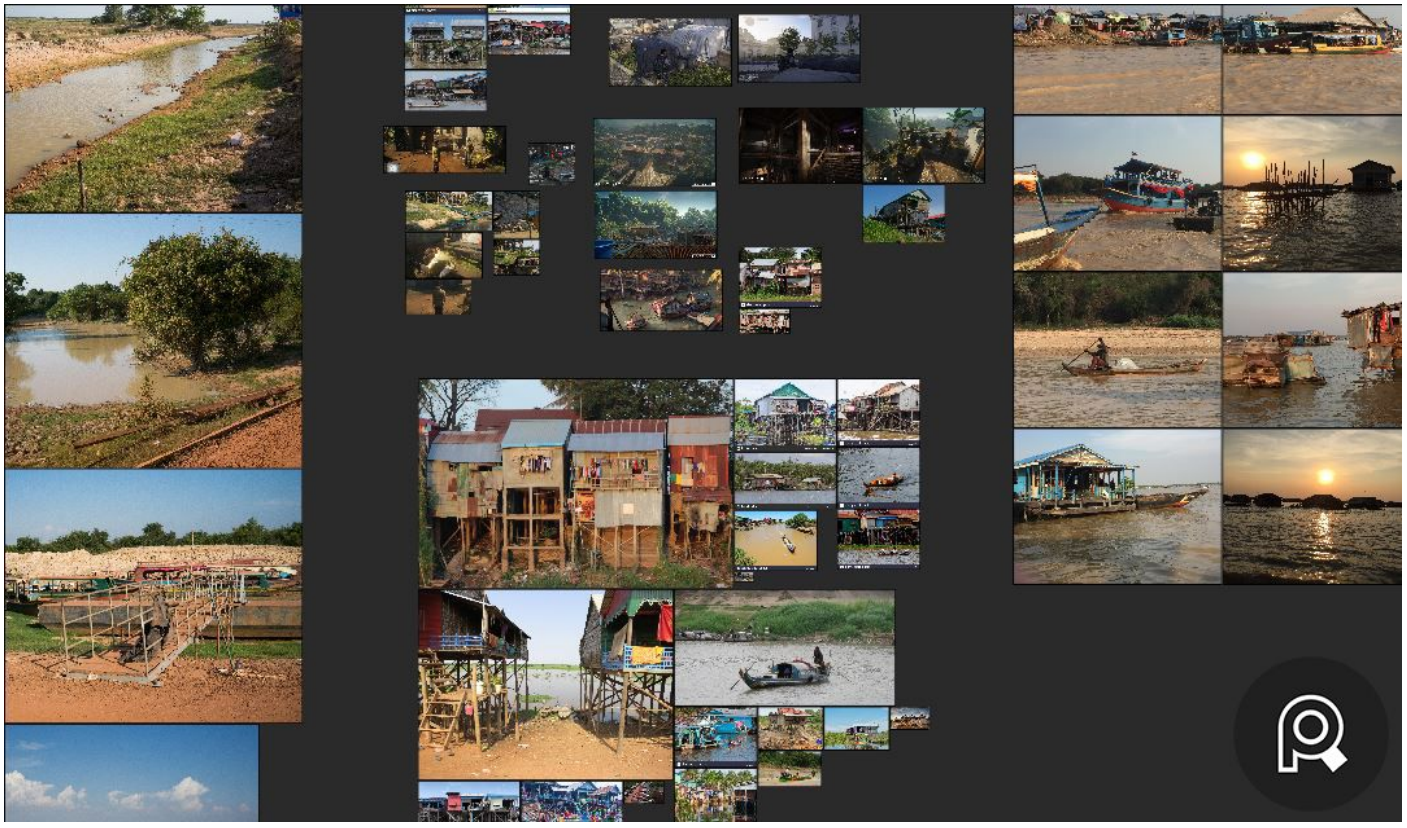
Cambodia

Robin Lhebrard is a concept artist from France. I fell in love with this drawing on Artstation and started a conversation with him. I told him about my project and he told me about his travels to Cambodia. He explained in great detail about the boat trips he made on the Siem Reap river to see the stilt and floating villages there on. He ended up sending me his personal photos from his trip. Later he redirected me to a photographer (Alexander Skold) on Artstation that made a compile of photos from that same area.

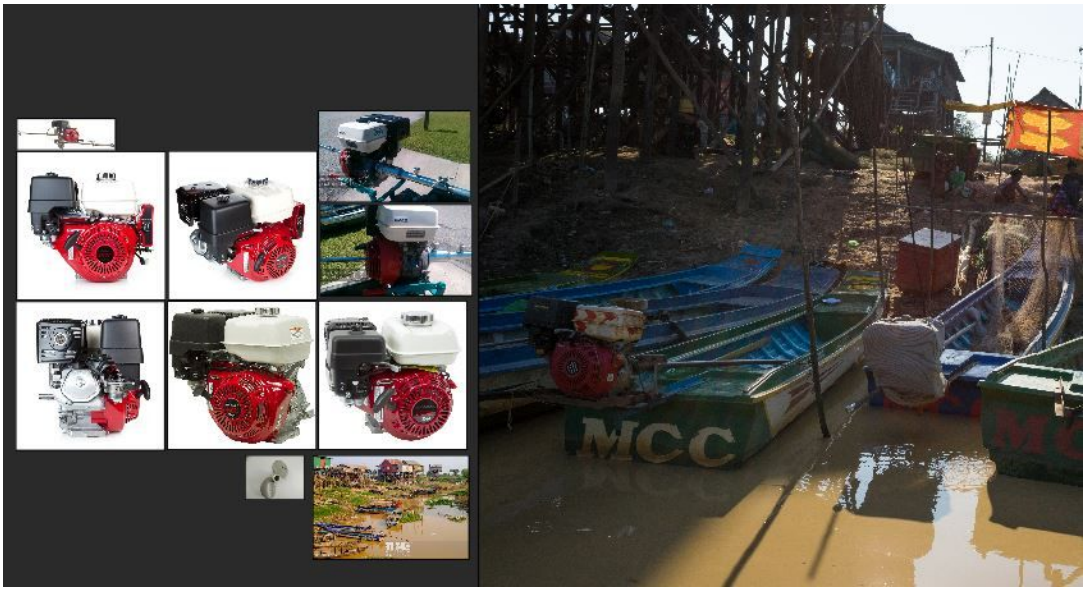
Referencing

Almost 1200 images from the area was collected and sorted into categories.

I then started a mood-board process through a piece of software called **PureRef**



That allowed me to get an overview of all my references and polish my initial idea into a detailed asset list.



Pipeline structure

From reference images I gathered an essential amount of assets and sorted them in **Trello** and parted them in three categories: **Litter**, **Assets** and **Heroes**.

Where **Litter** would be small objects that would make up a whole of impression. **Litter** in turn, was small pieces like trash; indefinable objects and forms.

Assets in this case, would be objects of no particular interest, but still objects that is seen and can be examined up close. They are assets of a detail and polycount that allows for a decent closeup, but doesn't really ask for any attention.

Last but not least, there were a section for **Heroes**. This would become the main actors in the finished scene. The objects that ask for attention and the player might be naturally drawn to. These objects are made in great detail and with great care.

In **Trello** I did a simple pipeline setup where I could move my assets horizontally through the phases I needed them to pass. The initial asset piping looked like this:

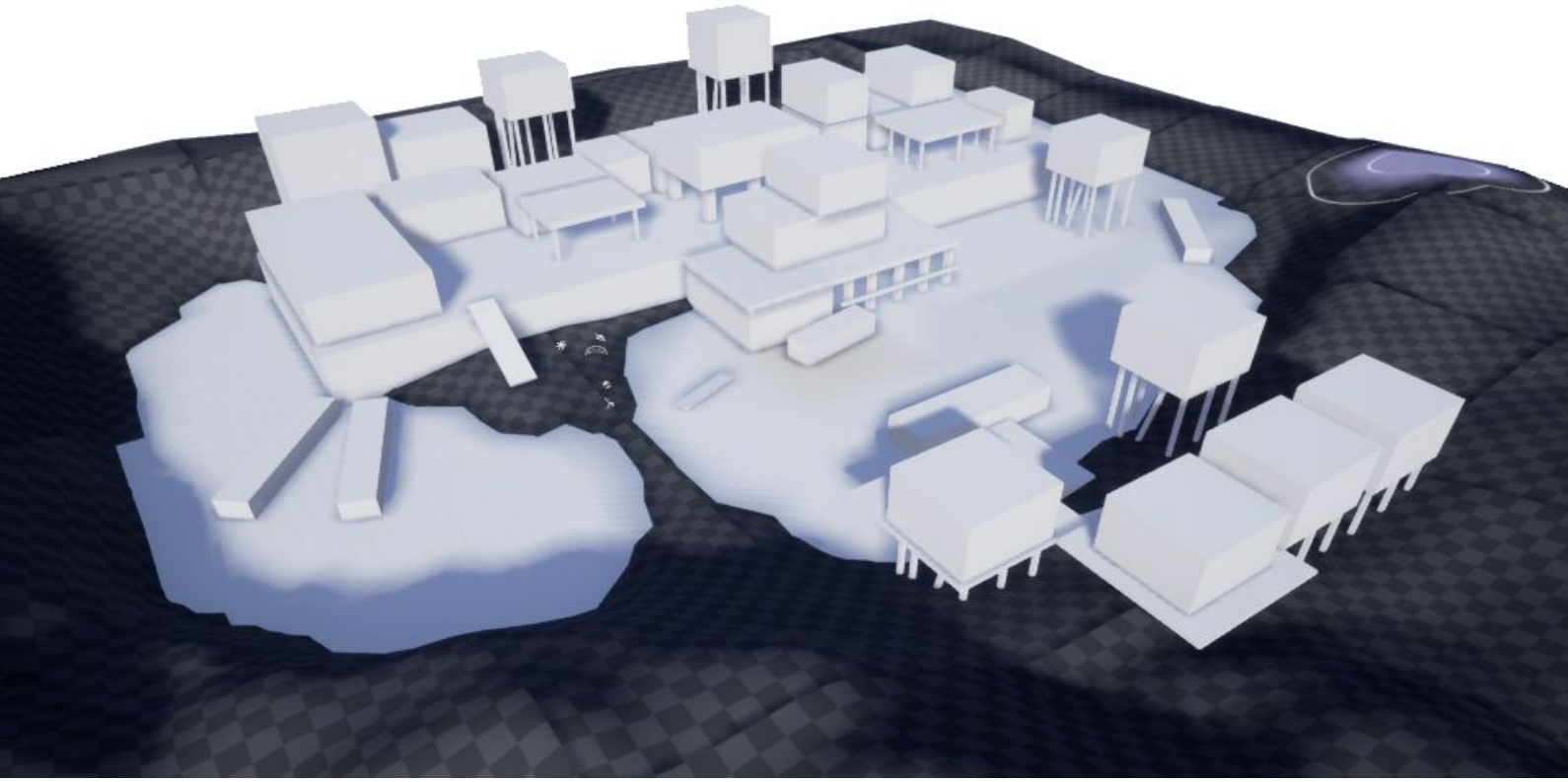
Asset list > Modeling > UV's > Texturing > Ready! > Implemented

I had additional branches from this. These were extension branches that allowed for assets to be quarantined for patching, before they eventually would be reinserted into the main pipeline. That ment i had an organized way to pull back implemented work to fix issues.

Blocking

Outblocking of the terrain and level started out with a simple mockup. I was drawing inspiration from the reference photos and the initial concept art from Robin Lhebrard. Later I moved in to Unreal Engine, where I did a simple outblocking

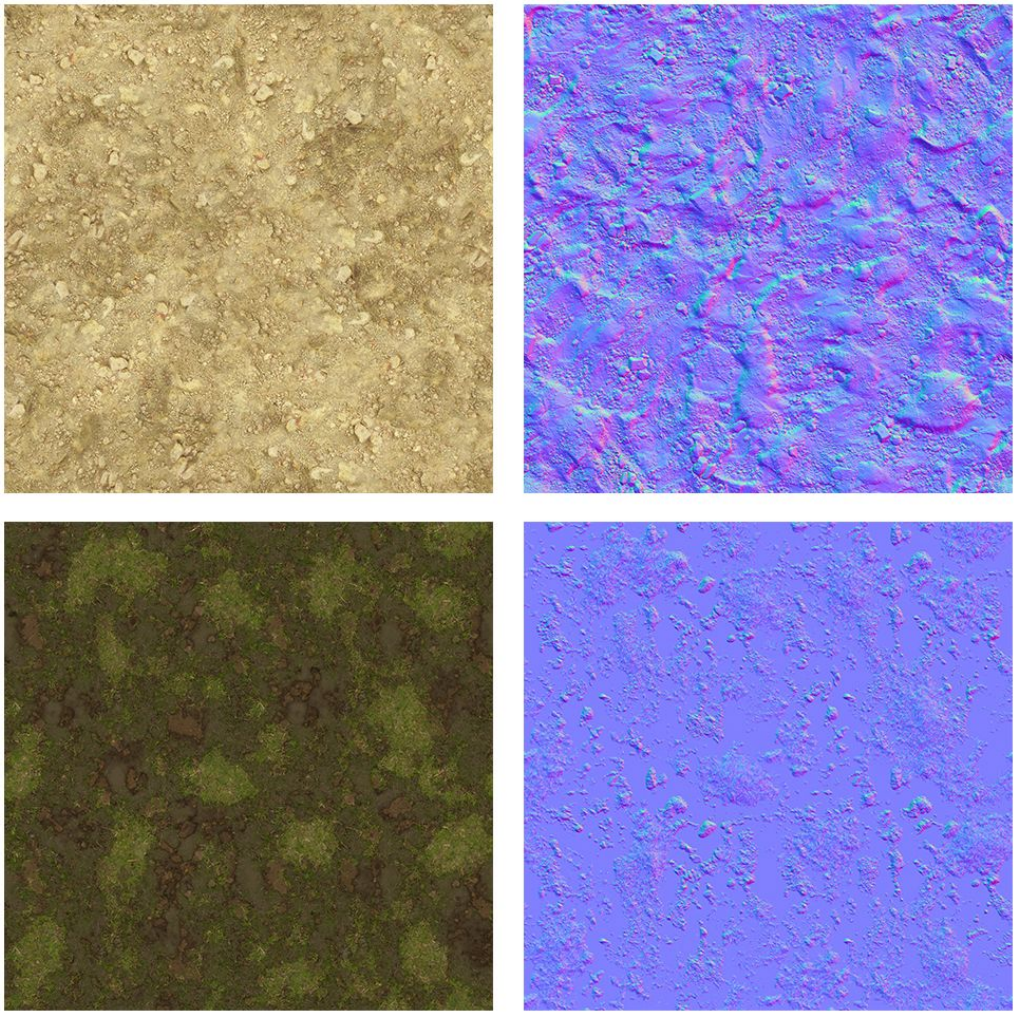

Boats
Skiffs



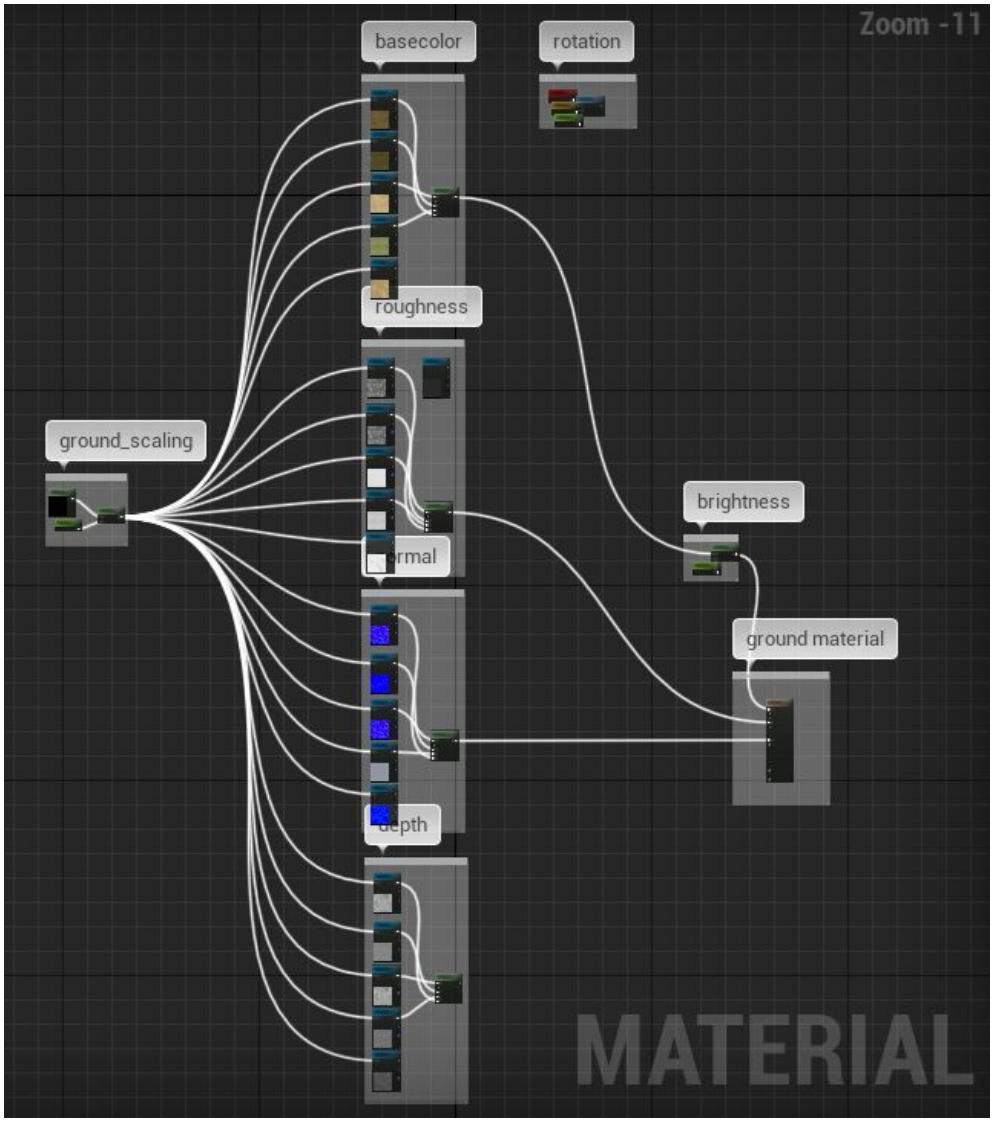
Terrain

One of my main concerns in the beginning of my project was terraforming the terrain. I was worried about the performance, tessellation and how I could get materials to blend well on surfaces. Using a heightmap and the in-engine landscaping tools I was able to sculpt a rough terrain before subdividing further. I worked with hydro-erosion and kept retopologizing the landscape to ensure avoidance of materials stretching more than necessary.

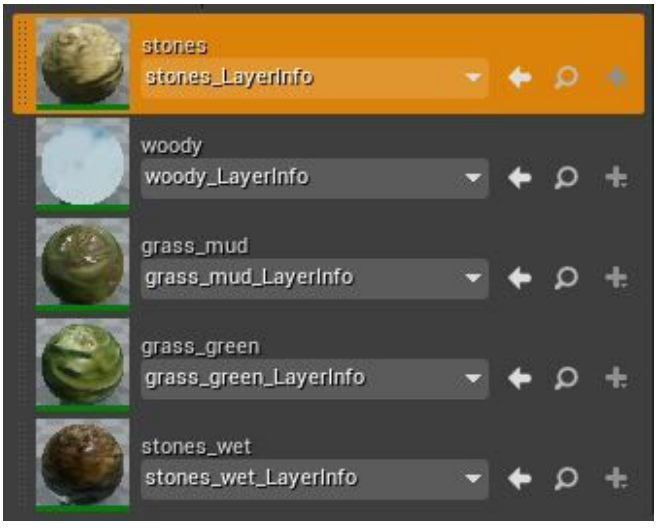
When I had a base of terrain I started working on materials. With **photoshop** I generated seamless textures and made use of different normal map generators.



After weeks of testing, I got blending between materials to look natural. I build a brush setup for the landscape painter in Unreal Engine. This consists of the PBR materials, including a tessellated depthmap.

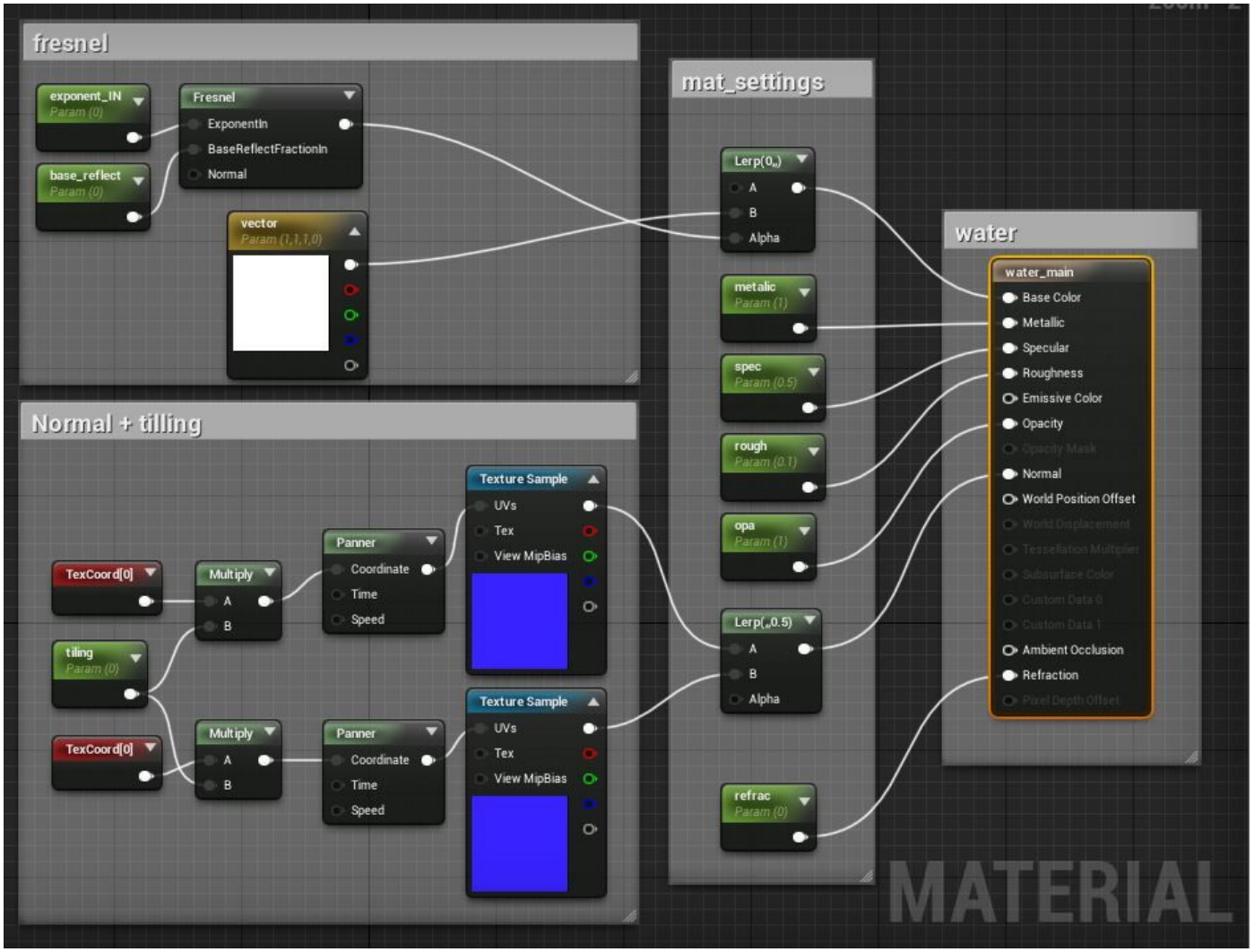


From this blueprint I could control the scaling, rotation and brightness. From there I could reuse ground materials only tweaking single inputs. This made it possible to change a roughness and brightness value and thereby generating a wet look on the same stone-brush i already had working.



Water

Figuring out how to create water in the scene, pulled me through different options, but I ended up working on a rather simple solution to minimize overdraw and optimization issues. The water consist of a simple plane spanning across the entire level. The material for the plane has a blueprint setup like this:



I have a fresnel describing reflection and transmission plugged into the base. Two normal maps is then plugged in through a lerp note with panners on each. This allows two normal maps to pan between each other to create the lapping of waves. Through this blueprint I then instanced it and finally had a rather complex amount of settings to work with. I could now easily control color, reflection and panning speeds in a simple menu. I made good use of this instance because I ended up changing the look of the water on a regular basis. Under the water and it's transmission I needed a terrain brush that could act as seabed. This

was solved using a highly reflective mud and grass PBR. This PBR made small highlights and reflections through the transmission of the water.

Running in real-time

In order to reach my goal of at least 30 frames per second, I knew I needed some kind of ruleset limiting polycounts, UV sizes, tessellation and for controlling LOD's.

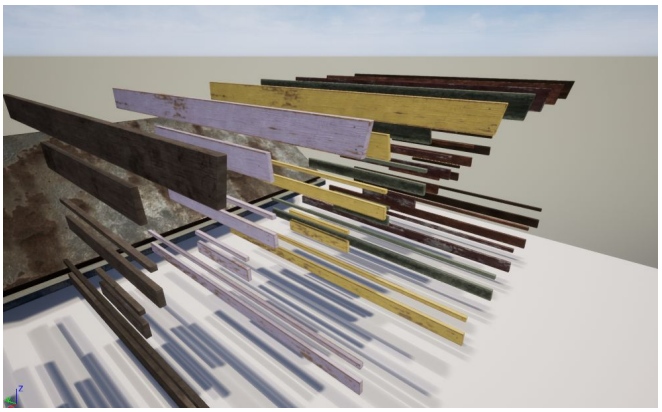
I set these parameters for optimization:

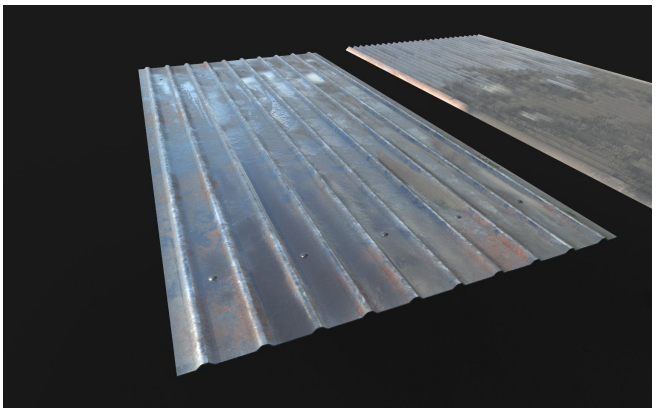
UV's got a maximum size of 2k and I calculated that a rule of thumb would be **1 x 2k per m2.** This rule had wiggle room, but I kept it in mind to even out the density. Furthermore I scaled down to 1k for single small objects. Some of the bigger hero props got multiple 2k maps.

I made categories for bulks of assets. That meant if I needed a bunch of small objects I would make a pack while still considering my texture density guidelines.

Procedural methods

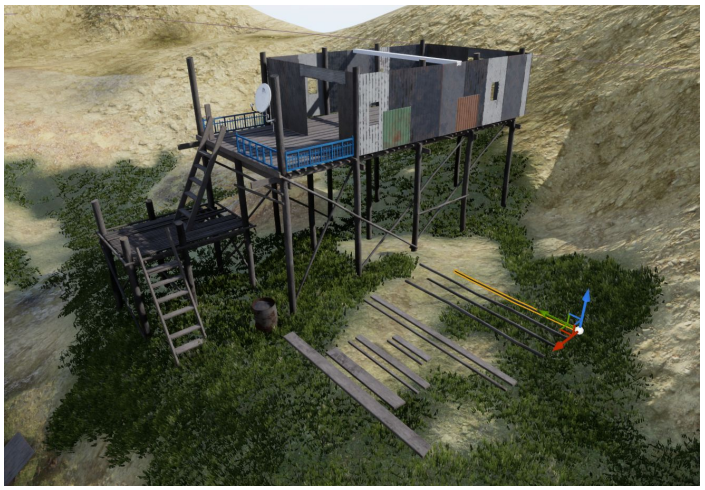
Considering my scope of size and content I needed a way to create sets of procedural packs for the construction of buildings. I made a separate Unreal level to play around with the possibilities of making these packs and blocked out objects as stand-ins for construction materials. This became the basis for a simple solution; making packs of wood-planks that would act as LEGO-pieces, making it possible to create endless procedural pieces and sections directly in engine.





Packs of building blocks was modeled and the UV's for these packs was then reused for texturing in substance painter. I used the Unreal blueprint editor to setup prints that controlled the different UV's to get maximum control of variety. For most of the objects I only needed to control the color UV maps, the rest I reused in different combinations. That meant that I could use grunge in different forms on top of color and I could change that color through blueprints and via single UV's instead of changing a whole set of PBR materials. In effect, one boat became more boats.

With that in mind I started assembling **procedural building packs**. I was building the first houses for testing. Later I combined bigger **clusters** of objects for reuse in other projects. Few alterations made new houses look completely different. The more clusters of objects I created, the more i could build. Exponentially this resulted in great variety. This method proved to be extremely powerful and easy on performance.



Workflows

Assets and texturing

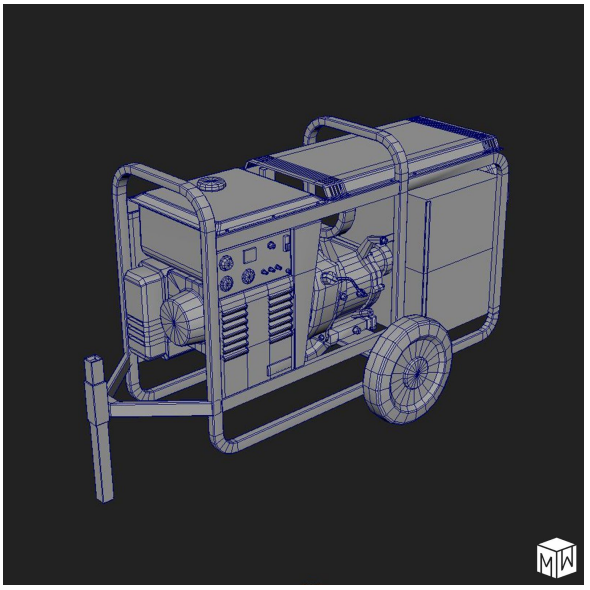
Starting in Maya, I was sub-d modeling **Hero** assets with a target tricount of 10k per model. The before mentioned **Litter** and **Assets** in general was gathered in packs and modelled together. Again, aiming for around 10k tricount per 1 m2.

Many of the models had the possibility for two-sided geometry. Flat plates got either alphamaps or two-sided geometry to avoid unnecessary faces. This image shows the use of two-sided geometry:

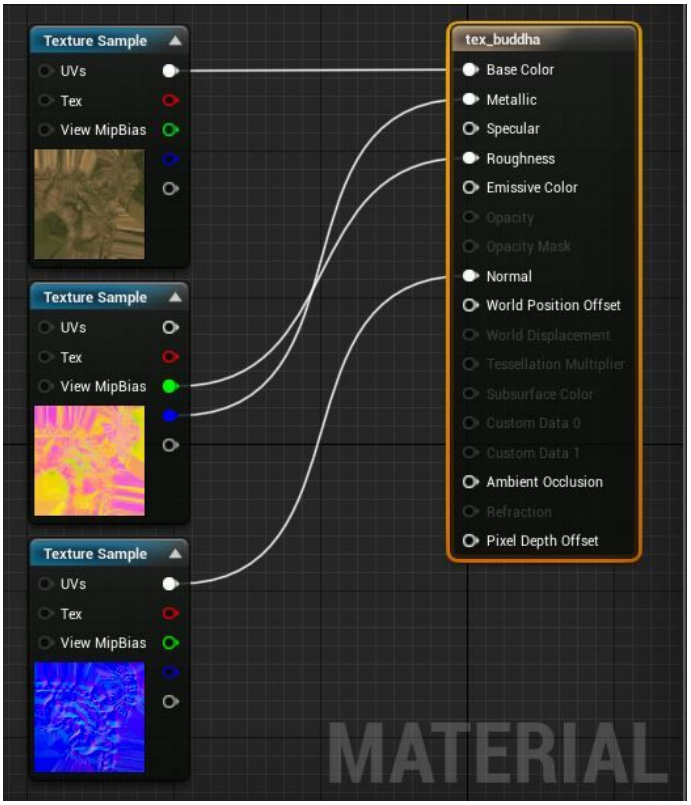
I referenced level of detail from AAA games currently, in order to imitate how low I could go and still avoid visual segmentation.

Before moving out of modeling I unwrapped the models and made sure the UV's was optimized and neat. **Making sure UV's are tight is doing myself a great favor**. Being sloppy with UV's will cause me problems in the later workflow, or if not - cause me even bigger issues when implementing into engine. Overlaps in UV's and lightmaps can easily become something that makes life unnecessarily hard.

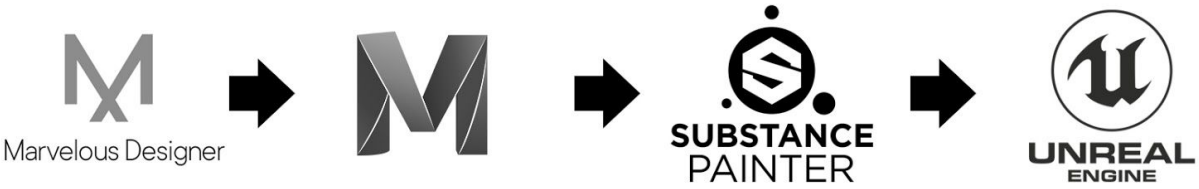
Making my way into **Substance Painter** i learned a lot about procedural work as well. Creating solid materials for reuse was key in my texturing workflow. When i had setup versatile material solutions, the materials were saved as smart materials so I knew I could apply them to other object later in the process. Working with the custom smart materials made the scene come together a lot easier as well. This created cohesion in the scene that otherwise would have been hard to detect before the final stage of development.



Throughout the first part of the process of texturing I made use of normal PBR output from Substance Painter. Later in the project I realized the potential of combining maps through RGB channels. This meant that instead of having roughness and metallic channels separate, I could combine them and output them through green and blue. I left the red channel (for occlusion) unused and used a post processing volume to solve that instead. Below you see a combined UV and it's respective blueprint with a double output into rough and metallic. This model is a retopping and texturing of a scanned buddha statue. Using this workflow I gained a decent amount of performance that I could clearly notice in the advent of Unreal Engines build-in profiling.

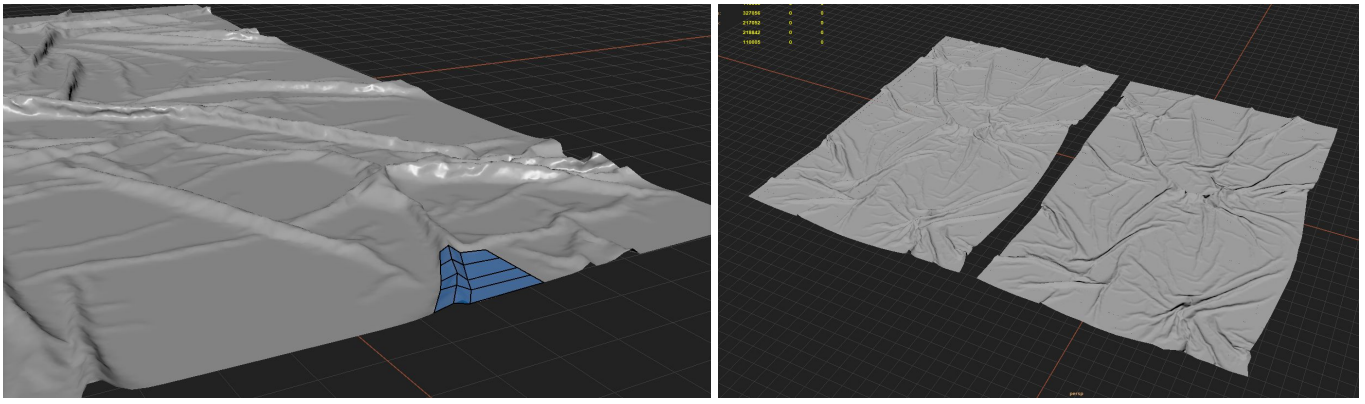


This is Marvelous!

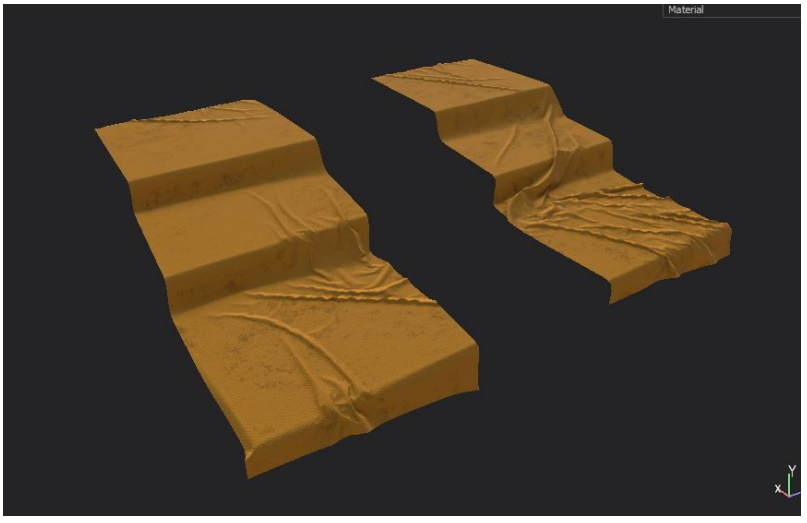


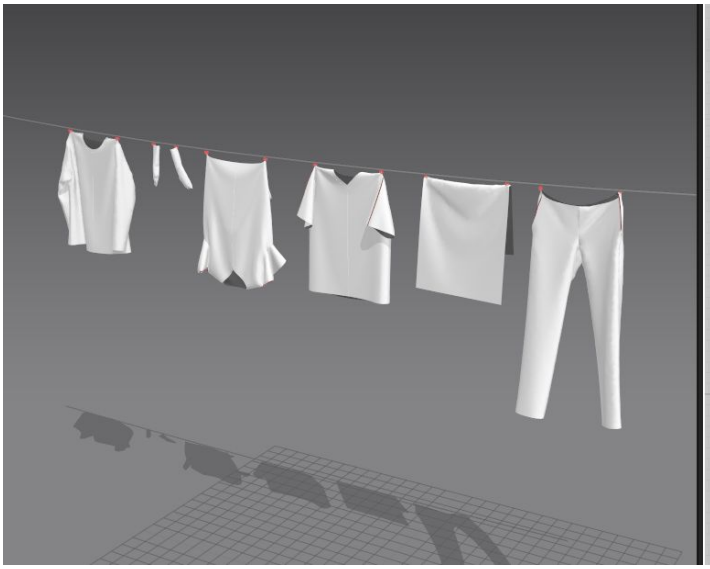
Simulating cloth in Marvelous is, quite frankly, **marvelous**. I knew from the beginning that I needed a lot of cloth in different forms, shapes and materials. Making sure that the scene kept an organic feel alongside all the hard surfaces was key. From reference I saw all the clothing and carpets dangling from wires and balconies.

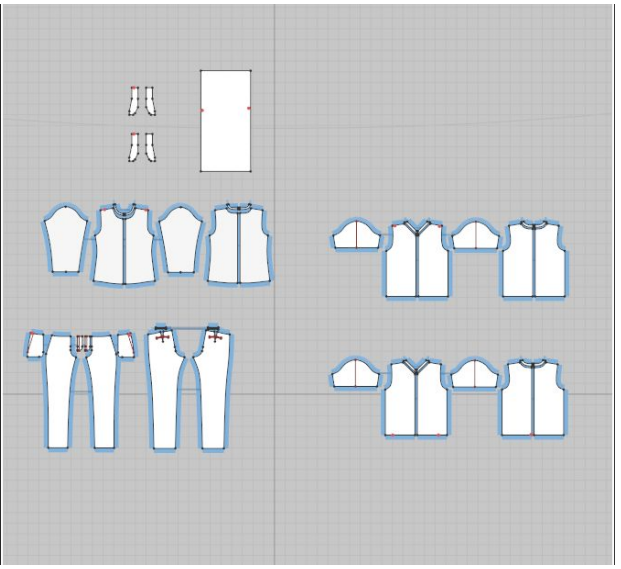
Although I knew I had to get better at retopping cloth, I also knew it was the only way to get **200.000 polygons** for one object into Unreal Engine. Daunting task nonetheless, I quad-drawed my way through the first simple simulations. By **baking** the high-poly down onto the low-poly I got normal maps.



Later in my Marvelous adventures I found that the whole workflow could be sped up tremendously. By driving up particle distance in Marvelous, exporting, and then after export lowering it to a point where the polycount was more reasonable, gave me a low and a high-poly instantly. I started baking normals in Substance painter before texturing as well, avoiding baking normals inside Maya. This new and optimized workflow made it possible to speed up the process to the degree where I had the freedom to make even more cloth. And who doesn't think cloth makes a game look even flashier. Not me!


Material

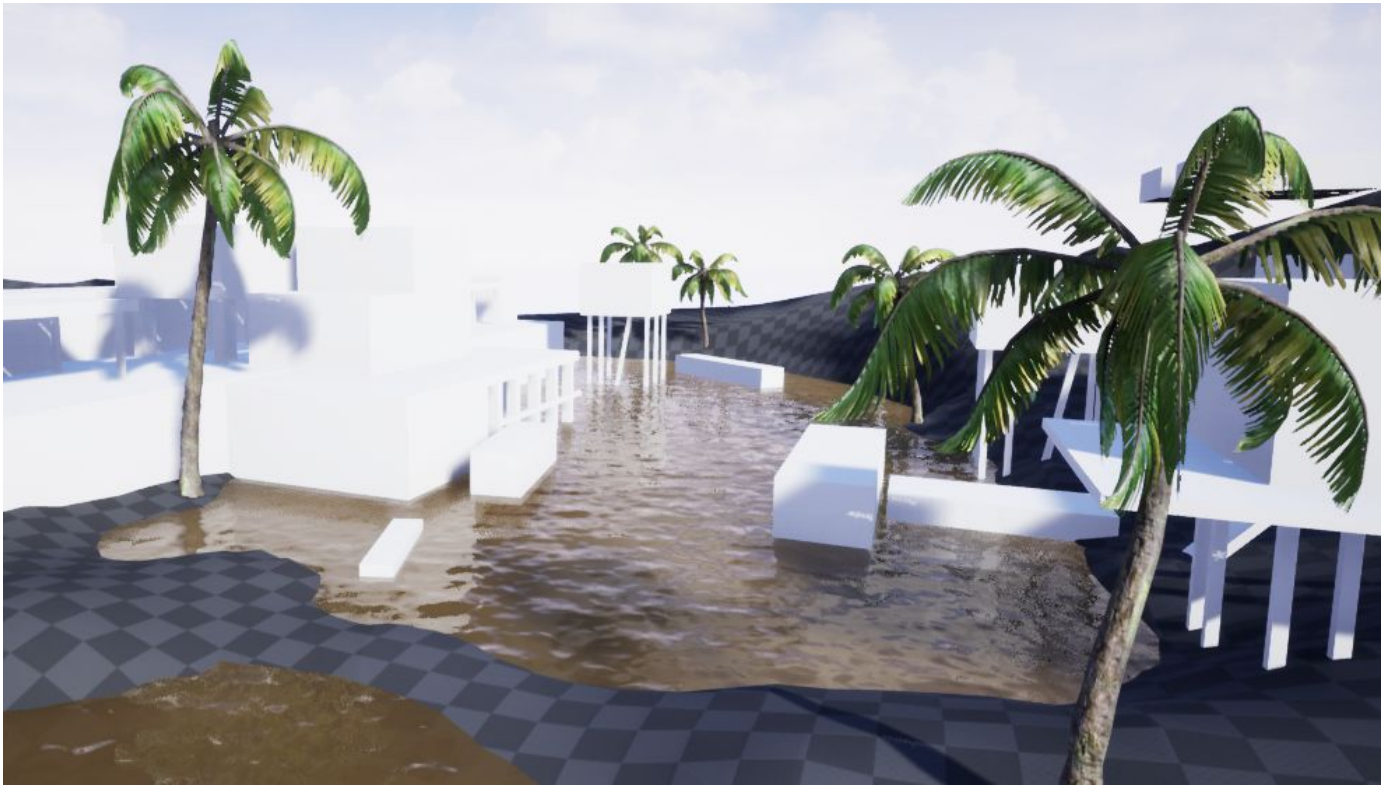




Trees

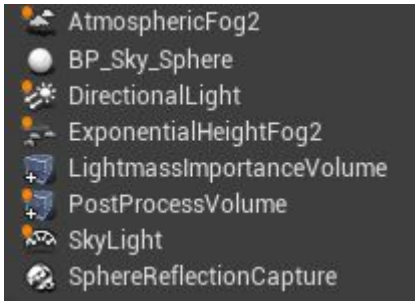
Trees and foliage is essential when trying to create an outdoor environment. **Speedtree** does it best. It allows for endless possibilities when creating natural elements such as plants and trees.

In Speedtree I created my palms. I created five different iterations of palms. Some were lying, some were standing straight, some without leaves. But due to palms in general looking quite generic, I ended up using one single palm. I slanted it, I made it bigger or smaller, and rotated it. I used both Speedtree Cinema and Speedtree UE4 for my Speedtree workflow. Speedtree also allows for **wind** to manipulate trees and foliage, and with a bit of tweaking, that translates directly into Unreal's wind system.



Lighting

I knew from the get-go that lighting would be extremely important. I also knew that it would be something that would take time and a lot of reading to achieve. Doing my first lighting setup I was content, but later I realized it wasn't good enough and I redid it. I ended up redoing lighting from scratch at least a dozen times.



One of the issues with the lighting in general was that I wanted to have only one lightsource. I tried solutions with stationary lights, lighting up small parts of the level, as well as sections of the maps lighted by post process volumes. I ended up running with one source and tweaking a big postprocess.

Dealing with indirect lighting was also something I cared deeply about. In some areas of the map, the shadows seemed to hard and dark, in others too light. This needed a lot of tweaking back and forth.

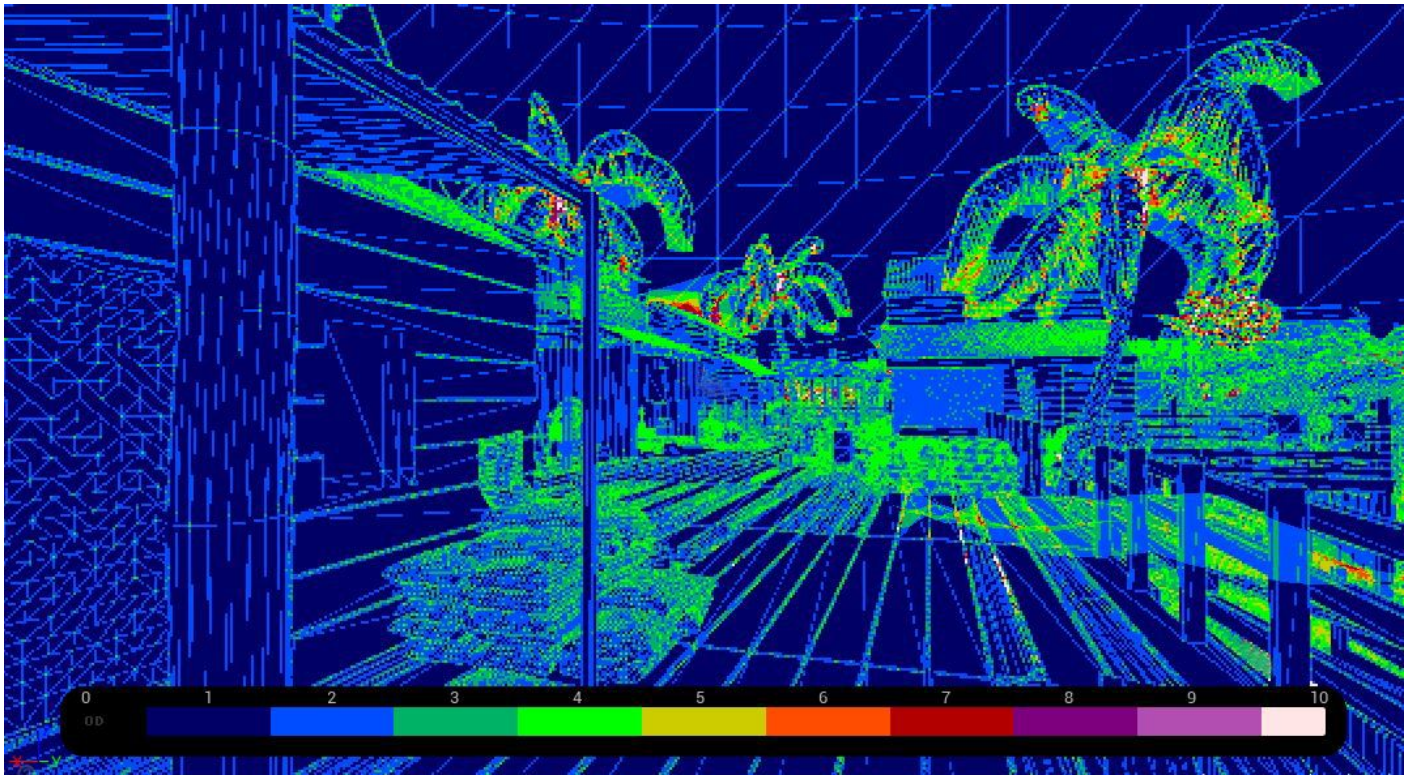
Dealing with backlit faces of buildings often made the faces stand out completely black. I needed the build-in auto exposure for that. This means that moving from heavily lighted areas and into darkness the "eyes" would adjust. After working with auto exposure I learned a lot about how light works with the human eye. I used an excessive amounts of time talking with my mentor about this issue whilst looking out my window, seeing my own lit living room compared to the living rooms across the street, that had windows dark as night.

I grew a huge interest in understanding lighting over the course of this project and a great respect for people who master it. Not illuminating objects correctly can easily make them look flat and uninteresting.

I worked a lot with HDRI maps in my lighting process. Considering the use of only one lightsource, these made a world of difference to the density of shadows, general tone and contrast to the level. I used HDRI maps from **HDRIhaven.com**. I worked with several different maps in combination with my postprocess to get it right.

Performance and Profiling

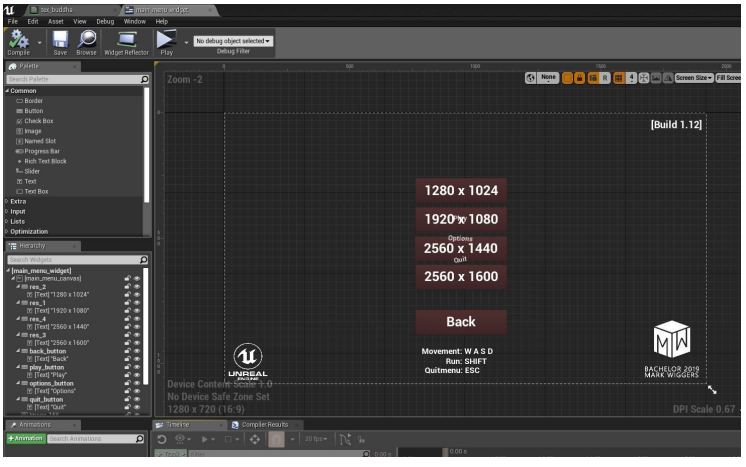
As aforementioned, performance was one of my key points in this project. Alongside all the procedural work and optimized texturing, I needed to make sure everything was in order. Therefore I used the build-in profiling in Unreal Engine.



If I encountered issues with the profiling this meant a hit to my framerate. So in this **QuadOverDraw overlay** I could look for white and determine whether to work on fixing the problem or if I wanted to accept that overdraw. In some instances I solved the problem, in others I could see that adjusting the overdraw would not cause a big impact, and therefore didn't see the need to put in the hours to fix it. Using precious hours on a problem causing framerate drops of 1 or 2 frames, was in some cases not worth the while. In other cases it freed up huge amounts of **GPU performance.**

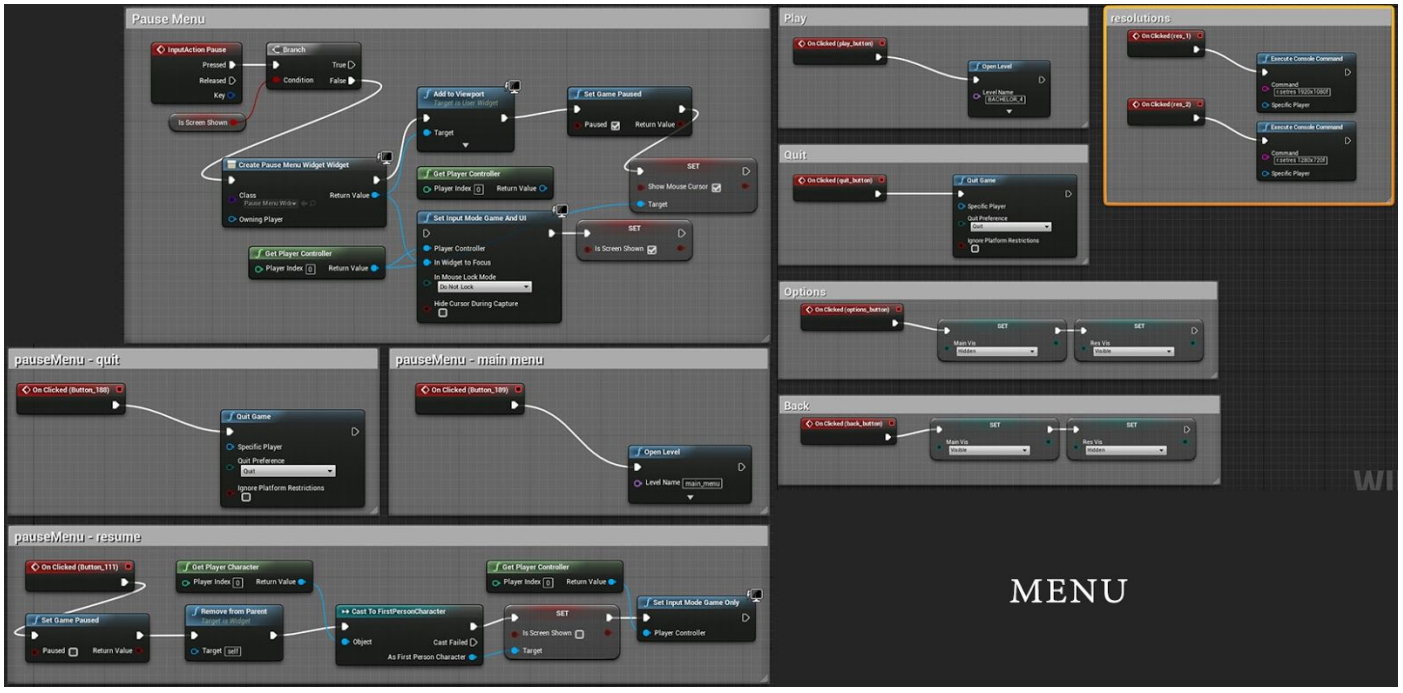
Menu

To be honest. This process of the project was in my opinion not needed at all. **I just couldn't help myself.**



Setting up a menu system was something I did between other more important work. In the beginning I needed a way to quit out of the game instead of ctrl-alt-delete and killing the process. So while i was at it, I might as well build a system.

From blueprints I created a system where I could quit the game.

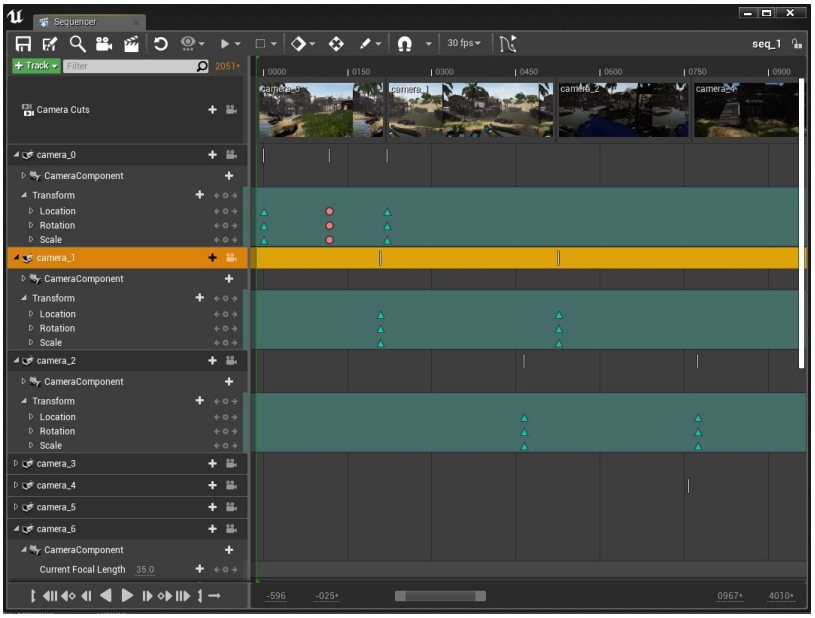


I did testing of my game on many different PC's and noticed that resolution scaling was a problem in general. I therefore build a menu system to adjust resolution of my game and anchored the menu to those resolutions. (More resolutions can easily be added but these were the ones I ran in to).

Finally I wanted the main menu to look cool so I built a seperate level for the menu system. To avoid any further performance issues I contained that space to only the viewed portion of the map. The final level looks like this:

The final menu looks like this:

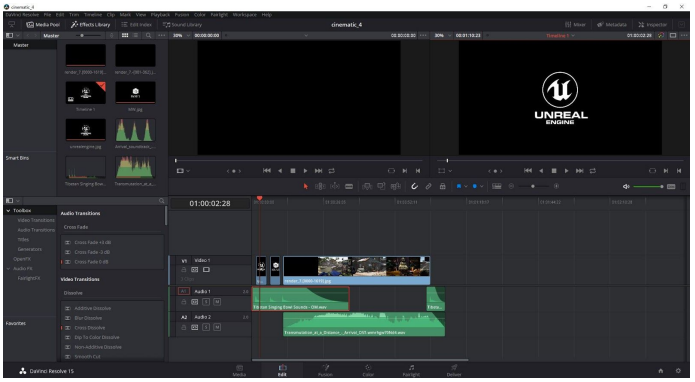
Cinematic



I wanted the cinematic to be something that captured the essence of the environment without gameplay.

I used Unreal Engine's cameras and **sequencer** to create slow and emotional movements that allowed the viewer to see the environment for what it is, and not using any flashy zooms or drone shots.

I think the sequencer in Unreal has become very intuitive and has a very large range of compositing interfaces that makes it possible to use effects and animation curves. As the game also runs smooth real-time, the render times for my 2 minute cinematic was around 30 seconds.



When I had my exported output as an image sequence, I moved on to **Davinci Resolve.** There I edited my final cinematic and added music and sound effects. I ended with a light color grading.

I used my cinematic for **Vimeo**, **Youtube** and **Artstation**.

Packaging and Testing

I arranged a system for my versions. Everytime I reached a new milestone in development I did a light packaging of my project with static lights and low quality lighting build. That way the building wouldn't take too long. Then I wrote a changelog to my new version and named it i.e "cambodia_alphabuild_1" - Then I zipped it up and transferred it to an FTP.

I got some of my friends with different PC setups to download the .exe and run it.

They gave me feedback, both on performance and issues that arose from glitches, interface issues and also artistic critique.

I noted the problems and when I build a patch build i.e "cambodia_alphabuild_2" I put the correlating fixes in the new patch notes.

As I got closer to deadline, I made more versions and kept fixing. This system was extremely effective for my work. Just knowing that the build worked on different specced PC's were a peace of mind.

In the final package I got a peak performance at 162 frames per second and a low on 55.

That shows an **average in-game frame rate of 108**

What now?

This project has taught me so much more than I could have ever expected. There are so many facets that this project has pushed me further in to.

I started out with almost no experience with Unreal Engine and I forced myself through almost every single part of a game development workflow in Unreal Engine. If I knew what I know now the time spent on this project would probably be half if not less.

So what now? Concerning the environment, I strongly believe that with all the building blocks made, and the optimization that has gone into it, the level could be expanded to ten times the size. LOD's should be optimized some more, but in general the buildings and surroundings could be varied infinitely.

I have had the pleasure of trying out and working with the **HTC Vive** for some time now. And I have started working on getting my environment to work with it. So far I can have a small portion of the environment running real-time, but the fact that Vive has to run 90+ frame rate on two seperate monitors makes it a bit on the heavy side to expect any high-end gaming PC to run the full environment at this time.

I have loved every second of working with this game and it has only strengthened my hopes and dreams of getting to work with games in the future.