Creating an environment in Unreal Engine 5

Jesper Larsson (jesla966), Julian Voith (julvo429) and Eric Edelo (eried124)

Figure 1: *Real-time rendered screenshot of the final environment.*

1 Introduction

As 3D computer games are getting more realistic and sophisticated, the demands on the hardware of computer have increased. These hardware limitations always have been a problem in the area of real-time computer graphics. Brute force methods are not viable solutions for real-time rendering, it requires smart techniques to overcome that.

The project's aim was to create a small game level inspired by the game Mirror's Edge to get a better understanding of creating game assets and deeper knowledge of the process and limitations of creating a video game environment.

1.1 Idea

The initial idea for the project was to create a real-time environment in Unreal Engine 5 (*UE*) inspired by a game called Mirror's Edge. After assessing the groups skills and previous experiences and evaluating videos and images of the games various levels, the group decided on creating an indoor scene heavily inspired by the Mirror's Edge level, shown in **Figure 2,** which could be navigated through using a simple character.

2 Method

2.1 Preproduction

At the initial stage of the production, a low fidelity scene was produced using simple primitives inside of the modeling tools, to serve as a rough block-out of volumes and the layout of the environment. This rough block-out can be seen in **Figure 3.**

The group evaluated which objects or assets that would be required to populate the environment and theses were recorded and represented by temporary low fidelity placeholders in the block-out scene. The result was a spreadsheet of required assets, where the references, production progress and work assignment were recorded.

An internal documentation wiki, shown In **Figure 4**, was also created where software configurations, project folder structures, naming conventions, workflow guidelines and links to resources were documented.

2.2 Modeling

The modeling was done using both Autodesk's Maya and Blender. These tools were chosen based on the group members prior experience with the software.

All key assets identified in the preproduction stage were equally distributed amongst the group members for modelling. After completion of the high-fidelity modeling, they were passed on and reviewed by the other members. From these high polygon models new models with lower polygon counts, better suitable for real-time rendering, were derived.

Lastly additional models with even simpler geometry were created, only containing the basic silhouette of the original model. These models are used by UE's Level-of-Detail system (*LOD*) swapping out models based on a model's screen area coverage to optimize the scene performance at the expense of very minor visual differences.

Complex models had collision meshes constructed out of simple convex primitives whenever UE's generated collisions were insufficient. An example of this process can be seen in **Figure 5**.

Finally, the low-poly models had UV-map sets created. One primary UV-map for the material textures and a second one for a baked lightmap. The basic UV work was done in Maya and more complex layouts and UV packing was done in the UV tool RizomUV.

2.3 Texturing and materials

UE uses a node based Physically Based Rendering (*PBR*) system for shader materials and textures, using albedo, metallic and roughness channels. To keep the shader interface unified and the shader count at a reasonable level, a few master materials were created: one for PBR materials with basic constant values, one for PBR bitmap textures and one with opacity masking support for decal projections. Materials for the assets where then created by instancing these master shaders, overriding only the necessary parameters and reusing the core underlying shaders.

Many of the assets only required simple materials with some surface roughness imperfections and using constant values or tiling textures. Most of the project's bitmap textures were created using Substance 3D Painter with procedural textures made using Substance 3D Designer. These tools allow textures to be created procedurally by using mesh geometry information combined with layered noise and image processing functions. These tools are exposed through a node-based interface as shown in **Figure 6**.

Some models and deferred decals were given additional details by using tangent space normal maps. Ambient occlusion and cavity maps were also used to give some materials non-direct diffuse shadowing and specular occlusion.

In addition, a plaster texture was sourced from AmbientCG and a handful other textures and a plant asset were sourced from the Quixel Megascans asset library, accessed using the built in Quixel Bridge plugin. Megascans is a library with both high-quality 3D scans and PBR textures available free of charge when using UE. These assets can be seen in **Figure 7**.

2.4 Populating the scene

Finalized assets were imported into the engine, configured with materials, level-of-detail placeholders, collision boxes, assigned lightmap slots, and placed in a calibration test environment for evaluation and scale comparison. After minor scale and mesh corrections were done, the final assets were used to replace all the low fidelity primitives used as placeholders in the block-out scene.

Group members were assigned regions of the main level, in which sub-levels were created. Sub-levels are their own isolated files and are instanced into the main level, which allows collaborative work on the main level without the issue of merge conflicts or file overwrites. Each group members were then free to populate their region with the previously created assets.

2.5 Implementing the character

The character for the scene uses the pre-made default UE model with a custom texture (displayed in **Figure 7**) and motion captured animation sequences sourced from the Epic Games Marketplace.

For the controls and camera perspective of the character, the blueprint visual programming system in UE 5 was used. The camera was configured in the blueprint as well as the actions which are being executed once defined keys are being pressed. Those actions are called events and are set on the Event Graph. The events are then triggered by pressing assigned keys. An example of the graphs used is displayed in **Figure 8**.

Blend Spaces were created to use different animation sequences at the same time if required. Blend Spaces allow the placement of animation sequences along a graph based on set samples. The overall animation is then calculated by blending between the set samples.

An Animation Blueprint was then created which is assigned to the skeleton mesh of the used character. Within the blueprint different Events were set up on the event graph which were used to execute different animations sequences and Blend Spaces.

2.6 Lighting and post-processing

After the level had been assembled, a final rundown of all the placed assets were made to ensure an even texture and lightmap density distribution was applied.

Guide and importance volumes were then placed to assist the engine in prioritizing where to send light samples during the baking process.

Reflection probe captures were distributed around the scene which captures the scene from their locations at when the level is built and allows surrounding geometry to use nearby captures as a fallback solution for environment reflections. A planar reflection volume was placed and aligned with the floor, which then captures the scene in real-time and allows for dynamic and accurate reflections.

Finally, the lightmaps were baked using the in-engine photon-mapper. Some tweaks were made to the auto-expose to adjust the dark areas and bloom levels. Some slight color grading done to align the colors with the original reference.

3 Results

The completed project can be seen in **Figure 1** and **Figure 9**.

4 Discussion

One of the problems that came up during the project were the scheduling of meetings due to varying schedules. This sometimes led to meetings in the late evenings and during the weekends.

Another problem early on with the large textures produced by the Megascans plugin by default, resulting in texture memory budget issues. After profiling and identifying the issue, the texture count was reduced, and remaining textures were down-sampled.

Additionally, problems were caused by many of the new cutting-edge features available in UE. The latest iteration of engine target high-end computers, however most of the members lacked the hardware required to run these features. Hence, high end features such as raytracing, virtual shadows and virtualized geometry had to be omitted in favor of traditional light mapping and LODs. A large chunk of the project was spent optimizing the scene performance in order to run the scene smoothly on the members various hardware configurations.

5 Summary

The group set out to create a 3D environment in UE inspired by the game Mirror's edge. The group learned a lot in the process and are incredibly happy about the results.

***Figure 2**: The original Mirror's Edge level used as inspiration for the project. Screenshot was captured in-game.*

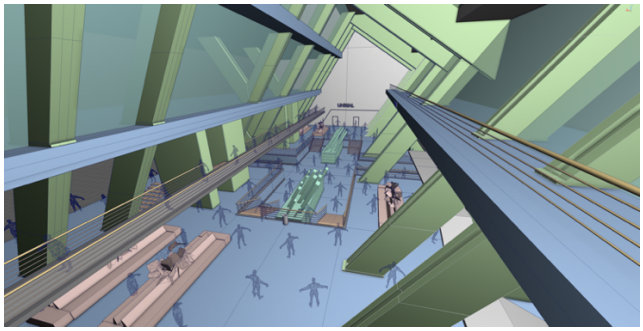

***Figure 3:** Initial environment block-out using simple primitives and human silhouettes were a scale references.*

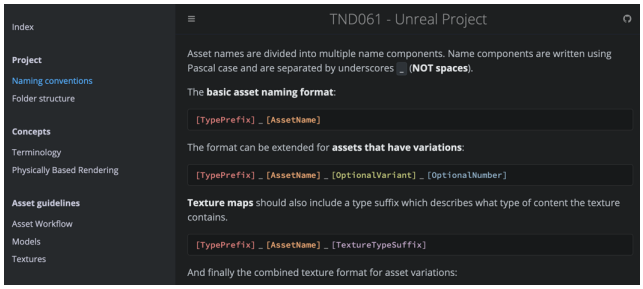

***Figure 4:** Internal documentation page, showcasing the naming conventions for project assets.*

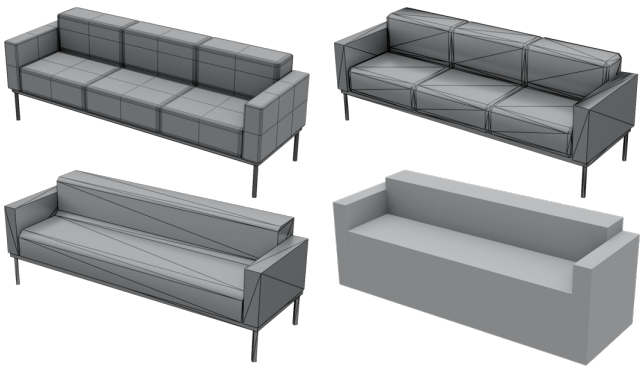

***Figure 5:** Example of one of the assets modeled for the project. Top left: High polygon model. Top Right: Low polygon model. Bottom left: LOD-model. Bottom right: Compound of box collision meshes.*

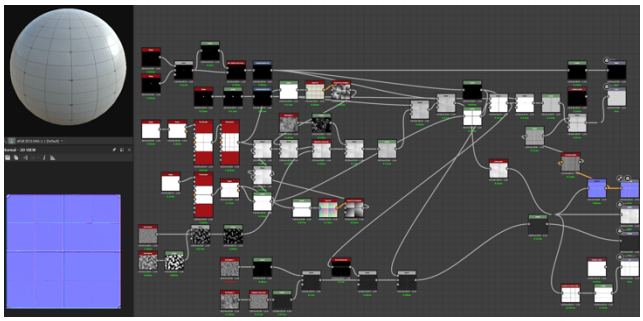

***Figure 6:** An example of one of the projects procedural textures created in Adobe 3D Substance Designer.*

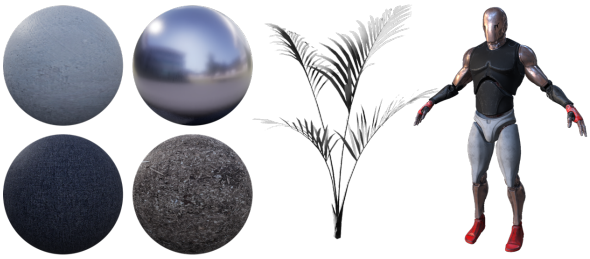

***Figure 7:** Showcase of all the third-party assets used in the project. The plaster texture used in the top right material was sourced from AmbientCG. The remain materials and plant model was sourced from Megascans. The character was part of the Unreal Starter Content with custom texture applied.*

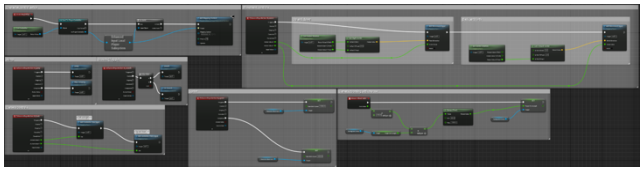

***Figure 8:** Example of an Event Graph (character controls)*

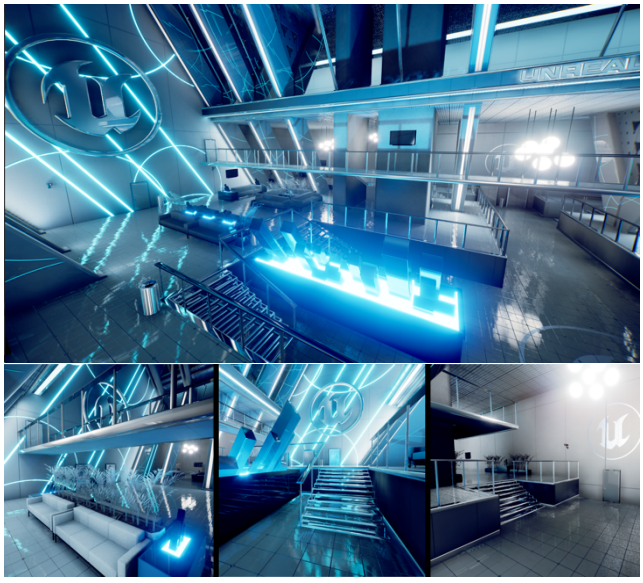

***Figure 9:** Realtime screenshots of the final environment inside of Unreal Engine 5.*