5-2014

The Tower: Constructing a 3D Scene

John W. Wesson

The Tower: Constructing a 3D Scene

Thesis submitted in partial fulfillment of Honors

By

John Wesson

The Honors College

University Honors Scholars Program

East Tennessee State University

April 18, 2014

Table of Contents

Introduction

This thesis accomplished several major objectives. The initial effort was to produce a work of environment art from beginning to end. This would result in a completed piece that would strengthen a digital media portfolio for an environment artist. As progress was made, it was realized that there was great potential for further research into a wider range of new workflows, techniques, and software for a constantly evolving industry landscape. The scope of the initial idea allowed for development in a wide range of skillsets. The ultimate result of this thesis project is an extensive body of knowledge that will drive future improvement as well as a completed work that demonstrates expertise with current development pipelines and cutting-edge technology.

The project went through several major phases of development that differed fairly heavily from each other, influencing the progress towards the final product. The shifting nature of development over the course of the thesis resulted in several outcomes. A series of different goals for the final result meant that a wide range of methods and processes were investigated. The breadth of skills required over the full course of this thesis project resulted in invaluable experience with a variety of methods and software. From a production standpoint, the changing goals and workflows caused major issues that hindered progress toward efficient completion of a final result. However, the challenges encountered over the course of constructing the scene provided valuable insight into future best practices for constructing a work of environment art, especially for a real-time game engine.

Design Goals and Concept

The initial target for the final output of the thesis project was an environment piece using techniques common to the film and VFX industry. As a result, the first phase of production was realizing the scene using matte painting with possible incorporation of 3D assets. The final product of this was intended to be a rendered output consistent with a wide angle establishing shot in a cinematic style.

Several iterations of sketches and paintings helped visualize various ideas as well as establish the thematic content of the environment. After multiple revisions, some common elements appeared challenging enough to warrant further development. These common components included mountains and integration of natural elevation with architecture. The other consistent theme that was expressed in most of the initial concepts was European architecture, specifically the fusion of stone, metal, and glass into vertical structures. These common elements were combined to complete the final matte painting (*Figure 1*.) that would set the tone for the remainder of this project.

Figure 1.

The focus for this particular idea was to accentuate the natural elevation of the landscape with architecture that molded itself to the cliff faces instead of forcing itself around them. Snow and ice accumulation on both the cliff faces and the structures helped to further assimilate the man-made buildings into the scene. The polar climate was chosen to create a specific color palette for the scene, along with the lighting and atmosphere. Using vertical lines in the architecture created interesting forms and silhouettes that contrasted with the more horizontally oriented landscape. The placement of the hills and bridges were intended to draw the eye to the horizon.

The image was created as a matte painting, which required photographic source material. This collection was used as both reference and inspiration throughout the course of the thesis project. Photographs were acquired as needed over the course of the entire project. The matte painting process was completed using Adobe Photoshop and incorporated a layered workflow that allowed separate elements to be introduced an adjusted separately, a technique influenced by and adapted from work by Jared Simeth. This is what enables a wide scope of photographic source material to be incorporated into the painting, and would also allow for any 3D elements to be composited into the scene.

Referencing a wide variety of source material allowed a visual style to evolve that was both believable with relation to existing architectural styles while remaining comfortably unique. The vast majority of the style in this project is a fusion of Old Gothic, Gothic Revival, and Baroque architectural elements. All three of these utilize stone as a primary building material with metal accents or ornamentation. The upward draw and sharpness of the Gothic style worked very well to complement the more rounded geometric forms found in the later Baroque mode, as well as the rounded spire of the clock tower, which is a common feature of southern Germanic towers.

High-Poly Modeling

As progress in this first phase was made, various decisions to include a higher proportion of 3D elements led to the target for the final output being shifted. The new revision was to a construct the major elements of the scene in full 3D with the background elements realized as matte painting. The final product was to be a cinematic-style video rendered in Mental Ray and composited with the matte painting elements. This change was made in an attempt to include higher levels of detail in the main elements of the scene and create a more engaging final product. The previous work on the matte painting would be incorporated both as concept and background elements.

The initial step in this new process was to start a rough blockout (*Figure 2*.) of the major architectural forms in Maya. This was intended as testing for what sections of the architecture could be modularized and iterated throughout the scene.

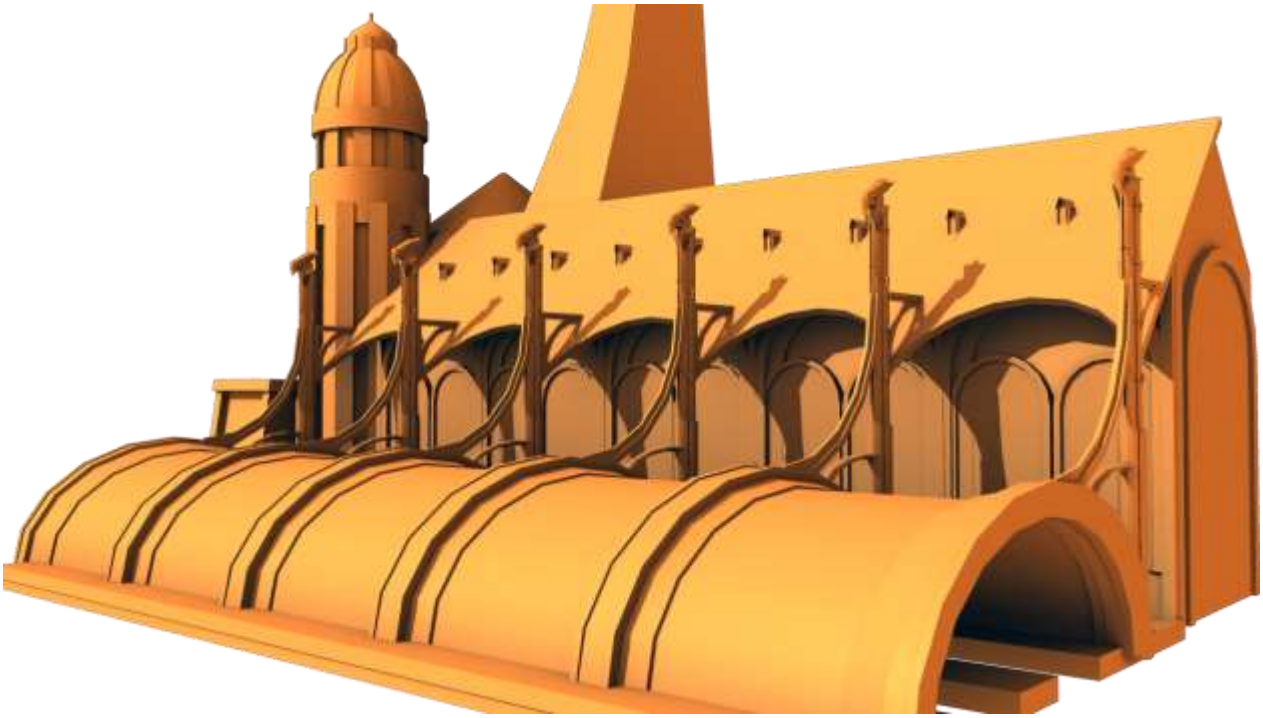

Figure 2.

The rough blockout was also used as a starting point for creating a series of more precise meshes that were constructed on a grid, to scale with real-world proportions. These grid-aligned meshes could then be used as templates when constructing the high-polygon assets to ensure that the meshes maintained their modularity.

The next step was creating high-poly segment pieces for the scene. Because of unknowns with the final output requirements of the thesis project at the time, creation of high-poly models was carried out with physical accuracy in mind. The assumption was that if the final rendering was done with a real-time engine, the high poly models would be used for normal baking and not the rendered geometry. A variety of photographic reference was heavily utilized to create high-fidelity detail in each model. At this point the focus was on two specific areas of the overall scene, those being the clock tower and the train station platforms.

To create these high-poly models a low-to high hard surface modeling workflow was selected. First, a simplified 'base' mesh was created. Lacking complicated shapes or detail, this mesh could easily be manipulated and measured. Then, after defining the basic shape and silhouette of the model, edge control loops were added to properly constrain the mesh when using smooth shading. While this

allowed edges to be moved to correct positions quickly and easily in the base mesh, it meant that the resulting final mesh was created without the topology as a primary concern.

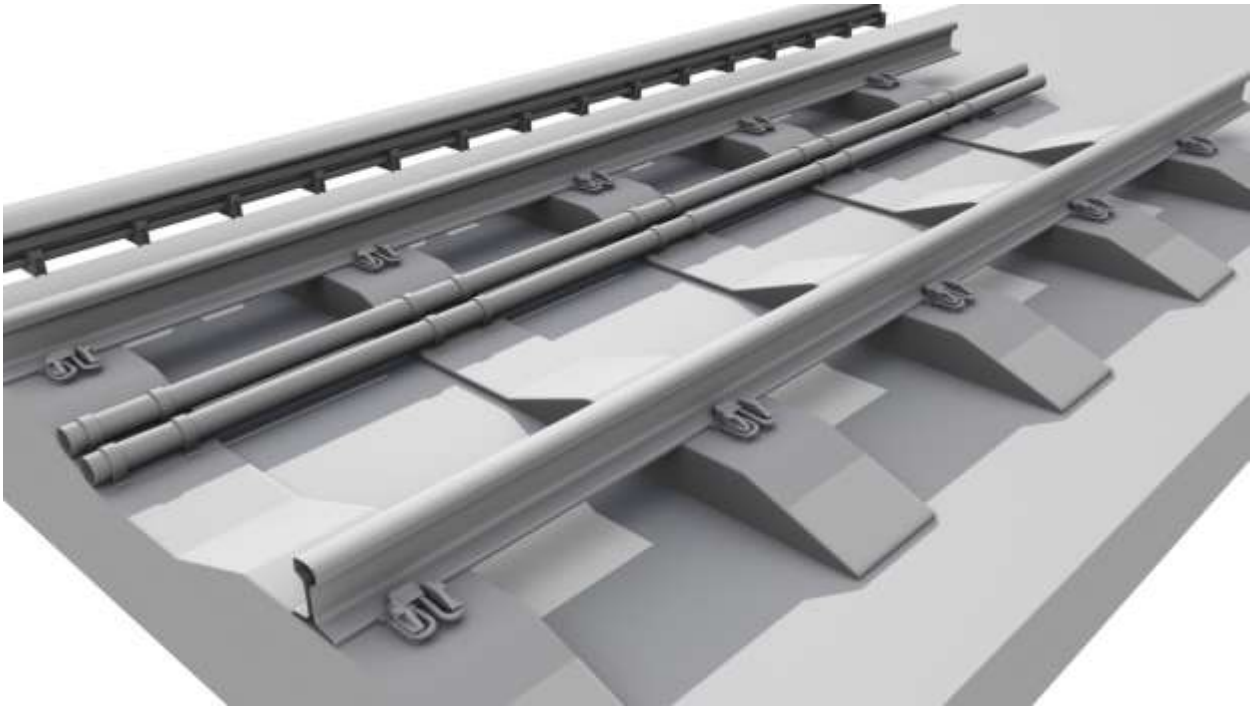

Figure 3.

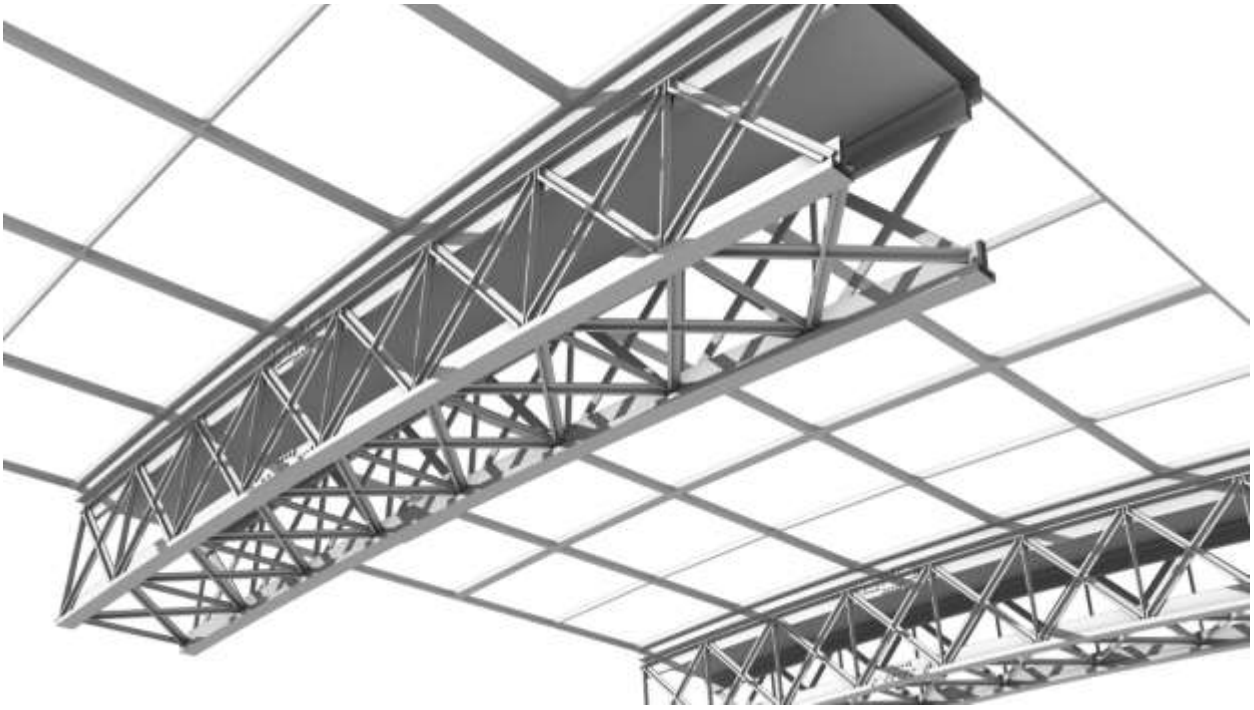

Figure 4.

The above figures are good examples of the level of visual detail this process produced. Considering the train tracks (*Figure 3.*), each individual clip and mounting bracket for attaching the rails to the ballast has been fully modeled. With the station roof arches (*Figure 4.*), the threads on the bolts holding the connecting trusses together have been modeled. Physical accuracy was researched and implemented at all levels of this process, meaning the rails of the train tracks are the proper width apart for a standard European rail transport line using electrical propulsion, and the platform height above the tracks was consistent with most European stations.

This high-poly process would eventually create problems later in the project, for several reasons.

1. While this level of visual fidelity is appropriate in some circumstances for prerendered sequences and cinematics, it was not feasible for use in a real time environment. Real-time rendering engines have stricter technical limitations that these models were not in compliance with. The assumption was that if a real-time rendering solution was chosen, the high poly models could simply be used for baking normals. However, given that at this point a decision had not been made as to how the final output of this scene would be presented, this was a mistake. Normal baking often requires a certain style of high poly model with respect to edge softness, sharp angles, and floating geometry. This could have been kept in consideration when constructing these models in order to make them suitable for either scenario. Ideally though, a decision on a rendering solution should have been made prior to the modeling process.

2. The level of detail was applied universally to all models without consideration to their location and purpose within the environment. For example, if planning on keeping the primary visual focus of the environment tied to a human perspective, then the modeling fidelity in the scene should be densest around both the ground level and major focal points. Modeling the threads on bolts more than sixty feet above the intended perspective was completely unnecessary and pointless. Even in these renders focusing specifically on the assets themselves, the bolts on the roof trusses are not even visible. The disregard for efficient application of detail at this point ended up being one of the most glaring issues faced during the entire project.

3. The level of detail was applied without consideration to the scope of the project. The scale of the planned scene did not allow for this level of detail on all meshes working within time constraints. By focusing too heavily on creating extremely high fidelity assets, the amount of work necessary to

complete the entire scene with a similar level of detail increased exponentially. There was a failure to fully realize how the size of the planned environment should affect the fidelity of the individual assets at this point in the thesis project.

Building for a Real-Time Renderer

At this point in the thesis project, a final decision had to be made as to how my final output would be rendered, as that would dictate a great deal of what methods would be used to create the remainder of the assets in the environment. Taking into account new experiences with coursework at the time and future career plans for employment in the gaming industry, a real-time engine was chosen as the method to render the final product of the thesis.

The next step was to begin the process of converting the workflow to one better suited for creating assets for a real-time environment. This required creating low poly models for the existing high poly assets that had already been built as well as creating any new assets with the specifications of the newly chosen rendering solution in mind. However, as stated earlier, there were problems with the existing high poly models that made them non-optimal for normal baking.

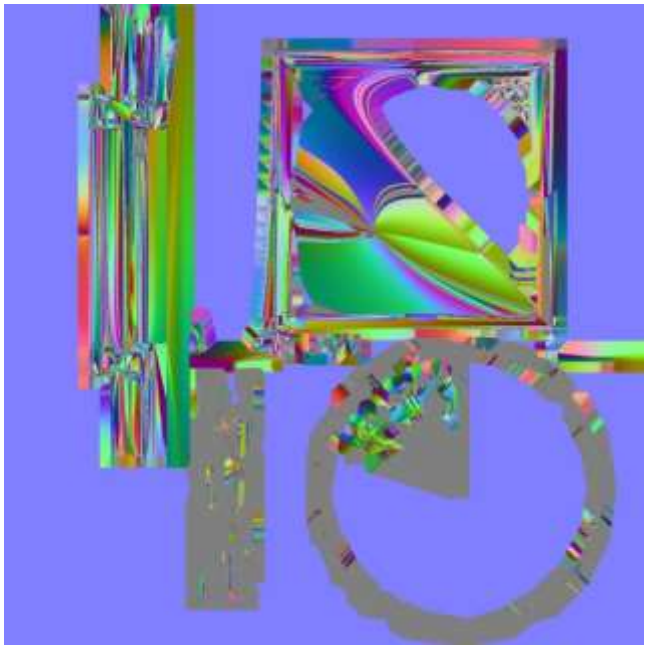

Figure 5.

Figure 5 shows the first result of a series of attempts to bake down the high poly clock model that was created for the corner tower. There were several issues in the geometry and design of the model that prevented a clean bake. Much of the geometry ends in right angles that directly face the baking target, which creates a very low intensity normal. A large percentage of the high poly elements are floating geometry, which can create baking artifacts if the cage is not properly adjusted, or if the low poly target is not well optimized. In this particular case, there was too much floating geometry and the low poly target model had too much discrepancy with the high poly source to create a workable bake. The choice of baking software also caused some issues. XNormal was used to bake all of my normal maps. Unfortunately, that package does not accept non-manifold geometry in either the high poly or low poly meshes, and the previous high poly workflow did not take this limitation into account. The high poly geometry of the clock face had to be heavily corrected before even these flawed results could be achieved.

XNormal was chosen for baking because it has a synced normals workflow that is compatible with Unreal Engine. This allowed for a more efficient vertex count in the low polygon models without sacrificing normal map fidelity. Further research into this issue was largely influenced by Joe Wilson's work on Polycount, along with the Unreal Engine documentation on the subject. Synced normal information refers to the way in which the normal data is interpreted. Different rendering software interprets the RGB lighting information from a normal map in different ways, and depending on how the normal map is baked this can cause errors or artifacts when it is rendered. XNormal can be set up to bake maps that are calibrated to be consistent with the way Unreal Engine reads and renders the RGB normal data.

Techniques for a synced normal workflow in Unreal Engine rely on utilizing the normals from the normal map over the normals from the imported geometry. Without a synced normals workflow, the rendered lighting data displayed in the engine is a composite of the geometry normals and the normal map. During the baking process, too harsh of a discrepancy between the high and low poly geometry can result in distorted areas in the normal map. These discrepancies are caused by the baking software attempting to stress the map to fit the target topology. When applied to geometry, composite lighting using a distorted map will manifest as lighting errors. With a synced normals workflow, settings are configured in XNormal that instruct Unreal Engine's renderer to override the geometry normals with the RGB data read from the normal map. This produces a better result because the final rendering is a direct interpretation of the normal map and is not altered in any way. Because the normal map is now the sole

source of the normal lighting, all edges can technically be softened without any distortion in the final result. Since lighting is calculated per vertex, and soft edges share vertex normals, this can greatly increase rendering efficiency. However, because vertex normals on UV seams are automatically split regardless of the normal map data, edges along these seams are left hardened to reduce any remaining distortion in the normal map bake.

The same baking issues encountered with the clock geometry were consistent with the rest of the older high poly assets that were created prior to the decision to use a game engine as the final renderer. This meant that time had to be taken out of new asset production to correct existing assets to make them suitable for use with a real-time rendering workflow.

To try and find a more efficient way around this problem, alternative solutions for creating normal maps were explored. A majority of research into these solutions came from the game industry website Polycount. These other methods would be used in conjunction with high poly baking to create the final normal maps for the scene assets. One method, which was used to create the normal maps for the face of the clock, involved the use of a greyscale height map (*Figure 6., below*) that is then converted to an RGB normal map (*Figure 7.*). The advantage of using a height map as the basis for the normals is that height maps can easily be created in Photoshop since in many cases it is simply a matter of creating vector or raster shapes to form the silhouette of the final map. The most significant drawback to this method is that it works best with maps that are applied to planar surfaces and it is very difficult to use with organic meshes.

Figure 6.

Figure 7.

There are several ways to convert the height map to a normal map. The method used in this instance was to use a normal map texture editing tool such as nDO2 to convert the height map directly to a normal map. The advantage to this workflow is that it can be completed entirely within Photoshop, without having to export to another program. Another method is to import the height map as an alpha mask into a sculpting program such as ZBrush. This method in effect allows for 3D editing of the height map, which is especially useful for adding wear and tear or for making changes to offset of the map. For the clock face, this method was unnecessary to achieve a satisfactory result, although it is useful in some other instances.

The other major process used to generate normal maps without baking high poly geometry was to utilize the normal texture editor nDO2 to generate normal maps from a diffuse texture. This is very beneficial because there is no need to later match diffuse and specular color to the normal map. Much like height map conversions, this works best on planar geometry and is difficult to utilize with organic meshes. This method was employed when working with the stone brick texture for the main structure of the tower. The detail normals of the stone were blended with normals from baked geometry, a technique documented by Paul Tosca.

Due in part to the issues with the older high poly geometry, and the concerning scope of the original concept environment, the objective for this thesis was amended. To better work within time constraints of the project, creation of the clock tower as a standalone work was designated as the final goal of the project, and several concept sketches to accommodate this design change were made.

Choosing a Game Engine

After choosing to implement a game engine as the means for completing the project, new research was conducted as to which software package would be the best fit for the project. This occurred concurrently with the ongoing efforts to adapt existing assets to the new workflow. At this point in the project, two options seemed most appropriate for implementing the planned scene; Unreal Engine 3 (UDK), and CryEngine 3. Both had their own benefits, but the final decision would be based on which engine best fit the desired goals of the project. Based on investigation into the official documentations for each engine, it seemed that CryEngine 3 had features that would best suit it to the goals of the project. Nevertheless, hands-on experimentation with both engines was conducted to gain a deeper understanding of the capabilities of each and make a final decision.

CryEngine 3 has gained an excellent reputation based on several key features. The most significant of these is the fully dynamic global illumination and shadowing system that CryEngine uses. Unlike Unreal, which has to precompute indirect lighting and shadows, CryEngine's renderer is much more of a 'What you see is what you get' solution. However the lighting appears inside the editor is how the lighting will appear during runtime. Not only does this speed up the lighting process considerably, but it also allows for very large areas to be lit very quickly. The other major feature that defines CryEngine is its well-developed set of terrain generation tools, which allow for water, roads, rocks, vegetation, and other small details to be placed into the environment quickly and easily. These environment tools, combined with the full dynamic global lighting, mean that CryEngine is oftentimes best suited for large outdoor environments that incorporate a large number of meshes and dynamic elements such as water or particle effects. Based on these features, CryEngine seemed like an excellent choice of engine. However, these features also come with certain limitations.

CryEngine has a more complicated file structure than Unreal Engine 3, and the asset import process is more involved as well. The most daunting difference between these two engines is that CryEngine uses a very different type of material editor from Unreal. Unreal Engine uses a node based material editor that allows for a great deal of freedom in creating various material effects. CryEngine's material system is mostly based on a material settings system that has a list of prebuilt settings that can be adjusted or turned on and off. Although this setup is not necessarily inferior to Unreal Engine's material setup, it was very unfamiliar. Having no prior experience with the new system, it was ultimately decided that the lack of proficiency with CryEngine's material system would be too much of a hindrance. This, along with the revised project concepts that did not incorporate large sections of the original scene, led to a decision to utilize Unreal Engine 3 for the final rendering engine.

Construction of assets for the newly defined tower scene continued from this point using technical specifications for Unreal Engine 3, and eventually Unreal Engine 4.

Physically Based Rendering

In the first several months of 2014, the concept of Physically Based Rendering, or PBR began gaining attention within the game developer community. Development of games utilizing PBR such as *Remember M* by Don'tnod Studios and creation of projects using PBR-capable rendering systems such as Marmoset Toolbag indicated noticeable improvements over traditional rendering engines. A

presentation at SIGGRAPH indicated that the next generation of the Unreal Engine would utilize a PBR system. Research into the topic ensued, mostly utilizing documentation from Marmoset and descriptions of the rendering setup used in *Remember Me*. Following the public release of Unreal Engine 4 in March of 2014, it was concluded that PBR actually had the capacity to improve the final result of the thesis project. There are several differences between PBR and traditional real-time rendering that make it an attractive choice for this thesis in particular.

In a PBR system, lighting is calculated in more physically accurate way that traditional rendering. In current real-time rendering the specular highlights of reflected light from a material are calculated independently of the diffuse color. Two separate maps control the intensity and appearance of the diffuse and specular lighting. In real life however, any light that is diffused from a material cannot also be reflected. The amount of light that is reflected vs. diffused is limited by the total light striking the surface. This is the basis for how PBR is calculated. Practically what this means for artists is a change in what textures are used for material input, as well as how those textures are created.

Instead of a diffuse map, PBR uses a texture called an albedo map (known in Unreal Engine 4 as a base color map). Unlike a traditional diffuse map, albedo maps cannot contain any directional lighting or ambient occlusion within the texture. In a standard rendering system, artists often add additional lighting information to the diffuse texture to complement the lighting calculated by the game engine. With PBR, the lighting calculated by the engine is sufficient such that this is no longer necessary. Only the base color of the material is necessary to make the albedo map. This saves time when creating these textures, as the additional lighting information no longer needs to be created by hand.

Specular maps no longer exist in a PBR workflow. Specular reflection is a set value for the entire material and no longer requires a texture map input. This set value is calculated to provide a more physically accurate appearance. Again, more time is saved by not having to create an additional texture map.

Most of the lighting data that would have been included in a traditional specular map is now expressed in a roughness map. This texture controls how smooth or rough a surface is, which directly affects the power of the specular reflections. This information used to be divided between the specular and gloss maps. Combining the information into a single map not only saves additional production time, but produces a more accurate appearance of materials, especially metals.

Normal maps remain exactly the same in a PBR workflow, except that much more of the lighting and shadow data for the mesh now relies on the normal map.

Following extensive research into PBR, it was concluded that the benefits provided by this new rendering system would provide a much better final product, and that any time it would take to convert existing textures to the new system would be made up for by the more efficient texture creation pipeline that PBR relies on. Fortunately, the user interface and Material Editor in Unreal Engine 4 are almost identical to Unreal Engine 3, which allowed for quick and efficient transfer of the thesis project to the new engine. Overall, the results of this change were satisfactory and it was concluded that PBR was a viable solution to complete the final scene with.

Conclusion and Lessons Learned

A significant result of this thesis is a new understanding of the best methods for creation of an environment from start to finish. Knowledge of many new techniques and processes for the creation of the individual assets is incredibly useful, but the most important experience gained was with the overall procedures for environment workflow.

Two very important elements of the workflow that were not fully implemented in this project were the concepts of software lock and feature lock. These practices are used in the gaming industry to ensure timely and completion of projects. Software lock is the point at which the final software packages both for creating and rendering assets are decided upon, and are not changed once the decision has been made. Feature lock is the point at which the scene contains all of the assets that will be in the final project, and a decision is made to add no new assets, but to complete and refine the ones already implemented. What the final product of the thesis project would be was not established definitively until a very late stage in the process. This resulted in several tentative targets for final results, which in turn caused the different iterations of development to be completed using different workflows that were not always compatible. The failure to incorporate the elements of software and feature lock into the workflow of this thesis was the source of many of the issues encountered.

One of the largest challenges in the entire project was the fact that most of the assets for the final scene were modeled and textured independently of each other from beginning to end. This led to lots of corrections late in the production process. In future projects, production for all assets should take place on the same development timeline, which allows for a much easier process of assembling the final

scene. This also prevents any one asset from becoming too detailed or inconsistent with the rest of the scene, and keeps the big picture in mind. In addition, creating a more detailed and precise blockout allows for an easier process of determining the composition of the environment and creation of modular assets. Overall, the entire scene should be built using an iterative design process that facilitates a gradual increase in detail and complexity. This type of workflow also allows for feature lock to be implemented much more easily, as the entire scene can be reviewed at once and creation of additional assets halted without compromising the basic integrity of the scene.

As a direct result of this thesis, new knowledge of different environment production methods has been gained, as well as the expertise as to when each of them is most applicable. Practice using a wide range of new tools and utilities to aid in the creative process is another benefit. Overall, completion of this thesis has resulted in valuable experience with the process of creating environments for games that will allow for continuing building of knowledge and abilities in the future.

References

Lagarde, Sebastien. "Feeding a physically based shading model." N.p., 17 Aug. 2011. Web. 14 Apr. 2014. <http://seblagarde.wordpress.com/2011/08/17/feeding-a-physical-based-lighting-mode/>.

Mathis, Ben, Krzysztof Dolas, and Evan Herbert. "Normal Map." *Polycount Forum*. N.p., 24 Jan. 2013. Web. 14 Apr. 2014. <http://wiki.polycount.com/NormalMap#Normal_Map>.

Simeth, Jared. “Matte Painting for Production with Jared Simeth.” *Gnomon Workshop*. 2010.

Russell, Jeff. "Basic Theory of Physically-Based Rendering." *Marmoset*. N.p., 1 Jan. 2013. Web. 14 Apr. 2014. <http://www.marmoset.co/toolbag/learn/pbr-theory>.

Schulz, Nicolas, and Adam Johnson. "CRYENGINE SDK Documentation." N.p., 13 Apr. 2014. Web. 14 Apr. 2014. <http://docs.cryengine.com/display/SDKDOC1/Home>.

Seymour, Mike. "Game environments – Part A: rendering Remember Me." *FXGuide*. N.p., 4 June 2013. Web. 14 Apr. 2014. <http://www.fxguide.com/featured/game-environments-parta-remember-me-rendering/>.

Tosca, Paul. "Making of Varga." N.p., 1 Jan. 2007. Web. 14 Apr. 2014. <http://www.paultosca.com/makingofvarga.html>.

"Unreal Engine 4 Documentation." N.p., 1 Jan. 2014. Web. 14 Apr. 2014. <https://docs.unrealengine.com/latest/INT/index.html>.

Wilson, Joe. "Making sense of hard edges, uvs, normal maps and vertex counts." *Polycount Forum*. N.p., 18 Oct. 2012. Web. 14 Apr. 2014. <http://www.polycount.com/forum/showthread.php?t=107196>.

Wilson, Joe. "PBR in Practice." *Marmoset*. N.p., 1 Jan. 2013. Web. 14 Apr. 2014. <http://www.marmoset.co/toolbag/learn/pbr-practice>.