Optimization of 3D Game Models

A qualitative research study in Unreal Development Kit

Av: Laura Lohikoski och Elin Rudén
Handledare: Petri Lankoski
Abstract

Our goal with this study is to examine how much optimization of 3D game models can affect the overall performance of a game. After a previous pilot study we decided on use a method where we worked with a 3D scene which was made earlier unconnected to this study. We created two versions of the scene in Unreal Development Kit, one with none of the meshes optimized and the second scene where the meshes are optimized. From these two scenes we wrote down the different stats: the draw calls, frame rate, millisecond per frame and visible static mesh elements as well as the memory usage. Comparing these stats from the two scenes, we found that there was a change in the stats. Draw calls and frame rate had dropped in the second scene, as well as the memory usage which made the game run more smoothly without losing much of its aesthetic quality.

Keywords

3D graphics, game performance, graphic optimization, 3D models
**Sammanfattning**

Målet med vår studie var att se hur stor skillnad optimering av 3D-modeller i spel gör för att förbättra spelprestandan. Efter att ha utfört en pilotstudie beslutade vi oss för att använda en tidigare byggd 3D-scen för undersökningen i vår C-uppsats. Vi skapade två versioner av scenen i Unreal Development Kit, en där inga modeller var optimerade och den andra där vi optimerat modellerna. Vi skrev därefter ner statistik från de olika scenerna, nämligen draw calls, frame rate, millisecond per frame och visible static mesh elements liksom minnesanvändning. Efter att ha jämfört resultaten såg vi att det fanns en väsentlig skillnad mellan scenerna prestandamässigt. Både draw calls, frame rate och minnesanvändningen hade minskat efter optimeringen vilket ledde till att spelet kördes smidigare.

**Nyckelord**

3D-grafik, spelprestanda, grafisk optimering, 3D-modeller
# Table of content

Abstract............................................................................................................................................. 2
Sammanfattning.................................................................................................................................. 3
Table of content................................................................................................................................. 4

1 Introduction .................................................................................................................................. 5
   1.1 Background ............................................................................................................................ 5
   1.2 Related Research .................................................................................................................... 5
   1.3 Research Questions ................................................................................................................ 6
   1.4 Essay structure ....................................................................................................................... 7

2 Literature ..................................................................................................................................... 8
   2.1 Poly count and textures .......................................................................................................... 8
   2.2 CPU/GPU and Memory .......................................................................................................... 8
   2.3 Level Optimization ................................................................................................................ 9
   2.4 Level of detail / LODs ........................................................................................................... 11
   2.5 Performance Profiling in Unreal Development Kit ................................................................. 11
      2.5.1 Memory Profiling ........................................................................................................... 12
      2.5.2 Render Thread Profiling ................................................................................................ 12
   2.6 Interview with a 3D expert .................................................................................................... 12

3 Method ......................................................................................................................................... 14
   3.1 Mesh Reusability and Aesthetics .......................................................................................... 14
   3.2 Performance and Memory Profiling ...................................................................................... 15
   3.3 Scene 1 .................................................................................................................................. 16
   3.4 Scene 2 .................................................................................................................................. 17
   3.5 The Creation of the 3D Models ............................................................................................. 18
      3.5.1 The Roofs ...................................................................................................................... 18
      3.5.2 The Walls ...................................................................................................................... 19
      3.5.3 The Vegetation .............................................................................................................. 19

4 Results ......................................................................................................................................... 20
   4.1 Stats ...................................................................................................................................... 20
   4.2 Aesthetics in Scene 1 and Scene 2 ....................................................................................... 21
   4.3 Comparison of the Results .................................................................................................... 22
   4.4 The Importance of 3D Game Model Optimization in UDK ................................................... 23
   4.5 Optimization within the boundaries of aesthetics and mesh reusability ............................. 24
   4.6 Self-criticism ........................................................................................................................ 27

5 Conclusions ................................................................................................................................. 29

References ........................................................................................................................................ 31
Appendixes ...................................................................................................................................... 33
1 Introduction

1.1 Background

Working as a 3D artist in the game industry not only requires a sense of aesthetics, but also an understanding of how game engines work and how one can optimize 3D models and textures\(^1\). This is an important part when creating games since the memory and the computer’s graphics card as well as its central processing unit are limited. If these are exceeded the game's frame rate will be low and the game will not run smoothly. Furthermore, optimization of game graphics must be done within certain boundaries, such as the aspects of aesthetics and mesh reusability\(^2\) in order not to lose the feeling of the game and to avoid repetitiveness both when it comes to textures and 3D models.

Furthermore, the matter of poly count\(^3\) is a topic that is often being discussed among people working with or studying 3D game graphics. The topic is frequently brought up for discussion on game developers forums such as Polycount (2013a; 2013b). Since the technology of computers and software such as game engines is evolving at breakneck speed, the limits for poly count for 3D game models increases. A few years ago it was really important to keep the poly count for game models down as much as possible in order to save memory, whereas today it is sometimes considered better to add detail to a mesh than to implement it through additional texture maps in order to save time.

The purpose of this study is to gain knowledge concerning graphics optimization in video games with focus on mesh optimization within certain boundaries such as the game’s aesthetics. We are doing this by creating multiple scenes in Unreal Development Kit (Epic Games, 2012) where we investigate how different components, that is poly count and so called normal maps\(^4\), affect the performance within the boundaries of mesh reusability and aesthetics.

1.2 Related Research

We conducted a pilot study in order to decide upon a method that we could use for the practical part of this bachelor thesis. To decide how to approach the subject of 3D model optimization for games, we looked at previous literature that had been written on the subject. We are presenting the most significant sources in this study as well.

However, there are not a lot of written reliable sources to be found on the subject of optimization of game graphics. The literature that we found when conducting a pilot study is foremost tutorials written on the internet by game

\[\begin{align*}
\text{1 Images that are applied to the surface of a 3D model.} \\
\text{2 To be able to reuse the same 3D model.} \\
\text{3 The total amount of polygons found in a 3D model.} \\
\text{4 A texture map used for simulating fake details and bumps on a model.}
\end{align*}\]
developers that discusses the matter of optimization, CPU/GPU and memory use. Furthermore, we have found some handbooks that mention the subject. Unfortunately these are quite outdated, hence we also decided to interview a game developer to get a better picture of how game companies work with game graphics optimization today. A lot of the texts we read mentioned both poly count as well as how to optimize textures for a scene. Other texts written by De Jong (2011) and Unreal Development Network (2013c) discussed performance profiling in Unreal Engine 3 that is going to be part of our method when we are going to conduct the practical part of our bachelor thesis. We are building multiple scenes in Unreal Development Kit and examining how different aspects of 3D game models affect the game’s performance. We are focusing on aspects such as poly count mentioned by Omernick (2004), the combining of meshes, the use of normal maps and the creation of modular sets5.

After reading about the subject of optimization we needed more up-to-date information on where the main focus lies when it comes to game graphics optimization today. In order to be able to find the right method for this bachelor thesis we had a conversation with a 3D expert who had agreed to help us. The conversation with the 3D expert strengthened the information we had gathered from other sources and helped us decide upon suitable method.

The most up to date information that could be found were on online forums, but these consisting of discussions and personal opinions, we deemed the information unsuitable for our study.

1.3 Research Questions

The purpose of this study is to contribute with knowledge about the importance of 3D model optimization when creating a game. This is done by examining how different aspects of 3D game models affect a game’s performance. We are looking at optimization of 3D models, but within certain boundaries, such as aesthetics and mesh reusability. In order to do this, we created scenes in Unreal Development Kit from a concept, where we built two scenes, one optimized and one unoptimized. The scene that we are going to work with and optimize is one that one of us made a year ago for an environment art assignment in the University. Since we already have all the assets we do not have to create everything from scratch, but can instead focus on the optimization and unoptimization of the scenes.

The main focus of the study lies on investigating how optimization of 3D game models can improve the games performance. Another important part of our essay is the qualitative comparison between using 3D models with a lot of modeled detail with 3D models that acquires detail through texture maps such as normal maps. We are investigating this within certain boundaries, being aesthetics and mesh reusability.

5 A workflow where you create modular polygon model sets that are easy to reuse and place in a game editor.
The research questions that we aim to answer with this essay are:

1. How can you create an optimized 3D environment for a game in Unreal Development Kit without losing the important aspects of aesthetics and mesh reusability?

2. Is it worth adding more detail to 3D models in a scene in order to avoid using an additional texture map such as a normal map?

The focus on this study is limited to optimization of 3D game models. We are not looking at texture optimization or level optimization that is done within the game engine. We are solely optimizing the models in the 3D software that we are using when creating the models.

1.4 Essay structure

Next we are presenting previous literature on the subject of game graphics optimization and performance profiling in Unreal Development Kit (UDK). We shortly summarize our expert interview results that formed the basis of our method for the empirical part of this essay (Lohikoski and Rudén, 2013). Under 3 Methods we thoroughly describe our method that we are using for our empirical part of the study in order to investigate how different parts of 3D game model optimization effects the performance and how much one can optimize within certain boundaries such as aesthetics and mesh reusability. Under 4 Results we present the results of the empirical part done in Unreal Development Kit and we discuss our findings. We conclude our findings and propose further research topics on the subject under 5 Conclusions.
2 Literature

2.1 Poly count and textures

When it comes to poly count and poly reduction in games it all comes down to what the player will see in the scene and from what perspective the player will see it. Omernick (2004) explains the basics of poly reduction in his book Creating the ART of the GAME. He states that when reducing the poly count on a polygon model you should ask yourself what the player will see and delete unnecessary parts or faces of the 3D models that will not be seen in the game. You should also consider if all parts of the model are important or if they could be deleted in order to reduce the poly count. The third thing that Omernick mentions is to consider if the tessellation/subdivision level of the model is too high. He also states that a lot of details does not need to be modelled but can be implemented through texture maps. (Omernick, 2004)

2.2 CPU/GPU and Memory

There is a common misunderstanding when it comes to performance and the importance of the memory and the CPU/GPU. Sjoerd de Jong (2006) writes about the subject on his website. He writes that it is not always faster for the PC to use the same mesh multiple times instead of using unique meshes when building a level in a game engine. It all comes down to balancing in order to stress the memory and the CPU/GPU equally, even though the CPU/GPU generally has more influence on the frame rate. De Jong writes that “The memory can make things run smoother; the CPU/GPU makes them run faster” (de Jong, 2006).

De Jong (2006) continues by explaining that when it comes to rendering an object in a game engine, such as Unreal Development Kit, the engine starts with cutting the meshes into pieces and renders the different materials applied to an object one material at a time. For example, if you have a mesh with one material applied to it, it renders the whole object at once, but when you are using two materials on an object the engine has to render the object twice with the different textures after cutting up the mesh into triangles. The CPU/GPU cannot render multiple triangles with different textures at once and the sorting of the pieces that need to be rendered requires power and time.

Another important part to have in mind when it comes to real time rendering in a lot of game engines such as UDK, is according to de Jong (2006), the fact that every mesh is rendered separately. Every mesh and materials are divided into

---

6 The method used in order to reduce the poly count.
7 The amount of subdivisions found on a 3D model.
8 Central Processing Unit.
9 Graphics Processing Unit.
10 Another word for a polygon model.
sectors that require draw calls\textsuperscript{11}. Large amount of polygons can be drawn as long as the engine can mass process them at the same time. Consequently, it is good to combine objects into single meshes when possible. (de Jong, 2006)

Holding back on the amount of textures used is also a good idea to keep the performance at a reasonable level, since rendering each texture requires a draw call. Textures are often more consuming than meshes. The time that can be saved when rendering in real time is not to be underestimated, even though it is only a millisecond. It is a lot of time for the CPU/GPU when working on a game with hundreds of meshes and textures that need to be rendered in a desirable amount of frames per second. (de Jong, 2006)

De Jong (2006) states that it is the memory that sets limits for making too big combined meshes of objects and to make unique textures for each mesh. Thus, there need to be a good balance between the memory and the CPU/GPU load. Making meshes and textures unique will benefit the CPU/GPU, while it will result in a higher workload for the memory. On the other hand, when producing a lot of reusable meshes and textures the workload on the CPU/GPU will increase, hence, the importance of balance between the two elements.

De Jong (2006) believes that it is also essential to have in mind that when it comes to memory, unique textures are worse than unique meshes. While a single texture can take up large amount of memory space, a single mesh is less expensive memory wise. This results in that one may want to reuse meshes with higher poly count, while making unique examples of lower poly meshes, in order to make the best use of the memory available.

One issue that de Jong (2006) mentions with combining multiple objects into one mesh is that a lot of game engines render an object when it comes into frame, and with bigger objects or group of objects it is a higher chance that the triangles of the objects comes into view. A method used for combining and rendering a group of identical object at the same time is called batching.

2.3 Level Optimization

In his text Provost (2003b) mentions different levels of optimization and how sometimes it is not the meshes that need to be worked with but the level design. What he means is that depending on the level, if it is a room or an open space, the optimization differs from mesh to mesh. If an artist knows how big the visibility spectrum is on the level that they are working on, they know how much optimization needs to be made. For example, a big environment with a wide visibility spectrum, such as a garden or a ballroom, needs more optimization on the meshes and textures because more of it will be rendered in a scene. On a scene with lower visibility spectrum like a room the optimization does not need to be as severe nor as restricted due to the fact that there is less to render in the scene.

\textsuperscript{11} The amount of objects, such as textures and meshes that the GPU needs to draw on the screen.
There are three main things one ought to take into account to achieve constant good performance: Vertex density, texture density and visibility spectrum. A good balance between these three will result in a good scene complexity which will keep the frame rate low and performance well. (Provost, 2003b)

Vertex density is how tightly packed vertices are in a given volume of space (such as a mesh). A small area filled with high detail objects will affect the performance rate to drop. Assets that are high in vertex density should be distributed evenly across the playing space and place them in scenes and environments where the visibility spectrum is small. Distributing these assets evenly and strategically might save more time than optimizing each asset. (Provost, 2003b)

Texture density is the texture memory usage of a given area in the game. Concentrating lot of different textures into a constrained location can cause a breakdown in the frame rate since the area needs to be downloaded to the graphics processor. (Provost, 2003a) Here it is also important to distribute it evenly across the visibility spectrum (Provost, 2003b). Also, the textures need to be loaded on the memory and too many textures may cause a chain of deleting and loading a texture to the memory.

The visibility spectrum is the set of all visible space in a location. The vertex count and texture space one can put into the visibility spectrum is constant, so the larger the space is the less dense the details can be packed into it. The smaller the visibility spectrum is, the more detail can be put into the scene. (Provost, 2003b)

The visibility spectrum is the single most important aspect affecting rendering performance. Techniques commonly used to decrease the size of the visibility spectrum includes closing doors to rooms, transition ones that block the view and fog of depth-of-field effects in external unoccluded environments. (Provost, 2003b)

According to Provost (2003b), when a scene is rendered, two fundamental things happen. The vertices get transformed and its triangles get drawn. These two things happen parallel so whichever is the slower one determines how fast the asset is rendered. Assessing what needs optimization in a surface one needs to identify the bottleneck and the cost. The cost will tell you whether you need to optimize it, and the bottleneck tells you what to optimize.

If it is the transformative time which is causing the bottleneck then it is said to be “transformative-bound” and the cost is the amount of time it takes to transform the vertices attached to it. If an objects cost is its fill time then it's said to be “fill-bound” and the cost is the amount of time it takes to draw the surface on screen. Fill-bound meshes has different optimization rules than transformative-bound meshes so an artist needs to develop an intuitive sense which one a surface might become in an engine. A rough simplification on the rules of optimization concerning fill-bound and transformative-bound meshes is whether the object needs texture optimization or mesh optimization. Our research is on the

---

12 The cost tells whether optimization is needed or not, and the bottleneck what needs optimization.
optimization rules often used for transformative-bound meshes, in other words mesh optimization, whereas fill-bound meshes often need to concentrate the optimization of the materials and textures. In practice interactive articulated bodies are usually transformative-bound, walls and ceilings tend to be fill-bound and props lie somewhere in between. (Provost, 2003b)

2.4 Level of detail / LODs

Making Level of detail models, shortened LODs, is a way of optimization that is frequently used when developing games. Valve Developer Community (2012) describes the process of making LOD meshes on their website. LODs are low detail versions of the same mesh that are to be used/or switched to from certain distances in a game level in order to ease the processing of the models.

Valve Developer Community (2012) states that when creating LODs of models for games it is important to keep certain things such as vertex positions, normals and UVs consistent between the different LOD models for the same mesh, in order to keep the memory low. The LODs should also be created when the parent model as well as the texture for the model is finished.

2.5 Performance Profiling in Unreal Development Kit

Profiling means to locate the part of the game that is causing issues when it comes to performance. (de Jong, 2011) On a support site for Unreal Development Kit (Unreal developer network, 2013c) it is explained how to measure performance in the game editor.

We use GPU and memory profiling in order to understand how the scene impacts the performance. On the website of Unreal Developer Network (2013d) the different tasks done by the GPU and the CPU are described: The CPU is responsible for updating and calculating the game each frame, while the GPU renders the game by drawing polygons and textures and handling shaders, post processing effects etc. (Unreal developer network, 2013c)

De Jong (2011) describes how to measure how different parts of the game, such as lighting, meshes and code affect the performance. First of all, it is import to play in the real game and not in the Play-In-Editor game in order to get the right results when profiling. The game starts with the following command line options: -log (shows what the engine is doing when loading a level), -remotecontrol (Opens the remote control developer window by writing rc in the console), -windowed (in order to have easy access to the remote control and log window) and -NoVSync (turn the frame rate cap off). (de Jong, 2011)

The game is run with the command lines STAT FPS and STAT UNIT that displays the frame rate (fps) and milliseconds per frame (ms), as well as the milliseconds per component. The milliseconds show how long it takes to render a frame and the stats unit shows the time in milliseconds divided between the GPU and the CPU. (de Jong, 2011)
2.5.1 Memory Profiling

When developing games it is important to keep the memory usage on a reasonable level, hence it is a good idea to decide upon a memory budget in the beginning of a project which decides what to use the memory on. In order to know how much memory different parts of the game are using in a certain UDK level, use the STAT MEMORY command where stats such as the texture memory used in the level and the static mesh memory are displayed. (Unreal developer network, 2013b)

The texture pool in UDK is a fixed size and therefore the overall system memory cannot be exceeded by having too many textures. Instead, it is how the given memory is used that is important. As long as the textures within the area around the player seen by the camera do not exceed the pool size, the amount of textures does not matter. All textures, including light maps use the texture pool. (Unreal developer network, 2013b)

2.5.2 Render Thread Profiling

The render thread determines visibility and submits draw calls to the GPU. When profiling the render thread rendering statistics are accessed by using the Command STAT SCENERENDERING. The command STAT INITVIEWS displays important stats such as Visible Static Mesh Elements. (Unreal developer network, 2013d) To access stats on the amount of draw calls in the command line STAT D3D9RHI can be used (de Jong, 2011).

When optimizing a scene with the focus on render thread performance, the main goal is to keep the component count in the scene as low as possible, and minimizing the amount of unique materials on different meshes. When creating meshes for a level in UDK, UDN (Unreal developer network, 2013a) also mentions the importance of finding a balance between the usage of lots of reusable meshes, which will be hard on the rendering thread, and larger meshes that are less reusable.

2.6 Interview with a 3D expert

For our pilot study, we interviewed a 3D expert in order to get a better understanding of what is important when it comes to optimization of game graphics. Based on the result of the interview we then decided on a method that we are using when conducting our research study on game graphics optimization for this bachelor thesis. (Lohikoski and Rudén, 2013)

In the interview we discussed the matter of optimization and how we could investigate how different components such as textures and poly count in the terms of draw calls and memory affect the performance of a game. The 3D expert suggested that we should build multiple versions of the same environment in Unreal Development Kit where we examine how the different kinds of optimization influence the performance with the help of the profiling tools provided by the game engine. We asked him whether he thought that it would be better to focus on one aspect of game graphics optimization but he stated that we should focus on multiple aspects in order to be able to compare them.
Unfortunately we had to decide upon only one focus of game graphics optimization, being mesh optimization, since we did not have time to build more scenes. (Lohikoski and Rudén, 2013)

On the subject of poly count and the use of separate meshes/combined meshes he thought that it could be good to include it in the study as it has a big part of optimization and affects the draw calls. To begin with, we could build a scene where we use a lot of separate objects and where we do not pay much attention to the poly count. Thereafter, we build a second scene where we reduce the poly count and combine meshes in order to reduce the amount of draw calls needed. We can then use the profiling tools in UDK to compare the two scenes. (Lohikoski and Rudén, 2013)

We also discussed the balance between optimization and aesthetics. One should optimize a scene while considering the effects it might have on the game’s graphical profile with its visual clues and the overall aesthetic of the game. Another boundary we should have in mind when creating a scene with the combined meshes is the reusability of them when it comes to level design. (Lohikoski and Rudén, 2013)
3 Method

The empirical method we have decided upon using for our bachelor thesis is based on the results of our previous pilot study (Lohikoski and Rudén, 2013). When developing this method we have taken into account both what we have read in previous literature and what was said during an interview with a 3D expert.

We are using this method to gain more knowledge on how one can create optimized 3D models for a game environment without losing the important aspects of aesthetics and mesh reusability. With the scenes we also investigate how adding detail to a model through polygons affects the performance compared to adding detail through normal maps.

We are using Unreal Development Kit of Unreal Engine 3, where we build two different scenes with different focus on optimization of 3D game models. The scenes are based on the same 2D concept that we visually recreate in the 3D scenes. The concept can be found in the appendix. The two scenes, Scene 1 and Scene 2 are created with focus on adding details through either polygons or normal maps and are compared to each other within the boundaries of performance, aesthetics and mesh reusability. None of the meshes have collision on them in either scene. All components except from the textures and the 3D models are the same in both scenes.

We use the profiling tools in Unreal Developer Kit in order to investigate the differences in the performance among the scenes. We also look at the memory use of meshes and normal maps in the scenes. The results are discussed with the aesthetics and the reusability of meshes and textures in mind.

As we encountered problems when measuring the stats in UDK in the Play-on-PC mode we decided to use the Play-in-Editor mode instead. Since it is the comparison and the difference in stats between the two scenes that are important it should not make a difference in the results.

3.1 Mesh Reusability and Aesthetics

When creating the optimized scenes we do it within certain boundaries, being mesh reusability and aesthetics.

In terms of mesh reusability, it is important to create 3D models that can be used multiple times in a level of a game in order to make the most out of the given memory. As de Jong (2006) mentions it is not preferable from a level design perspective to create too big units of combined meshes, whereas the reusability level decreases. It is also not considered a good idea to make too small units that

13 An invisible space placed on the mesh, simulating a solid surface or wall in a game.
are to be placed in the level individually. We keep this fact in mind when creating our optimized versions of the polygon models.

When creating the optimized polygon models we try to optimize everything as much as we can, but without losing the aesthetics of the scene. Aesthetics and what we find attractive when it comes to video game art is very subjective. When optimizing the scenes we rely on our own artistic sense when setting the boundaries of what is aesthetically appealing and we decide the limit of optimization. Our decided limit of optimization is based on keeping the shape of the models when reducing the poly count and to not remove any relevant details that cannot be compensated through well baked normal maps.

Aesthetics in a game is a big part of game immersion and how the player perceives the game. Game developers work hard to push the graphics to the optimal state. This is where optimization and aesthetic views meet. The optimal game is a smoothly running game with beautiful graphics. Keeping this in mind we try to keep Scene 2 as closely resembling Scene 1 when it comes to the look of the game, but cutting out unnecessary performance thriving parts of the scene (when it comes to 3D meshes).

When importing the models into UDK and placing the meshes in the level we avoid making optimization that is mostly done by the level designer. Instead we focus on optimization that is mostly done externally in the programs Autodesk Maya 2014 (Autodesk, 2013) and nDo2 (Quixel, 2011) and that is part of a 3D game artist’s job. In UDK we place the meshes so that they match our concept and add meshes on the backside of, for instance, buildings as well, even though they might never be seen by the player. We do not make use of prefabs\(^\text{14}\) in our scene.

3.2 Performance and Memory Profiling

After creating Scene 1 and Scene 2 we are using the profiling tools of UDK in order to compare different aspects of performance and memory in the different scenes. The stats we are looking at and documenting are the following ones;

1. The amount of draw calls
2. Frame rate
3. Milliseconds per frame
4. Visible static mesh elements
5. Memory usage

When documenting these variables we enter the game and take 3 screenshots from an over viewing perspective of the scenes. Moreover we take one without any stats visible. We then use the stats on the screenshot with the lowest frame

\(^{14}\) Collection of actors that can easily be instanced.
rate and compare the stats from both scenes. Concerning the memory usage of the scenes, we documented it by summing the memory of all 3D models in the different scenes and added the memory of the normal maps to Scene 2 as well. The stats are documented when playing in the Play-in-Editor mode in Unreal Development Kit, since we encountered a problem that made it difficult to use the Play-on-PC mode. We use the console commands `STAT FPS`, `STAT UNIT`, `STAT INITVIEWS` and `STAT D3D9RHI` in order to measure the different variables that we seek. To position ourselves somewhere where we can take over viewing screen captures we also use the command line `GHOST`.

Since the computer we used for acquiring the stats is a laptop and also in bad condition and its graphics card as well as its CPU is quite outdated, the overall stats are bad and the fps is low. However this does not make any significant difference since the results are going to be interpreted as a qualitative comparison between the stats from the different scenes.

### 3.3 Scene 1

Scene 1 is created without any focus on poly count. When creating this scene we do not focus on reducing polygons that are not seen by the player and we model details instead of using normal maps. We also use what is often referred to as *korean bevels*\(^\text{15}\) to make the shading of the smoothed models look good. Korean bevel is a method used in 3D graphics where you add detail to a 3D model, such as *supportive edges*\(^\text{16}\) to make the shading around the softened edges of a 3D model look good and smooth. This method can be used instead of baking normal maps from high poly models that usually takes quite a lot of time.

We are using an environment scene that one of us created a year ago for a school assignment. The scene is at this state quite optimized in the sense of poly count and unnecessary geometry. Therefore, by adding geometry to certain parts of the polygon models and give it a higher subdivision level we make it less optimized in order to easier compare with Scene 2.

We decided on making the models with a polygon count in between what is considered as high poly and low poly. By high poly we mean smoothed models with a very high poly count in relation to its size and shape, in opposition to low poly count where the poly count is as low as possible in relation to its size and shape. It is difficult to define what is perceived as high poly and low poly, since it differs from object to object. When starting the practical work for this essay we discovered that UDK did not accept too high poly 3D models, therefore we had to settle for a poly count in the middle between high poly and low poly for the meshes for Scene 1. The poly count of the assets for Scene 1 is therefore in between 276 - 39 494 polygons per mesh depending on the size and shape of the models. In order to make the shading of the polygon models look good without

\(^{15}\) A method used in 3D game graphics where you add geometry to control the shading of models with softened edges.

\(^{16}\) Adding supportive edges near the corners of 3D models helps controlling the shape of the models when smoothed, as well as the shading when softening the edges.
Adding normal maps we decided upon using Korean bevels, where adding edge loops\textsuperscript{17} to certain parts of a model keeps the shading arbitrary. We also use ordinary bevels\textsuperscript{18}.

Increasing the subdivision level and making use of Korean bevels will increase the polygon count sufficiently. We also add geometry to parts of polygon models that are not usually seen, such as the bottom part of models. When creating models we try to keep as much as possible as separate meshes and not combine multiple polygon objects into one.

The materials used on the meshes in Scene 1 are different instance materials\textsuperscript{19} with only one diffuse texture\textsuperscript{20}. The instances are made from two master materials. The first one is being used for vegetation meshes, such as the trees, flowers and bushes that require two sided shading due to the use of planes. The second master material has only one sided shading and is used for the rest of the meshes in the scene.

### 3.4 Scene 2

Scene 2 is created as an optimized version of Scene 1 with the focus on low poly count and the use of normal maps in order to make the models look smooth and to add detail. We reduce the subdivision level and delete unnecessary parts of the polygon models that the players will not see. In order to do so we are using the methods of optimization that we have read about and that are described under 2 Literature.

We remove all supportive edges on the models and create high poly models in order to be able to bake normal maps\textsuperscript{21} in Maya that replaces Korean bevels and some modeled detail. We delete geometry that is not seen by the player and optimize the meshes within our limits. When creating the optimized models we try to keep the aesthetics on the same level as in Scene 1. We try to maintain the same shapes of the models and try not to remove detail that cannot be replaced by normal maps.

We make the polygon models as modular as possible in order to easily be able to reuse objects. We try to combine as much as possible without making the objects less reusable. When optimizing we also focus on not losing too much of the aesthetics.

The materials in Scene 2 are instance materials of two master materials. The materials are the same as in Scene 1, except from that the ones that are used for other meshes than vegetation include a normal texture.

\textsuperscript{17} Edges that run all way around a 3D model, forming a loop.
\textsuperscript{18} A way to round or chamfer the edges of a polygon model by adding geometry.
\textsuperscript{19} A material which is using the same setup as another master material, but where certain parameters, such as textures can be changed.
\textsuperscript{20} The most common type of texture map.
\textsuperscript{21} A process that let you bake down details from a high polygon model onto a normal map which can be applied to a low polygon model.
3.5 The Creation of the 3D Models

All 3D models are created in Autodesk Maya 2014 and the normal maps used in Scene 2 are either baked using the transfer maps tool in Maya or by creating normal maps from a reference picture using nDo2.

When creating the models for Scene 1 we follow our concept for the scene without altering the original design. Instead of focusing on decreasing the poly count we model the necessary details onto the meshes. For the models in Scene 2 we delete unnecessary polygons and details that can be added through normal maps. We avoid deleting polygons that create the main shape of the objects.

After finishing the game models we take screen captures of all the models in Maya’s viewport, placing the models used in Scene 1 in UDK on the right side of the viewport and the models used in Scene 2 on the left side. We take 2 captures of each asset, one with the wireframe of the model visible and one with wireframe turned off. The screen captures with wireframe are taken in order to display the difference in poly count between the models, while the ones without wireframe are used to display the aesthetic differences between the models using normal maps and modeled details. A selection of pictures are displayed and described below in order to give a better understanding of the results in the stats in Unreal Development Kit found below 4 Results. The rest of the pictures can be found in the Appendix.

3.5.1 The Roofs

The roofs used for our practical part of this essay are 4 different types of roofs that are used to create variety in the shape of the houses.

The roofs for the Scene 1 (to the right in the images) are created with a high subdivision level and with the use of supportive edges in a method often referred to as Korean bevel. The models are closed and even the bottom parts of the models have geometry even though it will never be seen by the player.

The roofs in Scene 2 (to the left in the images) are the same in appearance and shape but are using normal maps baked in Maya from high poly models that give the model the illusion of having smoothed edges. The geometry on bottom part of the model that is not to be seen by the player is deleted.

Most of the models in the Scenes are created this way, with some exceptions.
3.5.2 The Walls

The walls created for the scenes are different from i.e. the roofs.

For Scene 1 we create two parts of modular pieces that when tiled multiple times make up a facade or a wall. There is also a triangular top part. The wall meshes have to be used a lot of times in order to make up a whole wall in the Unreal Development Kit scene. The planks on the walls are modeled.

For Scene 2 we create a bigger amount of combined wall pieces. Most of themselves make up a whole wall. We decided upon combining the meshes in order to make it easier and less time consuming to place the walls in scene. The details that can be seen (the planks) are added through baked normal maps.

3.5.3 The Vegetation

The vegetation in the scenes is also different from the rest of the models. In order to make the vegetation complete it requires alpha textures. We chose not to implement such textures in this scene since our main focus lies on implementing details through poly count in contrast to implementing them through a normal texture. Since we do not have individual diffuse textures or alpha maps, we are not using normal maps for vegetation either.

In Scene 1 we use vegetation with a high poly count where most of the details are modeled. The vegetation also consists of separate meshes. In Scene 2 on the other hand the details, i.e. the leaves are supposed to be added using alpha textures, hence the low poly count. We also combine multiple meshes of i.e. flowers and bushes to make it less time consuming for a level designer to place it in a level.
4 Results

4.1 Stats

Stats Scene 1:

Image 4. Scene 1, stats in UDK, Play-in-Editor.

The amount of draw calls: 2163.02
Frame rate: 6.10 fps
Milliseconds per frame: 163.90 ms
Visible static mesh elements: 325.00
Memory: 17.361,71 Kbyte
Stats Scene 2:

Image 5. Scene 2, stats in UDK, Play-in-Editor.

The amount of draw calls: 1380.23
Frame rate: 10.21 fps
Milliseconds per frame: 97.90 ms
Visible static mesh elements: 181.87
Memory: 9.920,22 Kbyte

Comparison Stats:

Decrease in draw calls from Scene 1 to Scene 2: -36.2 %
Increase in fps from Scene 1 to Scene 2: +40.7 %
Decrease in ms from Scene 1 to Scene 2: -40.3 %
Increase in static mesh elements from Scene 1 to Scene 2: -44.1 %
Decrease in memory use from Scene 1 to Scene 2: -42.9 %

4.2 Aesthetics in Scene 1 and Scene 2

In order to discuss and compare the aesthetics of the scenes we took several screen captures in the editor of UDK. We will present one capture of each scene below to better visualize our results. The other screen captures and the wire frame versions of them can be found in the Appendix. When taking the images we also added a gray material to the ground.
Our goal was to have both scenes represent the concept art as well as possible without the use of texture maps. When working in the scenes we used our own aesthetic judgment to decide how far we would take the optimization.

Image 6. Scene 1, front.

The picture above is one taken from a closer perspective in the middle of Scene 1.

Image 7. Scene 2, front.

The picture above is one taken from a closer perspective in the middle of Scene 2.

4.3 Comparison of the Results

As mentioned in the method section the laptop on which we recorded the stats was not in the best of conditions and its graphics card and CPU were outdated. This reflects on the stats being lower than they would be on an up-to-date
computer. We also want to point out that we used the play-in-editor mode instead of the Play-on-Pc mode which also affected the performance level negatively. Due to time limitations we did not build in lights in the game scene, which affects the game performance to an extent as well, but the effect should be the same in all scenes so the results could still be measured and compared. Because of the same reasons we did not create a finished build of the game to conduct our research on, even though the results might have been more true to a game’s final performance as a consumer product.

There is a major difference in the results from the two scenes. Concerning draw calls we managed to decrease the amount of draw calls by 36.2 % from Scene 1 to Scene 2. As to the frame rate the difference was also essential. The frame rate in Scene 2 being 10.21 fps is much better than the frame rate in Scene 1 being 6.10 fps. In Scene 2 we also managed to decrease the amount of visible static mesh elements from 325.00 to 181.87. The memory use was decreased by 42.9 % from Scene 1 to Scene 2.

The results show that optimization of 3D game models makes a lot of difference performance wise when it comes to decreasing the amount of draw calls and increasing the frame rate for a game. Using normal maps instead of modelled detail does not seem to make a major difference to the performance and the GPU but we discovered that it does make a lot of difference memory wise, as the memory usage almost halved in our second scene. We saw this as a positive result for our research as both scenes were visually close to our original concept and within our aesthetical boundaries.

When deciding whether or not to add details to a model or work with additional texture maps, consider what works best for the game engine you are working in. Using normal maps instead of modeled details did not make a big change in UDK when it came to performance or GPU, but it made a significant difference for the memory. This may be due to the limit of poly count UDK imposes (see above 3.3 Scene 1). Consequently, it can be stated that it is preferable to use normal maps instead of modeled detail when possible and when time allows it. Deciding on making modeled details or using alpha texture maps in the material can make a huge difference, as the UDK engine has a harder time working with alpha textures.

4.4 The Importance of 3D Game Model Optimization in UDK

The purpose of this study was to contribute with knowledge about 3D model optimization when creating a game. By creating the scenes in UDK and profiling them we discovered that 3D model optimization and combining the meshes seems to make a huge difference performance wise in UDK. By optimizing our scene we managed to decrease the draw calls by 36.2 % and the milliseconds per frame by 40.3 %. We managed to decrease the amount of static mesh elements from 325.00 to 181.87 in the optimized scene, hence the decrease in draw calls. The frame rate also increased a lot in Scene 2, making it smoother to move around in the scene.

Our results indicates that optimizing 3D models for UDK is important as both draw calls and frame rate were decreased and the memory usage had sunk
almost half the size from Scene 1 to Scene 2. As a 3D artist for games you can help improving the performance by removing unnecessary geometry from the models and by using normal maps instead of modelled geometry as well as combining objects when possible. Avoiding unnecessary parts and triangles on a mesh, is an easy and fast method for improving game performance and is something graphical artists needs to keep in mind when working on a 3D model for a level. As it may not show a big difference on each mesh individually, it will have an important impact on a whole scene or a level in a game.

4.5 Optimization within the boundaries of aesthetics and mesh reusability

Optimizing meshes within boundaries of aesthetics is important as you do not want to lose the visual quality in a game. As shown in our research the visual difference is not that big, although the shading in Scene 1 appears smoother the overall appearance and feel of the scene is the same. This sort of optimization can be achieved by opting for baked normal maps instead of using korean bevels, deleting unnecessary faces in the mesh that will not be seen by the player and critical decision making on what details can be achieved through texturing.

Because we did not include texturing in our research we cannot show this in practice.

When optimizing a model by reducing polygons we could have reduced them to very primitive shapes, but as our goal was to achieve as close resemblance as possible to our concept, we had to keep the poly count at a certain level. We found that one can reduce the poly count a lot as long as the overall geometry and shape of the models are kept in mind.

Different shapes allow different levels of optimization. An object with a very primitive cube shape can be optimized more than an object with a curved silhouette since the latter needs more polygons to maintain its shape. Plain surfaces, such as walls and floors, can be made out of a few polygons while a tree trunk needs more polygons to maintain its curved and cylindrical form. With fewer polygons curved objects may lose their smooth silhouette and in some cases this cannot be redeemed through normal maps. We found that organic objects require more polygons to keep a more natural and realistic shape than manmade objects such as facades and porches.

In Scene 2 the facades, the stone wall, the porches, the windows and the benches were optimized to a higher degree than the more organic objects, such as the trees and the vegetation. Furthermore, in our concept the roofs had a very curved and organic silhouette which meant that we could optimize certain parts of the model more than other parts. The relatively flat sides could be implemented through normal maps instead of modeled details while the curved spine of the roof had to have a higher polygon level in order to maintain the original shape found in the concept.

By analyzing the optimized meshes in Scene 2 we formulate a set of guidelines that can be followed when optimizing 3D models for games without losing the shapes and the aesthetic values of the models. These guidelines can be found in 5 Conclusions.
The first thing we noticed when looking at our optimized models was the difference in the poly count between objects with a curved silhouette compared to objects with a sharp angled silhouette.

For example, the roofs in our scene (see image 8 and image 9 above) have organic and curved shapes and silhouettes and we had to keep this in mind when optimizing these objects. We could not remove too many polygons that made out the silhouette without losing its smooth and curved outline. The different kind of roofs in our scene also allowed different amount of optimization. I.e. Roof 1 did not allow a lot of optimization since both the silhouette and the shape required large organic geometry (see image 8) whereas Roof 2 had a curved silhouette but a plain shape (see image 9).

As an example for an object with a sharper silhouette, the door in our scene (see image 10) could be optimized by removing more of the polygons without losing the shape and outline. Other objects such as floors, porches and stone walls are made out of a more cubic form, where the silhouette has more sharp angles and corners. This allows more room for optimization through polygon reduction since the main shape can be achieved by using fewer polygons. As seen in our window models (see image 11), as soon as the shape and silhouette have more angles and softer curves the object becomes harder to optimize through polygon reduction while maintaining its preferable shape.
In Scene 2 we replaced supportive edges combined with softened edges with normal maps. We did this in order to give the objects more natural and realistic shapes, since the use of hard edges on models would have made the scene and the objects look unrealistic. Smooth edges can often be achieved by using normal maps, although in some cases normal maps cannot entirely replace geometry and give the mesh a smooth and natural look. One example of such objects is those made out of cylindrical geometry. In most cases, it is a bad idea to use an optimized quadrangular shape with a normal map baked from a cylindrical high poly model for a cylindrical object. In these cases it is better to maintain the cylindrical shape by keeping the polygons and use the normal map to keep the shading arbitrary. Good examples of this in our scene are the street lamps (see image 12). These are made out of cylindrical geometry that is difficult to replace with normal maps.

Our scenes were created for a game with a first person camera view, which means that the player will be able to get close to objects on ground and eye level. The bench in our scene (see image 13) stood on ground so it could be seen from many angles and close up and therefore the silhouette needed optimization without losing the shape. The details needed to be optimized while maintaining a high level of definition.

For their sizes the roofs could be optimized quite a lot in the detail level, since the player would never come close to them. If the player had been on the same level as the roofs we would have had to model some of the roof tiles to keep the detail level high enough in order not to break the game’s immersion.

Had our camera angle been further away from the scene we could have, for example, optimized the bench differently. We could have reduced the planks to one single plane and produced the gaps using alpha texturing. The doors could have been optimized by reducing them to planes and adding details through normal mapping.
Most of our vegetation (see image 14 and 15) was on ground level, so they needed to be somewhat highly detailed. Most of the details for vegetation come through texture maps, but the overall silhouette needs to be thought of when optimizing. Since the bushes, trees and flowers could be seen from many angles they needed more planes for the leaves than if the camera would have been further away or static. Far away vegetation could be optimized to only a couple of planes at the most since the silhouette could only be seen from one point of view.

In Scene 1 the stone wall (see image 16) was made out of individual stone pieces that we optimized for Scene 2 by reducing the wall to larger planes and adding the stone details through a normal map. This reduced the polygon count while maintaining the silhouette and details from Scene 1. The same thing was done to the facades (see image 17). In Scene 1 the planks were modeled into the mesh whereas in Scene 2 they were added through normal maps. In both cases the modeled versions for Scene 1 consisted of too many polygons for game models.

The same principle goes for most objects in a game. Depending on size of the object and the size and depth of the details on it you can model some of the details and add other through normal maps. Some of the irregularities and details of an object does not necessarily have to be implemented through normal maps or geometry; sometimes a well crafted diffuse texture map will suffice.

Vegetation was also a difficult subject when it came to aesthetic viewing as most greenery in games, as we know it, is created by alpha texture maps added on to the 3D models, creating the shape of leaves rather than modeling. We used our judgment to optimize the trees and vegetation, bringing the density of the planes, which represent where the leaves will be, to that level where we felt it would still look good in a game.

4.6 Self-criticism

Our research did lack in certain aspects that we would like to discuss. The absence of textures and alpha maps in our scenes does not provide a holistic view on how a scene can be optimized most efficiently and how different factors of game graphics optimization affect the whole performance. This is something that would have taken a lot more time than we had at our disposal.

We used our own judgment and view on aesthetic on how far we could take the optimization. This does make the limit unclear and hard to follow for a reader. A
suggestion how this could have been avoided was to test the optimized scene on test persons who could provide a more holistic result on where the limit for optimization in a scene goes while still maintaining the aesthetic profile we wanted to keep.

Our scenery could have also designed for this particular research. The use of a preexisting scene limited what parts of optimization on meshes we could work within our study.
5 Conclusions

In conclusion, we found in our research that 3D models play a big part in the game performance and thus optimizing these should be considered very important. By deleting unnecessary geometry and using normal maps, we managed to lower the draw call by 36.2% and speed up the frame rate by 40.7%, which resulted in a smoother running game without disrupting the aesthetic side of the game. Even the memory load was lessened by 42.9% when optimization was implemented.

In order to be able to investigate our first research question, the scenes were optimized while trying to keep the models aesthetically appealing and reusable in a level design perspective. We found that one could optimize the models a lot without losing the aesthetical appearance by using normal maps instead of modeled detail. However, the limit of how much we optimized the models was based on our own subjective views on what is visually appealing. The setting of the limit was based on keeping the shape of the models as well as the most vital details of the models, based on what we had read in previous literature on the subject. Our results show that optimization of models can go very far without losing the aesthetical values as long as the shape is not affected when reducing the poly count and as long as good normal maps are created in order to compensate for the removed details and the smooth edges of high poly models.

We found that when creating optimized models the level of optimization that is required to maintain the visual style is decided mainly by the silhouette and shape of the model. Manmade objects with relatively flat surfaces can be optimized further than organic objects with curves and unsymmetrical shapes. Some objects have a variation of both aspects when it comes to their shapes and silhouettes and therefore certain parts of the models allow different levels of optimization.

After analyzing the optimized meshes in Scene 2 we formulated a set of guidelines that can be followed when optimizing 3D models for games without losing the shapes and the aesthetic values of the models. The guidelines are described below:

1. Depending on whether the object has a curved and more organic silhouette or a sharp angled and straight silhouette the poly count optimization will differ. Objects with a more organic silhouette, such as a sphere, requires more polygons in order to maintain a rounded shape whereas a sharp angled and straight silhouette such as a box requires fewer polygons to maintain its shape.

2. When optimizing each object it is important to keep in mind what part of the object will be seen and from which camera angle. As a rule, objects far away from the camera needs less attention to details and more on the silhouette, whereas objects close to the eye needs more attention on details as well as silhouette. Optimizing like this is very good, but it is important to remember the mesh reusability and not to over optimize objects. That way they can be used in other scenes and angles as well.
3. Some details can be achieved more efficiently through normal maps than through modeled detail. This method works best on planes and large surfaces by adding smaller details that would have otherwise been modeled into the mesh. As long as the details does not need to affect the silhouette they can be added through normal maps on more organic and curved objects as well.

4. Normal maps can be used to achieve a similar effect as supportive edges combined with softened edges. No objects in real life have entirely hard edges and therefore it is important to try to give the edges a varying softened look depending on the object. Using supportive edges leads to a high amount of edge loops that are required to keep the shading arbitrary, in contrast to using normal maps where supportive edges are not needed. Using solely softened edges without supportive edges works in some cases, but the shading will not be as consistent as with normal maps. Therefore, if time allows it and if the object is to be seen from a close point of view it is a good idea to bake down the normals from a high poly object in order to keep a low poly count and good shading.

To answer our second research question, we found that using normal maps instead of modeled detail in our second scene did not make a big difference when it came to performance or GPU. On the other hand normal maps did lessen the memory usage in the scene. Whether or not it is worth to work with details on a 3D model does depend a bit on what sort of model it is and what it will be used for. It is preferable to use normal maps when possible and when time allows it. It is also important to remember what works the best for the engine you are working in, as some engines have harder to work with textures and materials, rather than a high poly count.

One important part to remember is what parts of the mesh is visible and how much of it will be seen in the game. Just by decreasing the polygon amount this way on every mesh, one can help to create a more balanced and smoothly running game. Instead of working with a huge amount of details in polygons, a well done normal map is preferable. A clever use of normal maps, LODs and reusable models will result in a better balanced and smoothly running game without the loss of an appealing aesthetic appearance.

Our research was pinpointed at the usage and importance of poly count and mesh optimization. This left a big part out of the game graphics which directly affects the game performance unexplored, such as textures and visual effects. It would have been ideal to compare optimization of game models with textures and materials and see how these affect the game performance and which weighs more. Another interesting future research subject would be to see how much could be optimized and removed when it comes to polygon models while keeping the game aesthetic and visually pleasing for the audience by the use of textures and materials instead.
References


Quixel, 2011. *nDo2 (1.1.2)*. [computer program] Quixel.


Appendixes

Images taken in Maya’s viewport:

A2. Image 2. Bench.
A5. Image 5. Entrance roof.


Scene 1 images taken in the editor in UDK:


A20. Image 20. Scene 1 editor, wireframe, overview.

A22. Image 22. Scene 1 editor, wireframe, right.


Scene 2 images taken in the editor in UDK

A27. Image 27. Scene 2 editor, overview.

A29. Image 29. Scene 2 editor, right.

A30. Image 30. Scene 2 editor, wireframe, right.
A31. Image 31. Scene 2 editor, left.

A32. Image 32. Scene 2 editor, wireframe, left.
A33. Image 33. Scene 2 editor, front.

A34. Image 34. Scene 2 editor, wireframe, front.
Scene 1 Play-in-Editor stats:

A35. Image 35. Scene 1 Play-in-Editor.

Scene 2 Play-in-Editor stats:

A37. Image 37. Scene 2 Play-in-Editor.

A38. Image 38. Scene 2 Play-in-Editor, stats.
Concept art: