Ville Ruti

OPTIMIZATION OF 3D MODELS FOR MOBILE GAMES

Balancing quality and performance

Bachelor's thesis

Bachelor of Culture and Arts

Game Design

2024

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Culture and Arts
Author(s)	Ville Ruti
Thesis title	Optimization of 3D Models for Mobile Games: Balancing Quality and Performance
Year	2024
Pages	40 pages, 5 pages of appendices
Supervisor(s)	Panu Vuoristo

ABSTRACT

Mobile gaming has reached a new level of popularity in recent years with an increasing number of people playing games on their smartphones daily. While smartphones have advanced significantly in terms of technology with high-end devices being able to handle visually stunning games, the majority of users still own mid-range smartphones. Hence, the need for optimization is still relevant to game developers to ensure performance on older devices and help their games reach a wider audience.

This bachelor's thesis focuses on researching and discussing the optimization of 3D assets for mobile platforms, striking a delicate balance between visual quality and performance.

To conduct this study, an extensive literature review was done to explore 3D asset optimization theory, and an interview with an industry professional was conducted to gain perspectives on best practices adopted in the mobile gaming industry. Furthermore, a hands-on approach was taken by creating assets for a mobile game project and optimizing them using the knowledge gained from this research. This involved applying the revealed optimization methods and assessing their effectiveness in achieving the desired balance between quality and performance.

In conclusion, the study uncovered strategies for improving the performance of 3D models designed for mobile game environments providing practical guidance for developers.

Keywords: 3d, modeling, optimization, mobile games

CONTENTS

1 INTRODUCTION

In the realm of gaming, where adventures unfold with a swipe of a finger and come to life on the compact screen of a smartphone, the immense effort behind the scenes often goes unnoticed. While captivating worlds are created, a crucial challenge arises – how to balance breathtaking visuals with the constraints of performance. This thesis attempts to uncover the art of striking a balance between brilliance and flawless functionality aiming to expand the author's knowledge of 3D modeling and game art optimization in mobile game development.

Qualitative research methods serve as the foundation for this study. A thorough review of existing literature, including articles, books, and online resources provides the foundation, and an interview with an industry expert contribute a perspective on practices in mobile game development. The combination of these will result in a better understanding of the methods and tools available for optimization as the research advances to the practical part of this thesis project where the theory is put into practice.

A significant component of this research involves hands-on application as the objective is to create optimized 3D models specifically designed to be used in a mobile game project, exploring various optimization techniques and comparing the results. The models and their textures will be produced with combinations of software including Maya, Zbrush, and Substance Painter. They will then be imported into Unity and tested using Unity’s native performance profiling tools.

As a result, this thesis will offer an exploration of optimizing 3D game assets tailored for those seeking insights into different optimization methods, within the realm of mobile game development.

2 3D MODELING

2.1 Definition

3D modeling is the process that uses specialized computer software to create a representation of a physical object or environment in a simulated 3D space using vertices that are connected forming edges and faces. An edge refers to a straight line that connects two vertices in a mesh structure while a face is formed when a minimum of three vertices are connected in a closed loop of edges. The most common types of faces are triangles and quads. (Slick 2020.)

A 3D modeling pipeline, on the other hand, refers to a process or a series of steps used to create a 3D model. This pipeline usually includes stages starting from the idea and concept to the ultimate visualization of a 3D object or scene. (Collins 2018.)

2.2 Stages Involved

According to Irsayd (2023), the creation of 3D models represents only the initial stage of 3D asset workflow, comprising roughly 40% of the process. The majority of the effort revolves around ensuring the assets can be seamlessly integrated into the game and delivered to the players while keeping them usable, appealing, and fun. The created assets need to hit technical specifications so that they do not cause problems to the game’s performance. While adhering to these specifications, the assets still need to look appealing to the player and meet the visual standards of art direction. Lastly, the assets should be fun to the player, but it can be a subjective aspect as it is ultimately up to the player to make that judgment. (Irsayd 2023.)

The stages involved in the 3D asset pipeline can vary from studio to studio, depending on the software tools used and the complexity of the project. An interview with an industry professional (Appendix 1), conducted as a part of this research, revealed that the typical 3D modeling pipeline generally includes the stages seen in Figure 1.

Figure 1. Stages involved in typical 3D asset pipeline

The stages color-coded in yellow (1-6) are typically handled by the art team or an artist and stages 7 to 9 often fall in the hands of the non-artists, such as programmers or game engine experts. Larger projects and companies may have sizable teams and a dedicated person for each stage of the pipeline whereas in smaller companies with compact team sizes, one team member tends to work on multiple stages. For instance, a 3D artist in a smaller team may handle most, if not all the stages in the pipeline. While some linearity can be found in the modeling pipeline, the 3D asset creation is rarely a linear process but is rather bounced back and forth among the stages involved (Appendix 1).

3 MOBILE GAMES

A mobile game is a game that is created and optimized for portable devices, such as smartphones or tablets. It can also refer to games played on any mobile device, such as portable game consoles. However, in regular conversations, the term typically refers to games played on smartphones. In recent years, mobile games have surpassed PC/Console games in popularity and have continued as the most popular gaming device among gamers (Google 2023), despite the growth cooling down in 2023 in terms of downloads (Data.ai 2024).

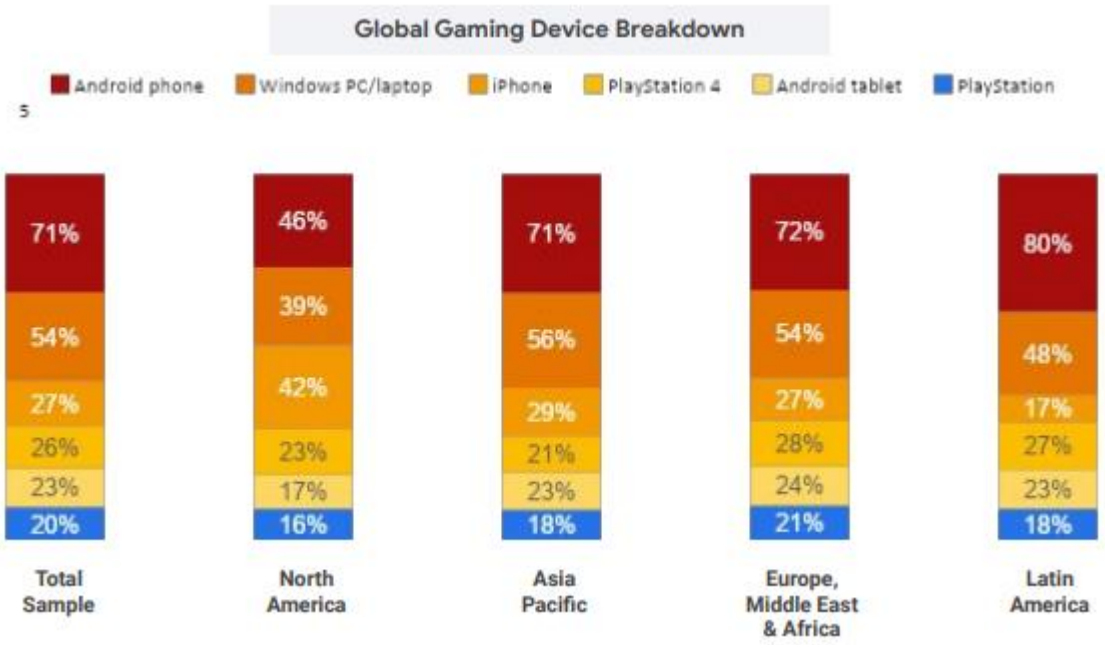

Figure 2. Global gaming device breakdown (Google 2023)

Gaming market research conducted by VGM reveals that gamers most often play games on Android phones (71%) and/or Windows PCs/laptops (54%). Among respondents who play games on mobile and PC/console, players most often say mobile is their main gaming device. (Google 2023.) The report clarifies the scale of mobile games' popularity in today's gaming landscape.

3.1 Characteristics and Constraints

One of the main defining characteristics of mobile games is portability. They are designed to be consumed as on-the-go entertainment, allowing users to play at the time and place of their preference. To cater to the mobile gamer's tendency for a convenient gaming experience, many mobile games are structured for short play sessions. According to Scolastici & Nolte (2013), the player should have a satisfying gaming experience in three minutes or less. This is supported by the data from GameAnalytics (2021) which points out, that the median session length for mobile games is only 4 to 5 minutes.

The differences between mobile platforms and PC / Console are significant, ranging from hardware capabilities, control schemes, business models, and

pricing policies. Where PC / Console games are designed to be played with a keyboard and mouse or a controller, mobile games are commonly played by interacting with a touchscreen interface of smartphones, utilizing gestures, taps, swipes, and other touch-based inputs. Due to the limitations of a touchscreen, mobile games often feature intuitive controls to ensure accessibility for a broad audience.

The smaller screen of a mobile device can also be a constraint for certain types of games, affecting the gaming experience and user interface (UI) design. The small screen, along with mobile device hardware limitations, imposes constraints on the complexity of the game and limits developers' ambitions to fabricate highly detailed visual assets. These limitations or constraints can ultimately be seen as positive, encouraging developers to optimize for efficient battery consumption.

3.2 Importance of Optimization

User experience (UX) in mobile gaming is pivotal in determining a game's success. It shapes how players perceive and interact with the game, affecting their emotions, behaviors, and overall enjoyment. Lengthy loading times, lack of performance, and unresponsive controls may have a negative impact on user experience and player retention rate.

Despite the advancements in high-end mobile device technology, most users still have average devices with limited capabilities. Given the popularity of the mobile market and its diverse consumer base, it is good practice to target the broadest audience possible. While a game may function flawlessly on a top-tier device it may encounter issues on lower-end smartphones. These factors will therefore dictate stylistic and technological choices the developer can make. By ensuring the game's smooth performance on a wide range of devices, game developers can influence the success of their game and how it is received by the players. This underscores the significance of optimization. (Scolastici & Nolte 2013.)

4 OPTIMIZATION TECHNIQUES

One of the main challenges of 3D modeling for mobile games is to optimize the models and make the game run smoothly on different devices. Mobile devices have limited processing power, memory, and battery life; thus, game developers need to ensure the 3D models are not too complex or heavy. In this context "*heavy*" means they have very detailed shapes, containing an excess number of polygons for the processing power of mobile devices to handle, or come with large texture maps that require an overflowing amount of draw calls (Gosch, 2016). Draw calls are lookups for assets, which happen every frame, and the number of draw calls for an application depends on unique meshes and unique materials in the scene. According to Unreal Engine Documentation (2023), a high number of draw calls is the largest contributor to low graphical performance. To create optimized 3D assets for mobile games, it is necessary to understand the optimization techniques and how and when to utilize them, for instance, how to reduce the polygon count, use textures and materials wisely, and how to avoid unnecessary details. It is also crucial to know how to test the models on different devices and platforms, and how to fix any errors or glitches.

4.1 Real-Time Rendering

One of the initial factors for consideration is the rendering. It is vital to decide whether it is worthwhile to create a real-time rendered 3D game for mobile platforms, or if a pre-rendered approach would be a more suitable solution. Pre-rendering, which is also known as offline rendering, is defined as a type of rendering where the rendered image or sequence of images (animation) is displayed later. Whereas in real-time rendering, as the name implies, the rendering calculations are done in a much faster timeframe. (Unity n.d.a.)

The main benefit of pre-rendering is the ability to incorporate complex graphic models that demand substantial computing power into your game. This is because pre-rendered 2D animation sequences or sprites do not require as much processing power as rendering 3D objects in real time. However, creating a

lighting setup for a 3D scene containing dynamic 2D characters, for instance, can be challenging as 2D images do not react to lights the same way 3D models do.

Garbett (2023) discusses that lighting is the most resource-hungry operation found in 3D games. Although, the lighting information can be baked into a scene to help streamline the rendering process. The process in question prepares the lighting in advance, saves the result to disk as lighting data, and loads the data at runtime, alleviating the need for players to wait for real-time rendering calculations. Baking lighting information into the scene is recommended in Unity Documentation (2023) for static objects that do not change at runtime, such as scenery, to reduce the rendering costs. It must be acknowledged that dynamic shadows cannot be created with baked light. This might look strange with dynamic or moving objects.

4.2 Image-Based Lighting

Several lighting and optimization techniques take advantage of image data, typically in the form of 2D texture maps. These are called image-based lighting algorithms. Examples of these types of texture maps include a normal map, height map, displacement map, and specular/roughness map. (Gregory 2018.)

To avoid large quantities of polygons in a 3D model, which would require a lot of computing power, the detail information from a high-poly model can be baked into a texture map. The texture map can then be used to provide the rendering engine with a highly detailed description of a surface’s shape. By using a normal map for instance, a simple flat surface can be made to look like it was constructed from millions of polygons, as demonstrated in Figure 3. (Gregory 2018.)

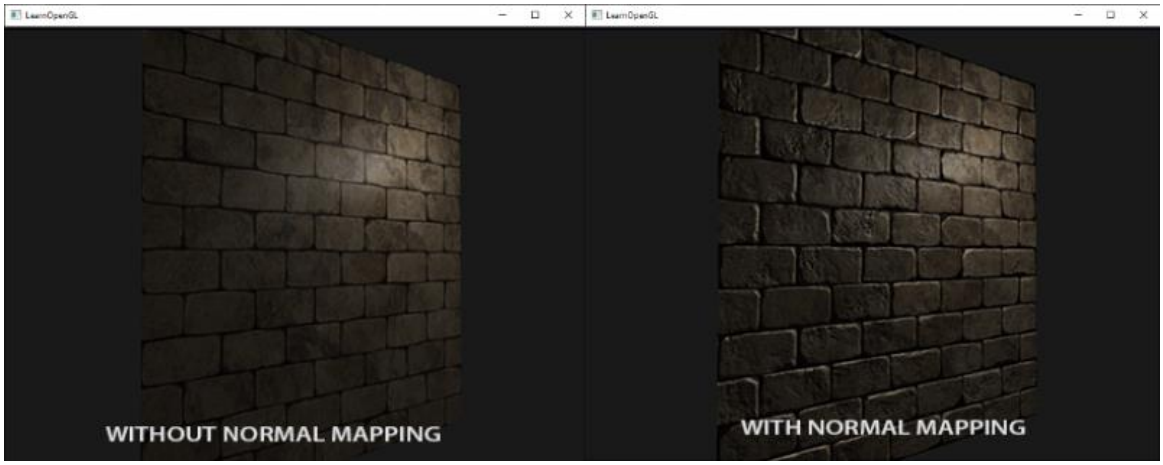

Figure 3. Surface with and without a normal map (LearnOpenGL 2017)

Normal maps can also be created without modeling in 3D. They can be converted from photo textures, created with node-based graphs, or even hand-painted. Nevertheless, a normal map baked from a high-poly 3D mesh will often yield better results than one sampled from a texture, given that information is being rendered from a highly detailed surface. (Polycount 2018.)

4.3 Polygon Reduction

Retopology and polygon reduction are important parts of the optimization process, as a considerable number of polygons correlates with a high demand for processing power. Polygon reduction can be done manually or automatically using mesh simplification algorithms. (Edesberg, 2023)

Figure 4. Example of polygon reduction: from 1.07 million triangles to 14 151 (Lesterbanks 2013)

For most static models the automatic algorithms perform the task adequately. Characters and other assets that will be animated typically need to undergo a semi-automated or manual retopologization to preserve the optimal topology and edge flow. This is essential to ensure the mesh does not break when deformed. Semi-automated algorithms let the artist create guides to have control over the result, while the manual method gives the modeler total control over the topology but takes the most amount of time. (Gosch 2016.)

4.4 Level of Detail (LOD)

Detail becomes increasingly important the closer the player gets to an object in a video game, but there is no need for a great amount of fine detail when the camera is distant from the same game object. The developer can set game engine to automatically switch to a simpler mesh, that contains lower number of triangles, as its distance from the camera increases. This optimization technique is called *Level of Detail*, or *LOD*. (Garbett 2023.)

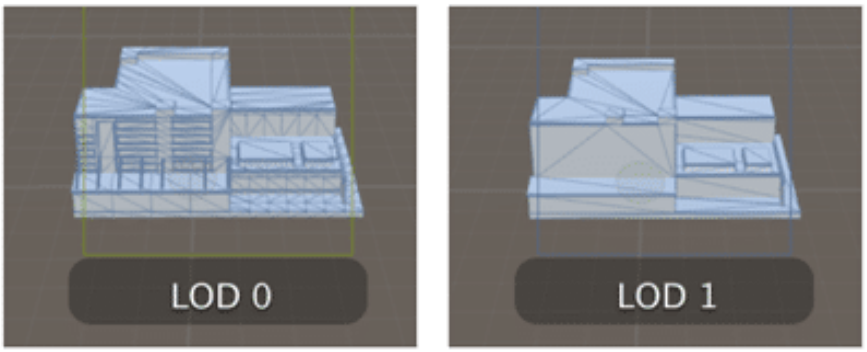

Figure 5. Camera at LOD 0 and LOD 1 level (Unity 2022)

The way LODs work is that the developers first create meshes with different levels of detail along with LOD groups for the game objects and decide whether to favor higher or lower LOD levels at threshold distances. Level of Detail then can adjust or switch objects as they move into the distance to use simpler meshes, along with simpler materials and shaders to enhance GPU performance. A simpler mesh in LODs can get as low as a single sprite. (Unity documentation 2023.)

4.5 Textures, Atlases, and Compression

Many 3D games use several texture maps for their 3D models, similar to 2D games having many small art assets used in the game individually. According to Roy (2016), this may cause lag for the game, and it is recommended to merge assets into one sheet. Such a texture sheet is called a “Texture Atlas”. Texture atlases can save memory by limiting the number of draw calls, earning a boost in performance. Software examples for this task include the commercial TexturePacker, the open-source packer of libGDX, or the Sprite Packer included with Unity3d. (Gosch 2016.)

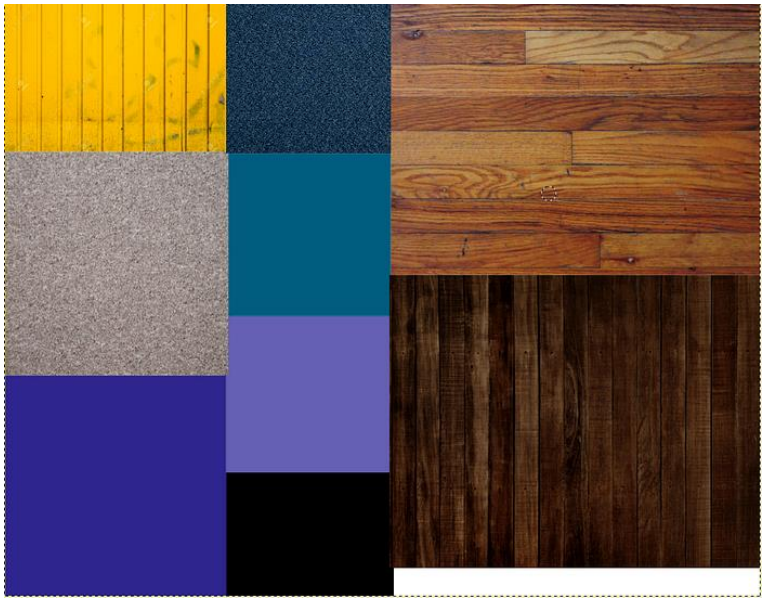

Figure 6. Example of a texture atlas (Bernardi 2018)

Textures can be compressed, and most game engines come with a built-in compression system. It is recommended by Unity to compress the textures using Adaptive Scalable Texture Compression (ASTC) for both iOS and Android as the compressed textures require less memory. The vast majority of games in development are targeted to devices that support ASTC compression, with the only exceptions being Android devices prior to 2016 and iOS devices older than iPhone 5. (Unity n.d.b.)

Using texture mipmaps can provide a rendering performance boost in a game with 3D models at varying distances from the camera. Texture mipmaps can be conceptualized as Level of Detail (LODs) for textures where a mip or mip level is a version of a texture with a specific resolution. A lower mip level is used for distant objects and a higher mip for 3D objects closer to the camera. There are also times when mipmaps are not beneficial and can increase the size of a texture by 33%, on disk and in memory. Mipmaps can be created manually or automatically. For instance, Unity can generate the maps automatically when instructed to do so. (Unity documentation 2023.)

Unity also has a built-in system called Mipmap Streaming. When enabled, it forces Unity to load only the needed mipmap levels to render the current camera position, instead of loading all of them by default. This system trades a small

amount of CPU resources to potentially save a large amount of GPU memory. (Unity Documentation 2023.)

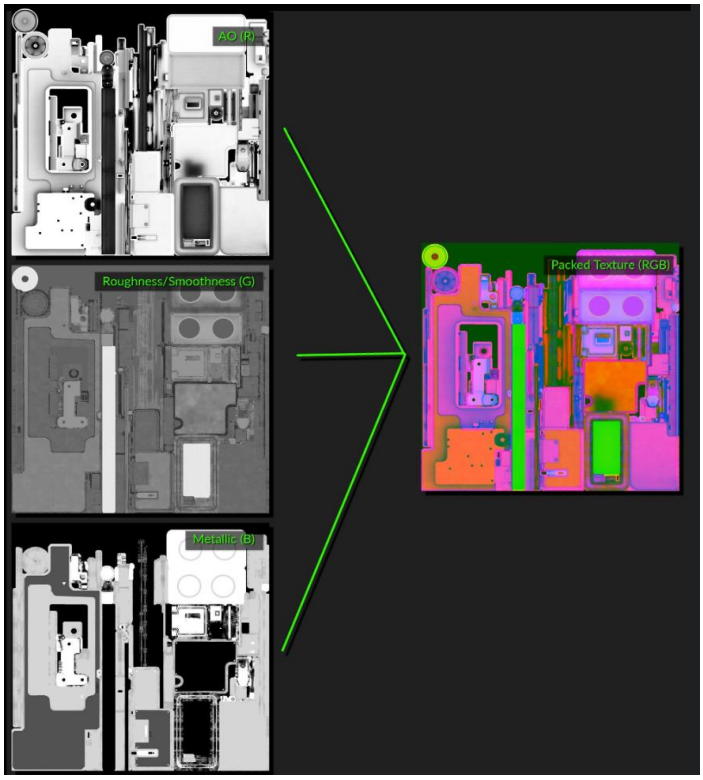

Figure 7. Example of channel packing

As revealed by the interview conducted, textures can also be combined to save memory and draw calls. This method is known as channel packing. For instance, instead of having separate textures for diffuse color, specular color, and normal maps, all this information can be packed into a single texture by storing each type of data in different channels (RGBA) of the texture (Appendix 1).

4.6 Reducing Mesh and Material ID Count

To ensure smooth performance and to save processing power mesh count can be reduced by combining as many objects as possible into one mesh. This can either be done manually in a 3D modeling software or a game engine. In Unity, a function called *Mesh.CombineMeshes* can be used to combine the desired meshes into one. Unity then renders the combined mesh in a single draw call instead of one draw call per mesh. Combining meshes is very useful for performance optimization. (Unity Documentation 2023.)

Reducing the number of unique materials can also decrease the number of draw calls. The simplest way to reduce the number of unique materials on a mesh is to use a program like Substance Painter to integrate multiple materials into the same texture, instead of creating a separate material for every material type in a game engine. (Unreal Engine Documentation 2023.)

5 QUALITY VS. PERFORMANCE

As mobile games become more popular each year, the expectations for premium graphics grow with their popularity. However, a visually stunning game can fail if it does not line up with the processing capabilities of mobile devices. Balancing between stunning aesthetics and smooth gameplay is a difficult task every mobile game developer face with each of their projects. As mobile devices and their hardware capabilities vary widely, finding the right balance for the broadest possible audience becomes even more challenging.

5.1 Trade-off Analysis

Prioritizing quality graphics can enhance the overall gaming experience and help with the immersion. Captivating visuals, vibrant textures, and quality animations can also make the game stand out in the competitive marketplace and attract more players. First impressions matter and for mobile games that typically is the graphics (Smith 2014). However, high-quality graphics can be performance-heavy, cause lagging, and other problems such as rapidly draining battery or overheating mobile devices.

5.2 Finding the Optimal Visual Appeal

Now that the optimization of 3D assets is understood, the task ahead is finding a balance between performance and quality, achieving optimal visual appeal.

Answering this complex question necessitates defining the target audience and comprehending the mobile phone market and the devices used for mobile gaming. Figure 8 illustrates the most sold mobile phones in the first half of 2023 and for comparison, the same statistics of the first half of 2022.

Global Top 10 Most Shipped Smartphones

Ranking	Model Name	Brand	1H23	Model Name	Brand	1H22
1	iPhone 14 Pro Max	Apple	26.5	iPhone 13	Apple	33.7
2	iPhone 14 Pro	Apple	21.0	iPhone 13 Pro Max	Apple	23.0
3	iPhone 14	Apple	16.5	Galaxy A13	Samsung	16.2
4	iPhone 13	Apple	15.5	iPhone 13 Pro	Apple	14.8
5	Galaxy A14	Samsung	12.4	iPhone 11	Apple	11.7
6	Galaxy S23 Ultra	Samsung	9.6	Galaxy S22 Ultra 5G	Samsung	9.8
7	Galaxy A14 5G	Samsung	9.0	Galaxy A12	Samsung	9.5
8	Galaxy A54 5G	Samsung	8.8	Galaxy A03 Core	Samsung	7.7
9	Galaxy A34 5G	Samsung	7.1	Redmi Note 11	Xiaomi	7.6
10	iPhone 11	Apple	6.9	Redmi 9A	Xiaomi	7.4

Source: Omdia © 2023 Omdia

Figure 8. Globally most shipped smartphones in the first half of 2023 and 2022 (Omdia 2023)

Every smartphone on the list displayed in Figure 8 has great processing abilities and is capable of running demanding rendering calculations. However, there are still many smartphones in use that have been purchased years ago. According to Kim Komado (Usatoday 2023), the average lifespan of smartphones is approximately 2.5 years but varies among brands with Apple's iPhone having the longest average lifespan of 4-8 years. Based on this data, it can be argued that it might not be a top priority to optimize for devices older than 8 years. This is a consideration to remember when identifying a mobile game's target audience.

After the target audience has been determined, the only way to understand how the game runs on their smartphones is to test it, ideally with similar devices and in an identical environment. The tests can be done with or without profiling tools, but without tools one must rely on what they see on screen, such as visual stuttering or noticeable FPS drops. Profiling tools are essential for pinpointing where the GPU's time is spent. More sophisticated tools even display a list of bottlenecks, which can be enormously useful for the developer. (O'Conor 2017.)

5.3 Performance Metrics and Profiling Tools

Mobile games, like any other game, require performance testing to identify possible bottlenecks and other problem areas of the game. Identifying where the performance problem is should be the starting point for any optimization (Unity Documentation 2023). The required test type varies depending on the game's nature, graphics, gameplay, and many other things. Typically, mobile game performance tests are performed to determine how the game performs regarding responsiveness, refresh rate, and stability under certain conditions. The typical metrics that are examined are CPU load, loading time, memory consumption, draw calls, and FPS. Based on the data gathered from these tests, game developers can further improve the game by scaling down and optimizing heavy graphical assets. (Helppi 2014.) These topics will be covered in greater detail later in the practical part of the thesis where performance profiling is taken into further inspection.

There are many profiling tools available for performance testing. Some of the popular and widely used profiling tools include Unity Profiler, Unreal Engine Profiler, Intel Graphics Performance Analyzers, Android Studio Profiler, and Xcode Instruments. They offer different features, such as deep profiling, frame debugger, memory snapshot, event graphs, stat commands, and can monitor and analyze CPU, GPU, power, and system performance. In most cases, game developers have the tendency to use the profiler that comes with their game engine of choice. (Modjadji n.d.)

6 IN PRACTICE

The practical part of this thesis consists of modeling 3D assets for a mobile game titled: *Captain Blacktail's Purrfect Plunder*. The task was to model in-game items and environmental assets as well as characters, and texture them while keeping them optimized for mobile platforms. Since the purpose of this thesis is to act as

a guide for mobile game 3D asset creation and optimization, the whole process is documented and annotated beginning from the idea generation and reference gathering.

6.1 Background of the Project

The story of *Captain Blacktail's Purrfect Plunder* centers around a pirate cat who reclaims a long-lost treasure from menacing sea creatures, much to the monsters' displeasure. The player takes the role of Captain Blacktail, tasked with safeguarding the ship and the reclaimed treasure from attacks of displeased sea creatures. The game's setting unfolds in a maritime environment and the theme is along the lines of pirate cats and sea monsters.

The game can be characterized as a tower defense game with a twist, where merging items (merge two) plays a major role in the game's mechanics. Separate research was conducted when the art style of the game was decided, and the team agreed on stylized art that is popular amongst mobile games and the target audience of the game.

6.2 Idea Generation and Reference Gathering

First, a list of graphical assets required for the game was written down. For the purpose of this thesis, only two of the list's assets were selected to be featured in the research: one character and one in-game item. A piranha character was chosen to be created and optimized for the game and for the in-game item, a barrel was selected.

The concept development of the piranha character started with examining the already existing characters of the game and by gathering images of real-life piranhas as well as images of other artists' drawings and 3D renders into a reference board. The reference board also contained color, mood, and style references to guide towards the desired look. A video by FlippedNormals (2020)

describes a reference board as a collection of images that help the artist visualize and fill the gaps in knowledge, making the piece feel grounded to reality, regardless of the art style. It can include pictures of anatomy, poses, textures, and other parts of the asset (FlippedNormals 2020).

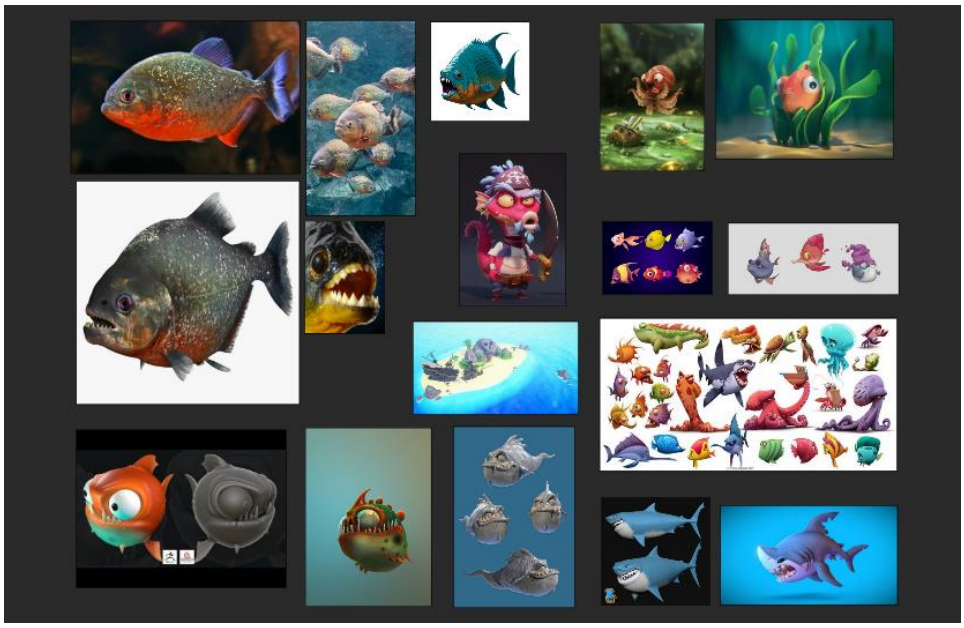

Figure 9. Piranha reference board

PureRef is a software made for this purpose; it was used to gather images of piranhas and barrels into their respective reference boards. While the piranha required a more thorough gathering of images, only a handful of pictures were selected for the barrel's reference board, focusing on the art style references and texture reference images. The reference board for the piranha character can be seen in Figure 9.

6.3 Modeling the Assets

The two assets are very different from each other resulting in a different approach to the modeling. While the modeling of each asset began from a block-out, the workflow for the piranha model was backward compared to the barrel. A high-poly model was initially created for the piranha, from which the low-poly model was derived. In the barrel's case, however, the low-poly model was crafted first.

6.3.1 Modeling the Piranha

The modeling process started with Zbrush, which is a software made specifically for digital sculpting. Zbrush was selected to be the starting point of the modeling process because of the number of organic shapes the asset being modeled contained. Organic shapes are typically easier and faster to create by sculpting than by polygonal modeling in a 3D modeling software like Maya or Blender, which was ultimately the reason for the choice of software and workflow.

A blockout, which can also be referred to foundation of a 3D project, is where the modeling process begins. It is the stage where primary forms are laid out on which the secondary and tertiary forms are built later. Blockout is often considered the most important part of the process, which essentially means if the foundation is not strong enough, no amount of detail or texture will later fix it. (Follygon 2020.)

Large defining shapes were implemented first. Using a sphere as a starting point, the basic shape of the fish's body was formed, and more shapes could be added. By adding more primitive shapes as new sub-tools or separate meshes, the character got eyes, fins, and flippers. Also, the jaw was added as a separate mesh, so that it could be easily edited until the shape and silhouette were pleasing to the eye. As the model was in its blockout stage, the number of polygons was irrelevant from an optimization perspective. Nevertheless, the polycount was deliberately kept low to prioritize focus on large shapes and maintain smooth surfaces. As the secondary and tertiary forms were introduced, the mesh was subdivided accordingly, and polygons were added. The basic

brushes of Zbrush were used to quickly shape the mesh, such as the move tool, standard brush, clay buildup, damStandard, and IMM Primitives.

Figure 10. Early blockout of the Piranha character

After completing the blockout, the mesh was moved to Maya for manual retopology. There were two main reasons to retopologize the mesh: to reduce the polycount and to ensure the shape would deform without breaking. The former is an optimization method introduced in Chapter 4.3 of this thesis. The objective was to minimize the polycount to ensure smooth rendering on mobile devices, even if multiple instances were present on the screen simultaneously. This needed to be achieved without compromising the silhouette and the defining shapes. The second reason, and the reason for manual retopology instead of automatic, was to guarantee proper topology and edge flow, particularly because it was anticipated that the character would be animated later.

The polycount was successfully reduced from 16,384 triangles to 1,464 triangles which accounts for over 90% polygon reduction. After the polycount was reduced to optimal levels, the model was ready for UV maps and textures. UV maps were created manually in Maya and the model was then exported back to Zbrush for sculpting of finer details, despite the model was going to be relatively small when displayed on a screen of a mobile device.

To sculpt detail into the mesh, it needed to be given more polygons to work with. To achieve the desired polycount, the model was subdivided several times, but as can be seen in Figure 11, there are sections in the model where polygons are not distributed evenly. The uneven and stretched topology makes sections, such as the tail, hard to sculpt details on. To fix this, a function known as DynaMesh can be used. It maintains a consistent polygon distribution and resolution across the entire mesh making it an ideal tool to be used in the beginning and middle stages of the sculpting (Zbrush n.d.). Regardless of the tool's original intended use, artists had found creative ways to utilize it in later stages as well, such as in this case. With the help of DynaMesh, the desired mesh resolution was achieved, and the rest of the details were sculpted into the mesh.

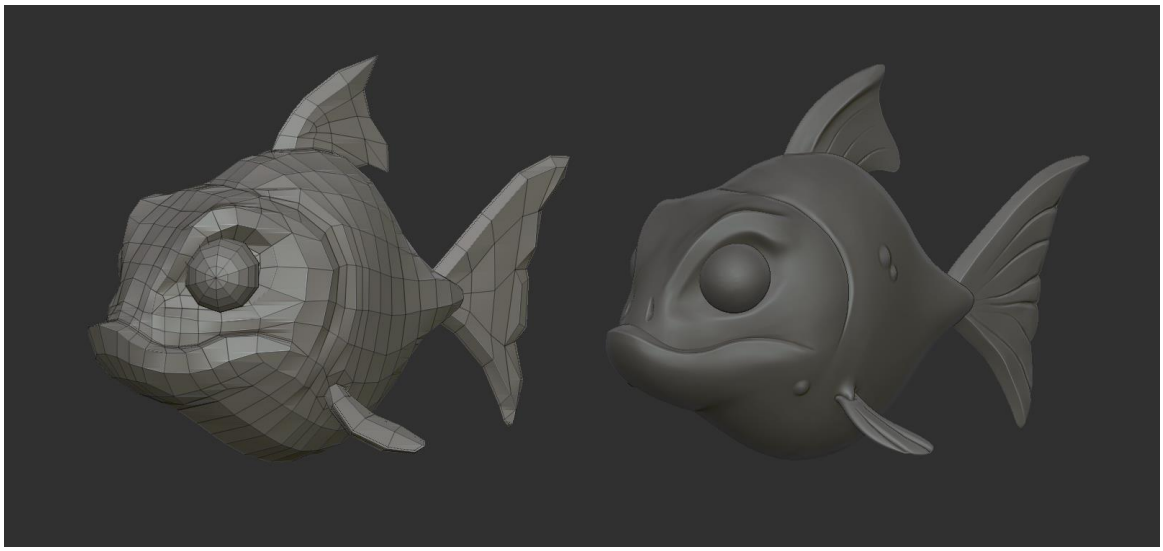

Figure 11. Low-poly mesh, 920 vertex points (left). High-poly mesh, 978 841 points (right)

As a result of subdividing and dynameshing, the mesh now consisted of hundreds of thousands of polygons and was extremely high resolution. As explained in Chapter 4.2, there is no need to use the high-poly mesh in a game when details from the mesh can be baked into a texture map that can be provided to a game engine. In this case, opting for texture map baking meant a 977,921-point difference in vertex count.

This was an excessive amount of resolution even for texture baking and had to be reduced to ensure a smooth and less time-consuming baking process in

Substance Painter. To achieve this, the Decimation Master tool was used. Decimation Master is a tool inside Zbrush designed for efficient polygon reduction while keeping all the sculpted detail. It is a fast solution that automatically optimizes the high polycount models, allowing the user to export their models into other 3D software packages. (Zbrush n.d.) As a result of decimation, the vertex count was reduced to 189,236 points which in this case equals 386,088 triangles.

The models were exported into Substance Painter where the detail information from the high-poly model was transferred to the surface of the low-poly model in the form of a texture map that mimics the appearance of the high-poly model. This is what essentially texture baking is (FlippedNormals 2021). A normal map was generated, along with additional maps, such as curvature and ambient occlusion (AO) maps. However, only the normal map was intended for later use in the game engine, while the other maps were created solely to assist with creating the albedo map.

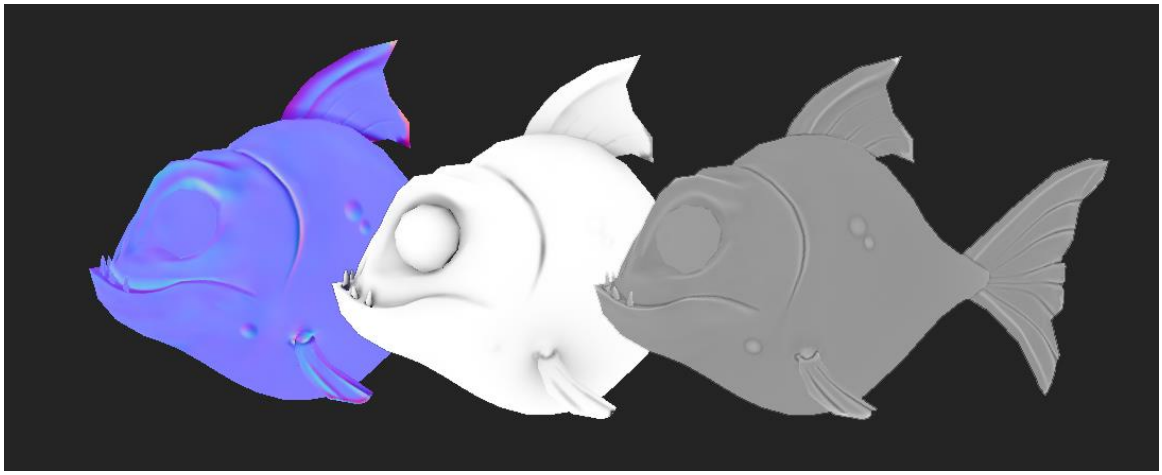

Figure 12. Normal, AO and Curvature maps displayed on the model

The texturing process will not be described here in detail as it does not directly relate to optimization. As explained previously, the created texture maps were used in Substance Painter to help with texturing the model procedurally. Working with texture maps helped achieve the desired look without the need to spend a considerable amount of time hand-painting the entire character. After the texturing was completed the albedo- and normal maps were exported into the game engine. A roughness map was also exported but was not used in the game

as the team had decided to use a certain art style and shaders for the in-game characters that did not utilize roughness maps. However, the roughness map would later be used outside the game engine, for instance, when images for key art were rendered.

Figure 13. Final render of the Piranha character

Finally, the character was brought to life with an animation which was carried out in Maya. The model underwent rigging and animation within the software, and the resultant animation, coupled with the 3D model, was exported in FBX format from Maya to Unity for integration.

6.3.2 Modeling the Barrel

The modeling process of a second game asset, the barrel, started with reference gathering, similar to the approach taken with the previous 3D model. As stated before, the modeling process differed from the previous one in such a way that this time a low-poly version was done first. After the final low-poly version, which would be used in the game as it is, was finished, the high-poly mesh was created. As the polycount was kept low from the beginning, the model did not have to go through the time-consuming process of retopology and polygon reduction, which

ultimately, along with the non-complexity of the model, was the reason for low poly to high poly workflow.

Figure 14. Wireframe, the final textured model, and high poly mesh

The process from here on followed the same steps as were taken with the piranha model: the high-poly mesh was created from the low-poly and the details were sculpted into the high-poly mesh in Zbrush. The detail information was then baked into texture maps and the 3d model was then textured in Substance Painter.

6.4 Optimizing for Mobile Platform

Once the 3d models were ready to be implemented in the game engine, it was necessary to go through which optimization methods could be applied and which measures would be useful, as not all the optimizations would be beneficial in terms of performance. Polygon reduction was already done as explained in the previous chapters as well as baking details from a high-poly model into a normal map. With these optimization methods, a significant amount of the polygon budget was saved as can be seen in Table 1. These measures also had a major impact on the file size which would later translate into build size as well as the loading time of the game.

The texture map size was decided for each asset individually by considering how close the camera can get to a particular object. The goal was to use the lowest

possible texture map size that passes the eye test to achieve the best performance and the most compact file size. Keeping the texture resolution small also saves memory and shortens loading time. The textures were also compressed by using Unity's built-in ASTC compression system to reduce memory load.

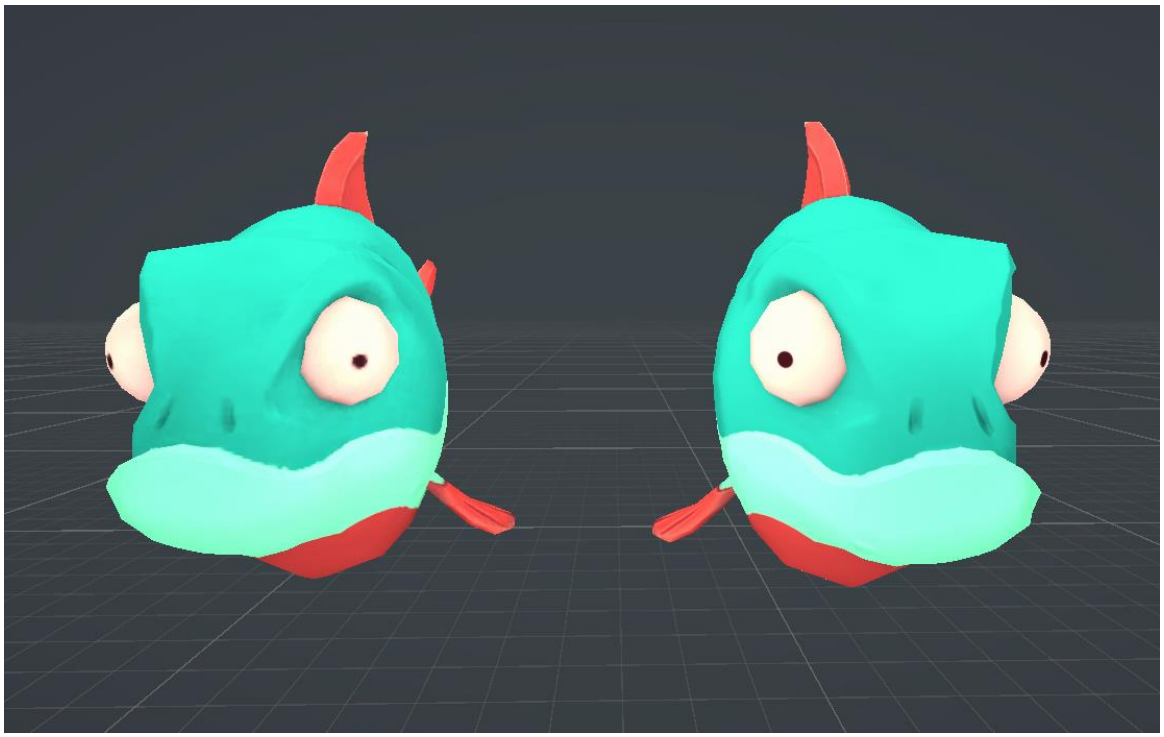

Figure 15. Texture map quality comparison in Unity. 512 resolution (left) and 4k resolution (right)

When inspecting the textures up close, only minor differences could be seen between the visual quality of 512 and 4k-resolution texture maps (Figure 15). Considering the tiny size of the characters on screen, it would not make any sense to use 4k textures in this case as the difference between the two would be unnoticeable on a screen of mobile device. In fact, the texture resolution could be lowered even further without noticeable loss in quality should there rise a need for further optimization.

Given that the camera angle and view distance do not change throughout the gameplay experience, there was no necessity for mipmaps. The same applies to Level of Detail (LODs). Using mipmaps or LODs would only decrease performance as all the texture maps would be loaded into the GPU memory and

separate 3D models would have to be created for every LOD set, which would increase the build size. Not opting for mipmaps also made the mipmap streaming redundant. These optimization methods have their use cases, but for this game, the decision was made not to employ them as they would not have provided any boost in performance.

6.5 Performance Comparison

To receive performance data from the newly created 3d models, a performance test had to be carried out. The device used for the performance test was an Android phone released 6 years ago (2017) at the time of writing. The game, in general, is not very performance-heavy and ran seamlessly on a testing device at a stable 60 frames per second (fps) with no noticeable stuttering or lag. To assess the game's performance on older mobile devices, an Android phone from 2015 was introduced for testing. Unfortunately, the game did not run beyond the main menu into the gameplay for unknown reasons as the author was encountered by a black screen. As a result, a performance test could not be conducted on the device. Nevertheless, it was a valuable experiment as it brought a different issue to the surface for the team to address.

It could be concluded from the initial test that further optimization of the game or the 3D assets was unnecessary, and the optimizations carried out so far were sufficient as the game was running smoothly on the target device. Instead of attempting to further optimize for better performance, it was decided that a new testing environment would be created. In this environment, the assets would intentionally be de-optimized to obtain comparative data on how the game would perform without any optimizations.

The performance tests were carried out on a desktop computer using the Unity game engine and its performance profiling tools for collecting performance data. A desktop computer was selected as a platform for testing instead of an Android device to save time by avoiding the need to create and transfer builds between devices. It is also more convenient to receive reliable performance data from Unity Performance Profiler than from Android performance profiling apps that are

not as sophisticated. It is worth noting that profiling a Unity game from within the editor provides different results than profiling a Unity game from a build as the editor itself will take up resources.

In Table 1, a comparison between two different 3D models and two different texture maps is displayed, along with their properties. The unoptimized model in this case refers to the decimated high-poly mesh which was previously used for texture baking.

Table 1. 3D model and texture properties in comparison

Piranha 3D model	Unoptimized	Optimized
Vertices	189236 points	736 points
Polycount	386088 tris	1464 tris
Filesize	17508 kb	94 kb
Texture map (albedo)		
Resolution	4096x4096	512x512
Filesize	4013 kb	171 kb
Texture compression	High Quality (ASTC 4x4)	Low Quality (ASTC 8x8)
Memory load	21.3 MB	85.4 KB

The game was tested initially with the optimized Piranha 3d model in a test level created solely for this purpose. This level included the same lighting setup, game assets, and functionality identical to an actual level from the game. The goal was to identify any possible bottlenecks or areas in the asset that could be optimized further. A screenshot of the game along with Unity's statistics panel and performance profiler window from the initial test run can be seen in Figure 16. More information on how to read the Unity statistics window can be found in Appendix 2.

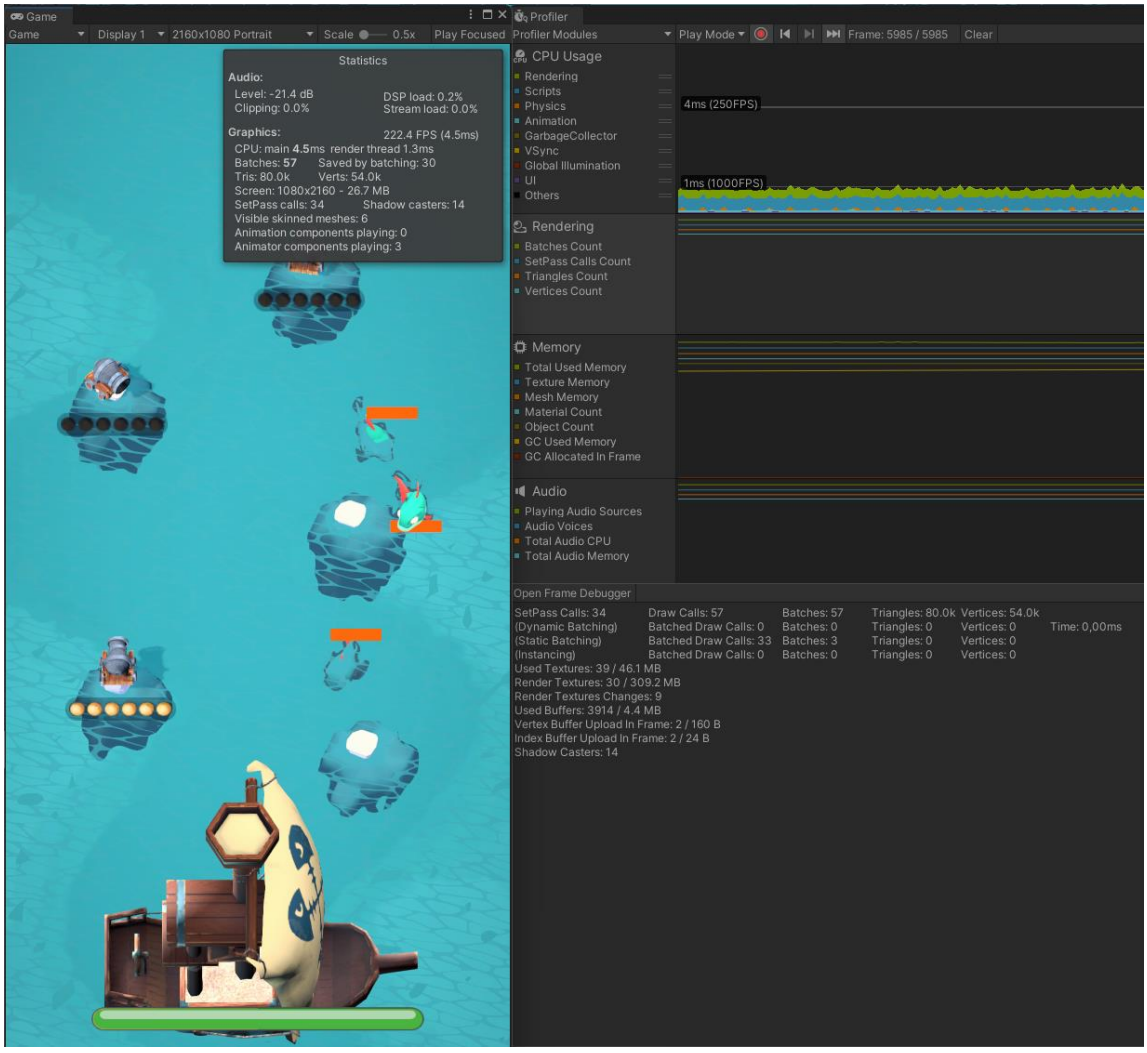

Figure 16. Screenshot of the test environment

As can be seen in Figure 16, the statics panel's *Visible skinned meshes* display the number 6 although there are only three animated characters in the scene. This means that each of the characters consisted of two skinned meshes instead of one. This is not recommended in the Unity manual as it states that using two skinned meshes instead of one can roughly double the rendering time for that model, and there is rarely any practical benefit to using multiple skinned meshes (Unity Documentation 2023). Closer inspection of the model revealed that the tail of the fish had not been merged with rest of the model and needed to be combined for better performance. Not only did the performance test reveal an optimization defect with this particular model but it was also able to bring similar problems to the author's attention, and the team was able to save a significant amount of draw calls by fixing similar issues with other 3d assets.

In the second test, the number of piranhas was increased to fifteen to see how much FPS would drop should there be high amounts of these characters simultaneously in the scene. The drop in FPS was quite significant, despite the relatively small increase in the number of vertices. Results are displayed in Figure 17.

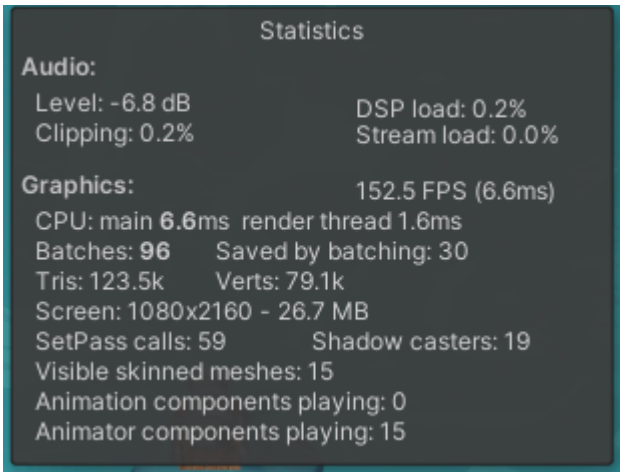

Figure 17. Performance test with 15 optimized models

The same amount of unoptimized piranha models (15) was used in the third test to see how they perform versus the optimized 3d models. As expected, the frames per second dropped significantly from around 150 to 95, which equals a 37% drop, as the scene now had a lot more triangles and vertices to render.

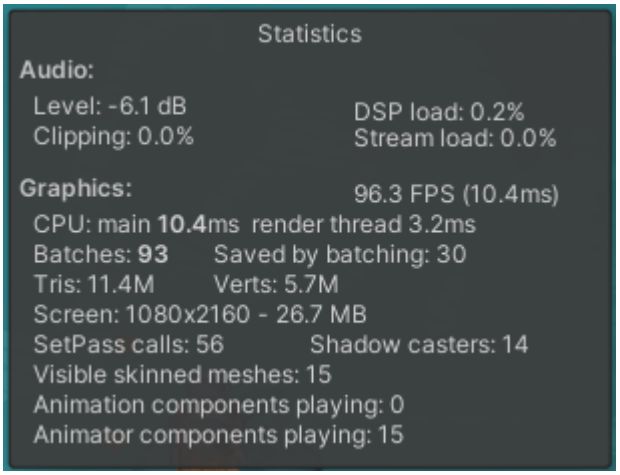

Figure 18. Performance test with 15 unoptimized models

One aspect that is not obvious and cannot be seen from the statistics window is how the larger texture size affected the results. The short answer is they did not as the texture size does not directly affect FPS or rendering time if there is enough memory to accommodate the textures. However, larger textures can indirectly impact performance if they exceed the available GPU memory, causing the system to swap data in and out of memory, which can result in a loss of performance. Additionally, problems with textures can arise when simultaneous texture transfers consume a significant portion of memory bandwidth. Such a situation may appear to the player as a game freeze, typically for a few milliseconds, as the shader waits for data to arrive. (O'Conor 2017.)

It can be concluded from these tests that keeping the 3d model's vertex count low is important to ensure the game runs smoothly. The surprising result was that only a 37% drop in fps was seen, with a vertex count difference being over 70x. Although the drop could be more significant in a mobile environment, it was anticipated by the author that the impact of a such high vertex count would be more drastic.

7 FUTURE TRENDS

The initial purpose of this thesis was to explore ways artists can optimize their 3D models for mobile games. However, as the research progressed it became evident that the significance of optimization from the artist's perspective is relatively diminished due to the limited selection of tools at their disposal, and much of the optimization falls into the hands of a graphics programmer or a technical artist (O'Conor 2017). Matthew Spencer (2023) demonstrated the potential of Unity and pushed the engine to its limits in his YouTube video. The results were remarkable as he showcased the ability to render 2.7 million beveled and animated cubes at a frame rate of 20 frames per second (fps), compared to the initial engine performance of 8 fps for only 90,000 cubes. This technology is available today; new technology is being developed and released daily and mobile devices are getting increasingly more powerful. It can be anticipated that certain optimization methods presented in this thesis, such as polygon reduction,

may become outdated relatively quickly and may not be required in the future when technologies similar to Unreal Engine's *Nanite* become increasingly common.

7.1 Advancements in Mobile Devices

The progress in smartphone technology related to hardware, processing capabilities, and graphics has significantly boosted the popularity of mobile gaming, surpassing PC and console gaming as was highlighted in the 2023 Gaming Trends Report (Google 2023). The advancements have allowed game developers to create visually stunning and immersive virtual worlds that run seamlessly on portable smartphones that can fit into gamers' pockets.

In addition to processing power and graphics capabilities, the display technology of mobile devices has advanced significantly. The displays are getting better with richer colors and higher refresh rates to give players a seamless and engaging gaming experience. The introduction of touchscreens and multi-touch capabilities has transformed the gaming landscape, on smartphones and tablets. Touchscreens offer a direct way to interact with games while multi-touch support allows for gestures and complex inputs expanding the range of gameplay possibilities.

These technological advancements bring an increasing number of games to mobile platforms as the demand increases along with the device capabilities. Given these advancements, more mobile games have turned into 3D graphics, as 8 of 10 most downloaded mobile games in 2023 were 3D games (Curry 2024). The availability of high-speed internet connection, including 4G and 5G networks, has facilitated multiplayer gaming also for mobile. This has encouraged game studios to bring their existing, highly successful multiplayer games to the mobile platform. These developers have long documented the promise of a cross-platform approach that allows game studios to reach a wider audience of players. A prime example is Fortnite (2017) and Call of Duty (2019) whose creators

recognized the power of the mobile market long ago and brought their game to the mobile devices.

Continuation of the advancements can be anticipated as the development of mobile devices shows no signs of slowing down especially now as the emergence of artificial intelligence (AI) and machine learning is forecasted to exponentially accelerate the progress of technology. This will inevitably affect mobile gaming as well. (Gilday 2022.)

7.2 Emerging Technologies and Implications

The game industry is rapidly expanding as is the technology around it. One notable technological advancement is the Nanite feature released with Unreal Engine 5. Nanite is an extremely efficient way to render triangles on screen. It is a LOD system that uses a feature called cluster culling that splits the 3d models into clusters that adjust based on size on screen, distance, and resolution. This allows users to implement 3d models with millions of triangles into the scene without a heavy drop in performance, essentially eliminating the need for developers to create low poly models from their high poly meshes. (Faucher 2021.)

A similar system for Unity is currently under development at the time of writing. Unity developer Chris Kahler unveiled his upcoming mesh rendering system, Nano Tech, which was inspired by Unreal Engine 5's Nanite. While the Nanite is not built to work on mobile processors and does not support Android, Nano Tech is promised to be available on mobile devices as well, allowing also mobile game developers to improve the productivity and quality of their work. (McKenzie 2022.)

These are great examples of evolving technology and ways game developers keep searching for solutions to improve their workflows and save time spent on certain tasks. Despite the evolving technology and increasingly better access to optimization tools, it is crucial to remember that these tools are not here to replace human labor but rather to streamline the process. As discussed with Haanpää during the interview (Appendix 1), often, the sacrifices made to visual

quality are not worth the time saved by using automation tools, and far better results can be achieved by applying certain optimizations manually.

8 CONCLUSION

This thesis set out to discover the optimization methods available for mobile game developers and deepen the author's knowledge of 3d game asset optimization. Those goals were successfully met and research questions, such as how to optimize 3d models without sacrificing too much quality, were answered through the literature review and the interview conducted. The literature review revealed numerous optimization methods available, while the interview provided insight into industry best practices and methods currently in use, enhancing the author’s knowledge in the form of a very comprehensive information package.

The revealed optimization techniques were put into practice later in the practical part of the thesis. With the help of performance profiling and studied theory, the author was able to pinpoint some of the weaknesses in the game's and its 3d model’s performance and address them properly using the discovered optimization methods. Unfortunately, the performance testing was mostly limited to the Unity and PC environment due to limited devices and software available for the author. Nevertheless, the impact of the optimization measures can be calculated with good accuracy and translated to mobile environments now that the cause-and-effect relationship is known.

In summary, this thesis emphasizes the role of optimizing 3D models in the development of mobile games. Through experimentation and analysis, this research showcases how effective 3D model optimization can enhance a game’s performance leading to improved user experience and optimal efficiency on mobile devices. By exploring optimization techniques and striking a balance between performance and visual quality, this study offers perspectives for advancing 3D mobile game development as the developers continue to create engaging gaming experiences that can be enjoyed worldwide right from players' pockets.

REFERENCES

Activision. 2019. Call of Duty: Mobile. Video game. Santa Monica, California: Activision.

Collins, T. 2018. 3D Modeling Pipeline. Blog. 22 June 2018. Available at: https://medium.com/@homicidalnacho/3d-modelling-pipeline-bd9be7dba136 [Accessed 06 April 2024]

Curry, D. 2024. Most Popular Mobile Games (2024). Web page. Available at: https://www.businessofapps.com/data/most-popular-mobile-games/ [Accessed 13 April 2024]

Data.ai. 2024. State of Mobile Gaming. PDF Document. Available at: https://www.data.ai/en/go/state-of-mobile-gaming-2024 [Accessed 06 April 2024]

Edesberg, A. 2023. Optimizing your 3D models for better video game performance. Web page. Available at: https://www.sloyd.ai/blog/optimizing-your-3d-models-for-better-video-game-performance [Accessed 03 December 2023]

Epic Games. 2017. Fortnite. Video game. Cary, North Carolina: Epic Games.

Faucher, W. 2021. Nanite: Everything You Should Know [Unreal Engine 5]. Youtube. Video clip. 15 June 2021. Available at: https://www.youtube.com/watch?v=P65cADzsP8Q [Accessed 28 January 2023]

FlippedNormals. 2020. The Right Way to Use Reference when Sculpting. Youtube. Video clip. 12 November 2020. Available at: https://www.youtube.com/watch?v=6QMtydVP4ZM&ab_channel=FlippedNormals [Accessed 23 December 2023]

FlippedNormals. 2021. What is Texture Baking?. Youtube. Video clip. 11 February 2021. Available at: https://www.youtube.com/watch?v=9NfVyxHtKW0 [Accessed 08 April 2024]

Follygon. 2020. Why Blockout?. Youtube. Video clip. 20 February 2020. Available at: https://www.youtube.com/watch?v=F7frk_7P00o&ab_channel=Follygon [Accessed 23 December 2023]

Garbett, S. 2023. How to optimize 3D models for game development. Web page. Available at: https://www.makeuseof.com/optimize-3d-models-for-game-development-tips/ [Accessed 03 December 2023]

Gilday, T. 2022. Accelerated Future: An Exponential Lens on Emerging Technology and Risk. General Dynamics Information Technology. Blog. 3 June 2022. Available at: https://www.gdit.com/perspectives/latest/accelerated-future-emerging-technology-and-risk/ [Accessed 30 December 2023]

Google. 2023. Gaming Trends Report. PDF Document. Available at: https://services.google.com/fh/files/misc/gaming_market_insight_research.pdf [Accessed 20 January 2024]

Gosch, P. 2016. 3D model optimization for mobile games. Web page. Available at: https://www.saphirestudio.at/wptest/3d-model-optimization-for-mobile-devices/ [Accessed 03 December 2023]

Gregory, J. 2018. Game engine architecture. Boca Raton: Taylor & Francis, CRC Press.

Helppi, V. 2014. Mobile Game Testing – Part #3: Graphics Performance Makes UX Good or Bad. Blog. 10 September 2014. Available at: https://smartbear.com/blog/mobile-game-testing-part-3-graphics-performance/ [Accessed 30 December 2023]

Irsyad, N. 2023. The 3D art pipeline: from asset modeling to in-game integration. Web page. Available at: https://medium.com/mighty-bear-games/the-3d-art-pipeline-from-asset-modelling-to-in-game-integration-51ed16586704 [Accessed 03 December 2023]

Komado, K. 2023. How long before a phone is outdated?. Web page. Available at: https://eu.usatoday.com/story/tech/columnist/komando/2023/10/22/how-to-find-smartphone-expiration-date/71255625007/ [Accessed 26 December 2023]

McKenzie, T. 2022. Nano Tech: An Upcoming Mesh Rendering System for Unity. 80lv. Blog. 9 June 2022. Available at: https://80.lv/articles/nano-tech-an-upcoming-mesh-rendering-system-for-unity/ [Accessed 28 January 2023]

Modjadji, K. n.d. What are the best profiling tools to identify performance issues in your game? LinkedIn post. Available at: https://www.linkedin.com/advice/0/what-best-profiling-tools-identify-performance-4ng5c#what-are-the-best-profiling-tools-for-your-game? [Accessed 29 December 2023]

O'Conor, K. 2017. GPU Performance for Game Artists. Web page. Available at: https://www.makeuseof.com/optimize-3d-models-for-game-development-tips/ [Accessed 09 April 2024]

Roy, A. 2016. The Android game developer's handbook. Birmingham: Packt Publishing Ltd.

Scolastici, C. Nolte, D. 2013. Birmingham: Packt Publishing Ltd.

Slick, J. 2020. What is 3D modeling?. Web page. https://www.lifewire.com/what-is-3d-modeling-2164 [Accessed 29 November 2023]

Smith, A. 2014. Why first impressions matter for Free-to-play (F2P) games especially. Web page. Available at: https://www.gamedeveloper.com/design/why-

first-impressions-matter-for-free-to-play-f2p-games-especially [Accessed 03 December 2023]

Spencer, M. 2023. How To Render 2 Million Objects At 120 FPS. Youtube. Video clip. 26 March 2023. Available at: https://www.youtube.com/watch?v=6mNj3M1il_c [Accessed 29 December 2023]

Unity. n.d.a. What is real-time rendering in 3D and how does it work?. Web page. Available at: https://unity.com/how-to/real-time-rendering-3d [Accessed 07 December 2023]

Unity. n.d.b. Mobile optimization tips for technical artists - Part I. Web page. Available at: https://unity.com/how-to/mobile-game-optimization-tips-part-1 [Accessed 29 December 2023]

Unity Documentation. 2023. Web page. Available at: https://docs.unity3d.com/ [Accessed 10 December 2023]

Zbrush Documentation. n.d. Web page. Available at: https://docs.pixologic.com/user-guide/ [Accessed 13 January 2024]

LIST OF FIGURES

LIST OF TABLES

INTERVIEW 24.1.2024 - EMILIA HAANPÄÄ

1. Name and what do you do for a living? What is your background?

I'm Emilia, a Senior 3D Artist and I've been in the games industry for about 7 years. My upper secondary education is in 3d modeling & animation. Later I went to Xamk to study Game Design and graduated in 2018.

2. What kind of games have you been involved in making?

I've been working on a mobile game called Redecor for several years, almost from the very beginning of the project when I started in my current company as a 3D Artist. The game has been released for Android, iOS, and web server platforms. I've previously worked as an artist in smaller companies and as a 3D artist in several VR projects. My current company was a start-up as well when I started on it, and it has grown since.

3. Do you use Unity, Unreal Engine, or any other proprietary game engine?

We don't use Unity or Unreal Engine, we have our in-house game engine, but I am an experienced Unity user and have some experience with Unreal Engine as well.

4. Can you tell me about the 3d modeling pipeline, what are the steps involved? Are these steps usually clearly divided to whom each step belongs? For example, artist's / programmer's task.

1. **Conceptualization and ideation** – The request often come from outside, whether it's a monetization expert, programmer, game designer, or someone else, that we need such and such an asset for the game and it should be of a certain type and have a certain purpose. Someone gives a framework from which to start developing.
2. **Sketch, suggestion** – There are different ways of sketching, some people like to gather a bunch of pictures from Pinterest and present their idea of the 3d model they will be creating, and others like to sketch their ideas on paper. The method doesn't matter if the idea and thoughts are conveyed clearly.
3. **3D modeling** – Either a high-poly or low-poly model is made first depending on what we are creating, but either way both are being created most of the time. The order doesn't really matter.

4. **UV mapping and texturing** – Once the low-poly is done, we move to UVs and texturing.
5. **Export and integration** – 3D models and textures are exported to the game engine.
6. **Testing and iteration** – The artist tests that everything works as it should and looks good on the screen of the mobile device. This must be done by the artist who created the asset.
7. **Shaders and shader optimization** – This is usually done by the dev side. The artist can ask the coders if they need a specific type of shader. The artist usually doesn't make the shaders or optimize them themselves, but someone else does, such as a tech artist or programmer.
8. **Performance testing** – Someone (other than the Artist) does performance testing for the newly created asset and if it needs more optimization, a lower vertex count for instance, it comes back to the artist.
9. **Finishing and publishing** – Once the asset is game-ready and optimized it can be published and integrated into the game.

5. When creating a mobile game, how do you determine for what and how old devices will the game be optimized for?

Device optimization is mainly based on data collected from the Play Store and Apple Store. For each game downloader, the store collects information such as age, gender, and the device on which the game was downloaded. This data is used to make optimization decisions. For example, if some less common devices or operating system versions cause compatibility problems, it is necessary to weigh up whether it makes sense to spend time on solving these problems. Especially if a great amount of work is required to optimize for or fix the issues occurring with these devices.

6. How do you optimize 3d models in your company? What optimization methods are used?

The programmers know the game and its performance and constraints best, so that's where the criteria come from, and those are what we usually try to follow. These criteria may include file format (obj/fbx), maximum file size, texture size, texture file format (jpg/png), which texture maps to use, and what vertex count to aim for. However, the artist's primary task is to focus on making high-quality, good-looking assets, with performance being a secondary consideration, without losing sight of it. A balance between the two is constantly being sought, and this strongly involves testing 3d models in a game environment. As for optimization methods, LODs, for example, are not useful in our game.

7. At what point in the pipeline do most of the optimizations of 3d assets take place? And by whom?

The artist optimizes as much as they can. This can be for example reducing the number of vertices or optimizing textures. The programmers also do their part. For instance, the programmers may have built a compressor through which textures and 3d models are passed, which then compresses the textures for better performance. Developers can also control the amount of content players download at once when playing the game for a smoother experience.

8. What is done with textures? How do you determine the size of the textures? Do you use Texture Atlases?

There are specifications for textures that you should stick to, e.g. no texture maps larger than 2048px, but the rules can be broken and deviated from if an asset requires it or if it is otherwise well justified. Generally, no more than three texture maps are used, e.g. diffuse, normal, and roughness. Maps can also be combined to save memory and draw calls, such as combining roughness and metalness maps into a single map. Atlases are not used that much, as they wouldn’t boost our game’s performance that much. But they have their use cases too. For example, if there is a lot of vegetation in the game, they can be put on the same texture atlas. Also, trim sheets can be useful in some cases.

Textures are usually compressed and, partly because of that, testing them on a mobile screen is extremely important. They may look very different on a phone screen than on a computer screen, especially when compressed. In particular, roughness and metalness maps can behave erratically.

9. How do you test 3d models and their performance?

Artists do not usually test the performance of 3d models or do performance profiling unless it's an indie studio, where game developers often have to wear many hats due to the small size of their team. This is often done by someone else, such as a programmer, so I can't really answer that. Quality testing is done by artists all the time, though, and as early as possible. The artist makes sure that the assets they build look good and fit the environment it is intended for. There are often situations where I feel I've done everything the 3d model I'm working on is now as optimized as it can be, but it still comes back to my desk. They tell me that my model didn’t pass the performance test and I should still be able to tweak the number of vertices somewhere, or that the textures are too large, and something needs to be done about them. Usually, they are right after all, and I’ll find a way to optimize the 3d model further.

THE RENDERING STATISTICS WINDOW

Statistic	Description
FPS	The current number of frames Unity is able to draw per second.
CPU	**Main**: The total amount of time taken to process one frame. This number includes the time Unity takes to process the frame update of your application and the time Unity takes in the Editor to update the Scene view, other Editor Windows, or process Editor-only tasks. **Render**: The amount of time taken to render one frame. This number includes the time Unity takes to process the frame update for the Game view; it doesn't include the time Unity takes in the Editor.
Batches	The total number of draw call batches Unity processes during a frame. This number includes static, dynamic, and instance batches.
Saved by batching	The number of draw calls Unity combined into batches. To ensure good draw call batching, share materials between different **GameObjects** as often as possible. Batches group draw calls with the same render state, so changing the material causes Unity to create a new batch.
Tris	The number of triangles Unity processes during a frame. This value is important when optimizing for low-end hardware.
Verts	The number of vertices Unity processes during a frame. This value is important when optimizing for low-end hardware.
Screen	The resolution of the screen, along with the amount of memory the screen uses.
SetPass	The number of times Unity switches which **shader** pass it uses to render GameObjects during a frame. A shader might contain several shader passes and each pass renders GameObjects differently. Each pass requires Unity to bind a new shader, which might introduce CPU overhead.
Shadow casters	The number of GameObjects in the frame that cast shadows.

Statistic	Description
Visible skinned meshes	The number of Skinned Mesh Renderers in the frame.
Animation components playing	The number of Animation components playing during the frame.
Animator components playing	The number of Animator components playing during the frame.

The Rendering Statistics window in Unity (Unity Documentation 2023)