UPPSALA
UNIVERSITET

Optimized Graphics for Handheld Real-time CG Applications

Written by Robin Powell 3rd year student,
Undergraduate student at Uppsala University – Campus Gotland, Department of Game Development

Faculty of Arts
Department of Game Design

Robin Powell
Degree Project in Game Design, 15 ECTS Credits
Game Design and Graphics
Supervisor: Jakob Berglund Rogert
Examiner: Dr. Masaki Hayashi
June, 2014

Abstract

The purpose of this theoretical thesis is to research the problem: producing optimized content for handheld real-time CG applications. This thesis is based on existing literature studies regarding the subject. It will explore the topic on how to deal with draw calls, shader optimizations, hard/smooth edges and how engines deal with polygons and textures and how this correlates with current technical limitations of current game engines for handheld real-time applications such as the iPhone. On these grounds, the thesis will expand on the subject of how those topics affect the hardware; more specifically targeted the GPU and CPU on mobile devices.

Today, the hardware used on mobile devices on the market is limited; it is important to understand that a lot of factors come in to play and making optimization a part of the design, not a final step regarding the creation of content is vital when creating CG applications for mobile devices. A closer look at the available evidences suggests that when dealing with the limited resources available every performance aspect should be closely looked into. It is vital to optimize the content and correlate your game budget accordingly to the marketed device. In the end it will make for a better game. This thesis sheds a light on the development strategies for CG applications on handheld devices.

Keywords:
3D, game, art, mobile devices, graphic optimization, 3D models, GPU, CPU

Table of Contents

Foreword

This thesis research focuses on the study of how to optimize modeling for hand held real-time 3D applications in a Game Design and Development process.

Every year the technical capabilities of the game industry are improving as the result of rapidly improving technological developments. Game Development still have problems in improving video games for mobile application, in terms of being able to use as much technical capacity as possible.

The iMac was released in 2000, and the capabilities of this device were powerful for its time. Ten years later the iPhone 4s was released which had the exact same specifications as the iMac, but, was 15,5kg lighter. The capacity of hand held devices has improved year by year since then.

This thesis is based upon the collection and summary of data from previous research regarding optimization of hand held real time 3D applications. The contents will give the readers a better understanding of how to model for handheld real-time 3D applications and how to apply this knowledge in a Game Development process for mobile devices.

The following literature and data has been of great value when writing and conducting the research regarding Optimized modeling for handheld real-time devices and it will be referenced often:

- Creating 3D Game Art for the iPhone with Unity written by Wes McDermott [(1)]
- Unity's - Practical Guide to Optimization for Mobiles [(8)]
- Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck [(19)]
- Beautiful, Yet Friendly Part 2: Maximizing Efficiency [(20)]

The thesis will also expand and speculate upon a conclusion regarding optimized modelling for handheld real-time devices by closely looking at the data and literature regarding, previous and current generation mobile devices.

Nomenclature

As with most research, there are some domain-specific words or phrases that need further explanation. There are also words that already have a general meaning that have been appropriated to describe more specific phenomena. This section will attempt to briefly explain these concepts.

Texture	*"The most common use for a texture is to add detail to the surface of a scene object "* (26)
Mipmap	*"Mipmaps are pre-calculated, optimized collections of images that belong to a main texture; they are a smaller version of the main textures which are used when the textures are displayed in a far of distance."* (26)
Diffuse	*"A diffuse map is a texture you use to define a surface's main colour."* (23)
Normal	*"Virtually all geometric surface detail should be represented in bump maps instead of drawn into the diffuse maps in the conventional style. This allows a single texture to take on different characteristics based on its interaction with lights."* (23)
Specular	*"Specular maps are the maps you use to define a surface's shininess and highlight colour.* *The higher the value of a pixel (from black to white), the shinier the surface will appear in-game. Therefore, surfaces such as dry stone or cotton fabric would tend to have a very dark specular map, while surfaces like polished chrome or plastic would tend to have lighter specular maps."* (23)
Glow	*"Glow maps, also known as self-illumination maps, are used to make surfaces appear like*

	they are emitting light." [23]
UV- mapping	*"UV mapping is a 3D modeling process of making a 2D image representing a 3D model. The map transforms the 3D object onto an image known as a texture.* *In contrast to "X", "Y" and "Z", which are the coordinates for the original 3D object in the modeling space, "U" and "V" are the coordinates of the transformed object wrapped to 2D "UV space"."* [27]
Game Engine	*"A game engine, also referred to as graphics engine, is a software system designed for creating and development of videogames, containing physics calculations and interactions. It imports polygon objects and renders them in real-time."* [28]
Unity 3D	*"A multi-platform game development tool. The editor runs on Windows and Mac OS X and can produce games for the Windows, Mac, Wii and iPhone platforms, including browser games."* [29]
Smartphone	*"A smartphone is a mobile phone with more advanced computing capability and connectivity as compared to basic feature phones."* [30]
NVIDIA	*"NVIDIA is worldwide renown manufacture of graphic processing units (GPUs), as well as having a significant stake in manufacture of system-on-a-chip units (SOCs) for the mobile computing market"* [31]
SOC	*"A system on chip is a combination of all components of a computer or other electronic system into a single chip; they are often used in mobile devices."* [32]
iOS	*"iOS is a mobile operating system developed*

	by Apple Inc. and distributed exclusively for Apple hardware." (33)
Vertex	*"A vertex, in computer graphics is an object, which has certain attributes, one of which is its position."* (34) *"A polygon consists of vertices."* (34)
Polygon	*"A plane figure which is a closed contour of straight lines. A basic primitive in the graphical representation of 3D objects. A triangle is the simplest polygon."* (34)
Triangle	*"Three vertices which are connected to each other by three edges is the definition of a triangle."* (34)
Resolution	*"The amount of data that is used to capture an image. The term is typically used to refer specifically to the spatial resolution of a digital image."* (35)
CPU (Central processing unit)	*"A CPU is the hardware within a computer that carries out the instructions of a computer program by performing the basic arithmetical, logical, and input/output operations of the system."* (36)
GPU (Graphics processing unit)	*"GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel".* (37)
Level of Detail (LOD)	*"Level of detail involves decreasing the complexity of a 3D object representation as it moves away from the viewer or according other metrics such as object importance, viewpoint-relative speed or position."* (1)

Material Batching	*"Unity can combine a number of objects at runtime and draws them together with a single draw call. This operation is called "batching". The more objects Unity can batch together, the better rendering performance (on the CPU side) you can get."* (13)
Real-Time	*"Real-time computer graphics is the subfield of computer graphics focused on producing and analyzing images in real time. The term is most often used in reference to interactive 3D computer graphics, typically using a GPU, with video games the most noticeable users."* (38)
Smoothing Groups	*"In 3D computer graphics, a smoothing group is a group of polygons in a polygon mesh which should appear to form a smooth surface."* (1)

1. Introduction

This thesis is intended as a theoretical paper, the thesis builds on the problem: producing optimized content for handheld real-time applications. It is based on existing literature studies regarding the subject as the theoretical problem will be analyzed.

In this thesis it will explore the topic on how to deal with draw calls, shader optimizations, hard/smooth edges and how engines deal with polygons and how this correlates with current technical limitations of current game engines for handheld real-time applications such as the iPhone (iPhone 1st gen, iPhone 3G, iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPhone 5C, iPhone 5S).

The purpose of this thesis is to explore issues of efficiency and optimization for a 3D game engine in order to accurately portray dynamics and believable game worlds in handheld real-time applications. It will also shed light on the optimizing process for real-time graphics on handheld devices by using the iPhone and Unity3D. The thesis will explore how the GPU and CPU are affected by textures and meshes and how this correlates to the engine, in order to summarize how to make mobile 3d graphics as effective and useful as possible.

In order to gain understating regarding the hardware limitations of current generation mobile devices such as the iPhone and how game objects are rendered on the devices. The thesis will underline how to determine triangle and vertex budgets as well as help outline what should be the focus to enable a mobile game to run as smooth as possible.

The iPhone brand was chosen for this theoretical paper because it has already been clearly bench marked for research purposes. It also has Unity3D support regarding optimization for mobiles, which will be used in order to provide a backbone for the research.

In order to portray the issue, the focus of this thesis will be to address a particular need in game design & development, which is creating 3D content. The outline of this research will not be covering the process of creating an entire game; however it will be taking an in-depth look at literature studies which are focused on 3D content directed towards Unity iOS in order to build a highly optimized video game for a mobile 3D application.

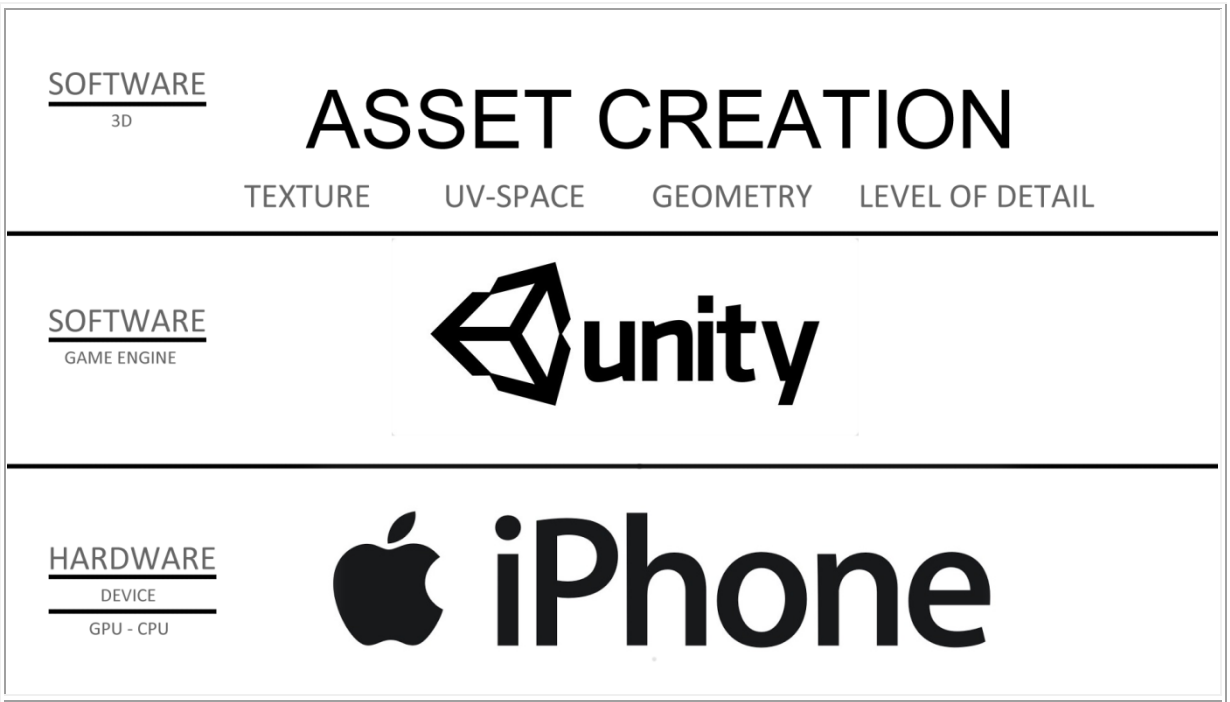

Figure 1.1 Thesis Outline

This study will draw on the previous literature studies made on the following area. It will attempt to gather and research the best ways to create assets by researching how to optimize:

- Textures
- UV-Space
- Geometry
- Level of Detail

Upon further investigation it will then proceed to monitor how these assets will affect the engine Unity.

Finally, the thesis will expand on the subject of how those assets affect the hardware; more specifically targeted the GPU and CPU on the iPhone.

This thesis is aimed for game developers and game artists who are aiming to increase their knowledge basis on the subject of creating optimized 3D art for real-time applications on mobile devices.

2. Background

The App store on the iOS opened July, 10, 2008; now as of February 10, 2012 there are at least 1,100,000+ third party apps available on the market, as of today.

This means that game developers, either professional or indie will have a huge market to tap into. It is rapidly expanding each day, currently there are over 1000 games powered by Unity available on the App Store.

Figure 2.1 Unity games sweep App Store awards

According to a 2012 Game Developer survey, *"Unity is far and away the most popular engine amongst mobile game developers, and they use it to make huge numbers of critically-acclaimed, innovative, mega-grossing games. In fact, no less than seventeen Unity-made games and apps featured in Apple's AppStore USA Best of 2012 list."* (21)

Currently Unity is the world top engine when it comes to creating mobile games. In the Apple's AppStore Best of 2012, see Fig 2.1. Two of the prices awarded to Unity made games were iPad Game of the Year & Top Grossing iPhone App of the Year.

Planning is essential when making 3D environment assets for mobile applications, a design of the optimization possibilities should be the main focus when creating 3D assets for mobile devices. 3D Models are just too performance heavy if they are to blend in with the quality level of everything which is required to create a believable world.

In the book "Creating 3D Game Art for the iPhone with Unity iOS" written by Wes McDermott with over 12 years' experience in the industry as a 3D Artist says: *"Most game art books discuss the typical scenarios of building low-polygon objects, creating normal maps, and so on. But these books never talk about the why."* (1, 2011, p. xi)

This statement applies to the very core of 3D game assets, 3D art is not just art; it is a very technical subject which needs a good understanding if it is to be utilized to it is full potential.

Wes also goes on to say that: *"3D art is technical, and creating 3D games is even more technical. I feel that you can't realistically talk about creating 3D game content without thoroughly discussing how it relates to the game engine and hardware"* (1, 2011, p. xi)

This means the whole concept of creating art is not just visual subject based purely on subjective thought based on aesthetics; on the contrary, it is the subject of an optimized engine; which is required in order to portray a beautiful world. See example Fig 2.2.

It is important to understand how 3D models and textures are used in a game world affects the engine they are running on.

Figure 2.2 Real-time graphics in video game "The Last Of Us"

Everything rendered in 3D game engines consist of triangles, triangles have many properties which make them faster to render. This mean it is vital that every triangle should have an impact on the volume and silhouette of the in-game 3d models. If two triangles do not have at least a slight angle between them they are sharing the same plane and could in most cases be either reduced down to just one triangle or be put to better use.

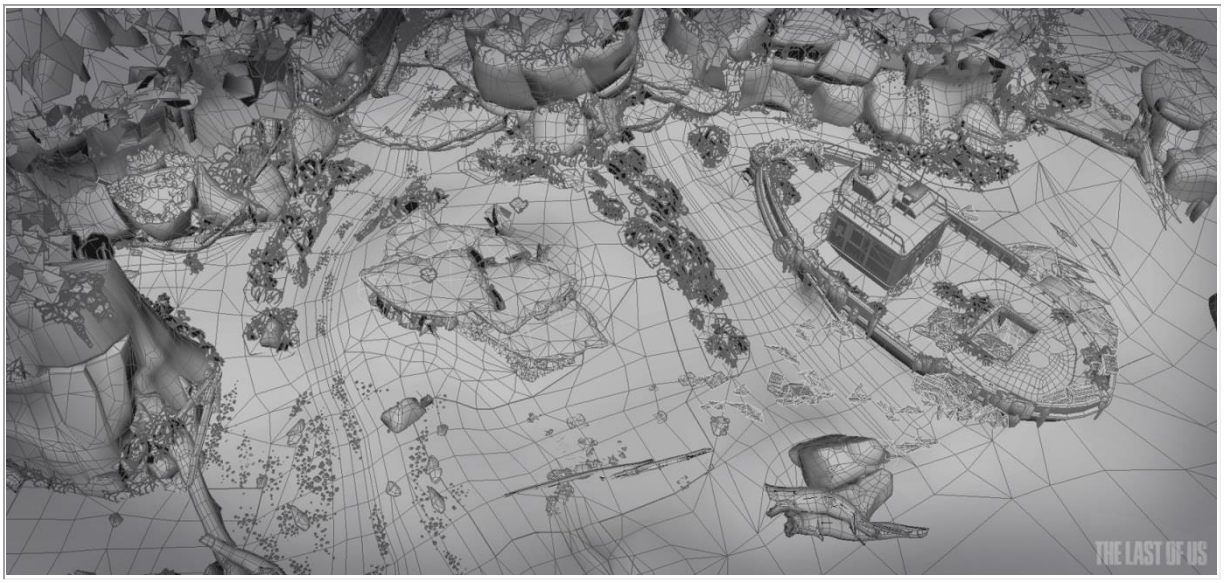

Figure 2.3 Game world rendered in 3D geometry

3D Objects that are used many times in the same scenes, like common vegetation in Fig. 2.3 can quickly escalate the triangle count. For these every single triangle is important since its performance impact is multiplied by the number of instances, which is why performance should be the main aspect applied to creating a 3D model.

The term bottleneck, also known as hotspots will often be referred throughout this thesis; the term is often described as: "the performance or capacity of an entire system is limited by a single or limited number of components or resources."

Reduction of bottleneck points is often achieved throughout optimization or content management of the designated area.

3. Methods and Targets of Analysis

This thesis was based on data collection through the following means:

1. Literature studies on 3D Game Art & iOS Devices
2. Statistic based studies on 3D Game Art & iOS Devices
3. Unity's documentation (User Manual & Component Reference)

Data gathered using the scientifically based research mentioned in Chapter 2 Background, will enable the possibility to attempt and address the issue of to the issues of efficiency regarding a 3D game engine in order to accurately portray dynamics and believable game worlds in handheld real-time applications.

This study draws on research from reliable sources as "Unity's - Practical Guide to Optimization for Mobiles", the book "Creating 3D Game Art for the iPhone with Unity iOS" written by Wes McDermott, the article "Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck", "Beautiful, Yet Friendly Part 2: Maximizing Efficiency" published in Game Developer, written by Guillaume Provost and statistics from NVIDIA which manufactures graphics processing units and hardware for mobile devices.

Data yielded through this study has been applied to methods in the field of game design and production.

Methods of 3d game creation that will be analyzed are as follows:

- Modelling (High Polygon & Low Polygon)
- Polygonal Optimization
- UV-Mapping
- UV-Layout
- Texturing (Diffuse Map, Normal Map, Specular Map)
- Level of Detail
- Material Batching

A closer look at the methods in 3d game creation will yield results in how they affect the following hardware units in mobile devices:

- GPU (Graphical Processing Unit)
- CPU (Central Processing Unit)

The aim of this will be to generalize beyond the data and achieve some insight regarding the production of optimized content for handheld real-time applications.

4. Analysis

In the following Analysis it will explore the subject of optimizing 3D models for hand-held devices into smaller parts and subsections to gain a better understanding of the topic.

Mainly when dealing with 3D content the GPU is mostly used for all 3D rendering in games and applications, the GPU will take over from the CPU to handle rendering more efficiently. However, the CPU will help out in calculating while the GPU is rendering 3D models on screen for games.

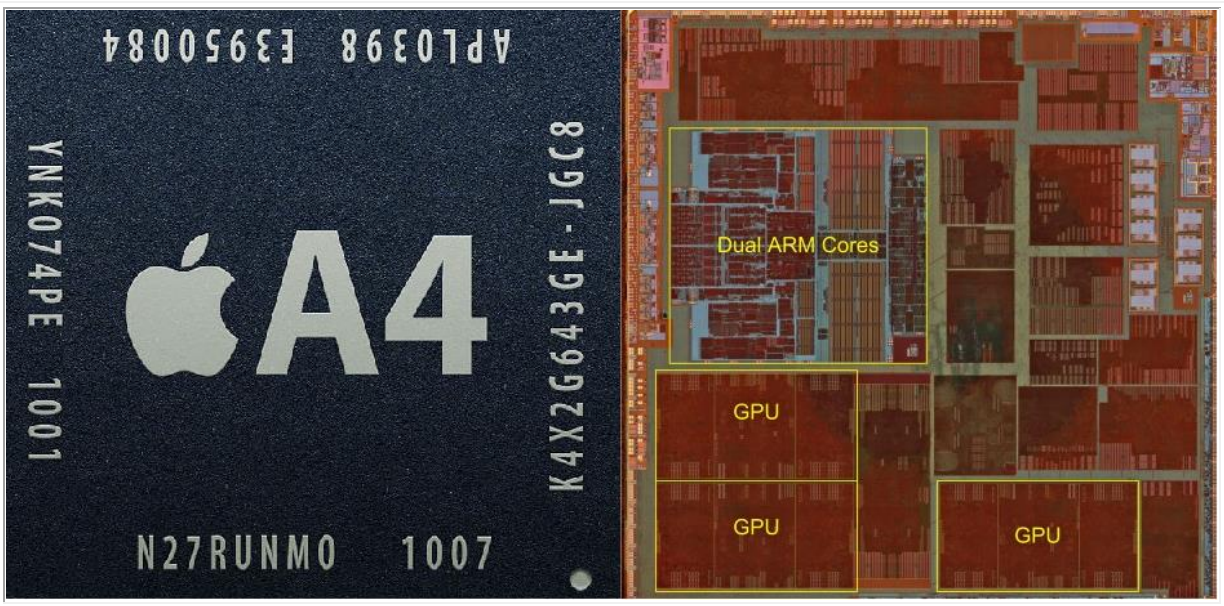

Figure 4.1 iPhone System-on-a-chip (SOC)

According to Unity's support guidelines regarding Optimizing Graphics Performance: *"The first rule of any optimization is to find where the performance problem is"* (7)

They also go on to add that: *"it's quite common to make GPU do more work while optimizing for CPU, and vice versa)."* (7)

The analysis will break down the optimization of polygons, textures, LODs (Level-of-detail) material batching and how this will help relieve stress on the GPU and CPU.

4.1 Mobile Graphics Performance

Mobile graphics performance is moving closer towards console level, the smartphone first really took off in terms of mobile graphics technology around 2007, since then there has been an increasing performance rate in mobile graphics.

In the article "Mobile graphics moving toward console level", posted on NVIDIA's blog, Matt Wuebbling, Director of product marketing writes: *"Another big focus of the industry has*

been to advance the state of mobile graphics. Since the introduction of the smartphone, mobile graphics technology has advanced considerably – and it is expected to close the performance gap with console graphics in the next few years, as shown in the chart below." (16)

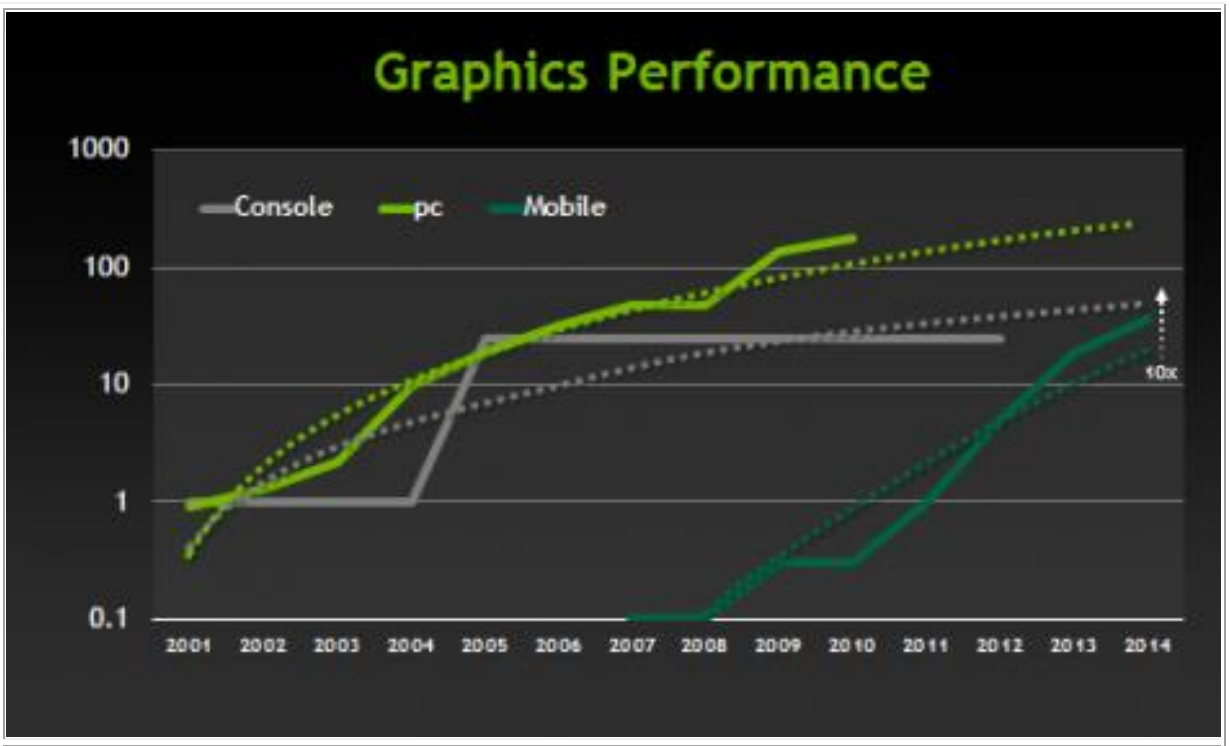

Figure 4.2 NVIDIA Graphic Chart – Graphics Performance

In the diagram, we can clearly see that the gap between smartphones and console graphics is likely to get closer in the next few years, as shown in Fig. 4.2.

4.1.1 iPhone Performance Preview

Performance is critical for mobile games, especially for action packed video games which require a lot of graphical input on screen at one time. The most important thing when constructing a game like this is to keep the triangles, vertices and draw calls as low as possible.

In the book: "Creating 3D Game Art for the iPhone with Unity iOS" written by Wes McDermott, he mentions that: *"The technical understanding behind creating game assets is more in - depth than just modeling low-resolution geometry. Before we can get into actual modeling or creating texture maps, we'll first need to have a solid understanding of the hardware and game engine that the content will run on. Each platform or device will have it's own limitations, and what might run well on one platform doesn't mean it will run as well on another."* (1, 2011, p. 1)

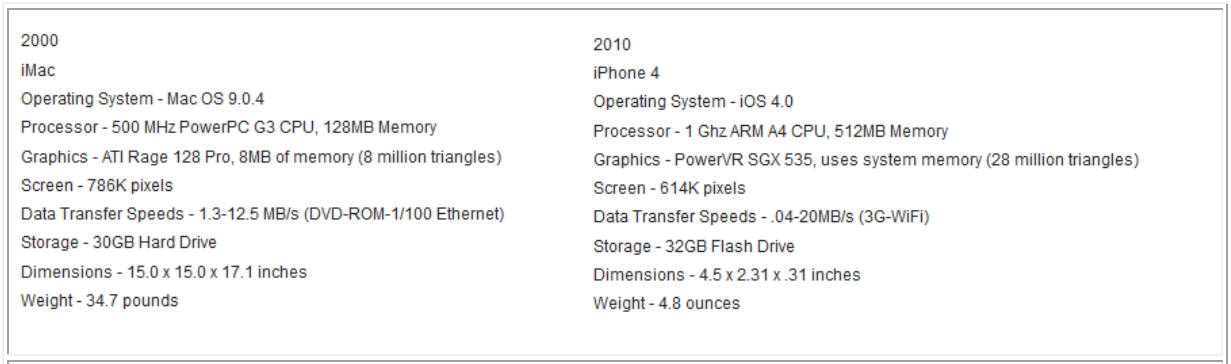

Figure 4.3 iMac vs. iPhone comparison

Wes also goes on to note the following regarding resolution for different iDevices. He mentions that: *"The iPhone 4, 3GS, and iPad all have different screen resolutions, and if you want your graphics to look awesome, you'll need to build to match the resolution of each device. For instance, if you build your game graphics based off the screen of the 3GS at 480 × 320 and then scale these graphics to the iPad at 1024 × 768, then your graphics and textures are going to look pretty ugly as they are scaled from the lower resolution to a higher screen resolution."* (1, 2011, p. 1)

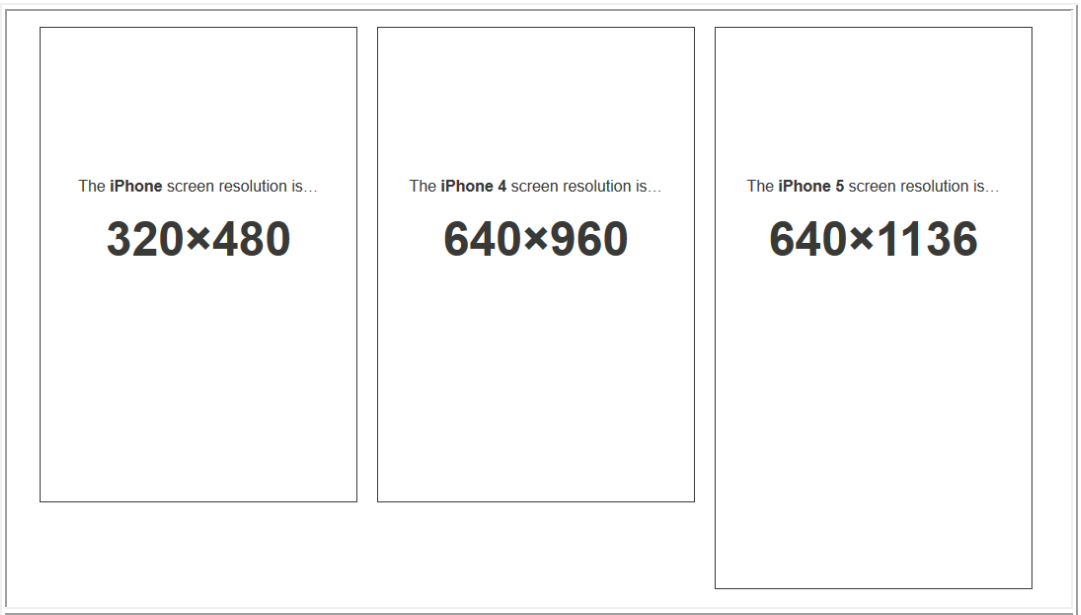

Figure 4.4 iPhone resolution

The point Wes makes regarding resolution is a very important point which needs to be decided upon early in production of making a video game for a mobile devices, because all iPhone devices run on different sets of resolution and hardware, see Fig. 4.4.

Figure 4.5 iPhone RAM

Wes McDermott mentions that:*" There is a difference in RAM among the iDevices. The iPhone 4 contains twice the amount of RAM of the iPad and 3GS at 512 MB while both the*

3GS and iPad contain only 256 MB. It's important to understand the RAM available and what you have to work with. The entire amount of RAM is available to your application, as some of it must be saved for running the OS and other apps with multitasking. Your textures are usually the main culprit when it comes eating up RAM in your game. That's why, it's very important to use optimized compressed textures to minimize the RAM usage in your game." (1, 2011, p. 11)

The more thought that goes into optimizing content and managing it towards the devices strict performance statistics, will result in the video game will look and play better (Fig. 4.5).

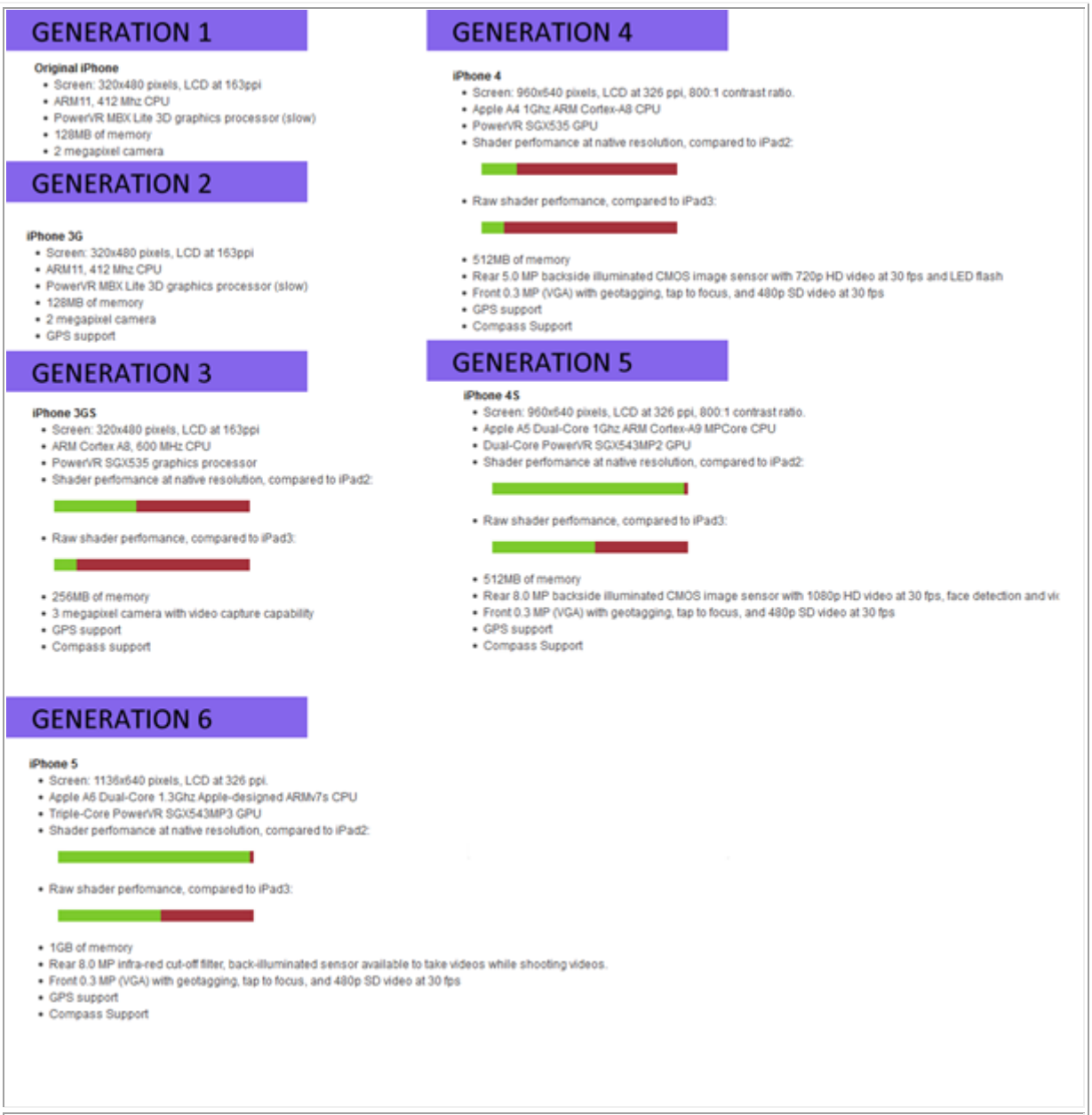

Figure 4.6 iOS Hardware guide & Hardware models

This is very important to note when creating content for the iPhone, currently there are six generations of the iPhone on the market which vary in hardware specifications and capabilities (Fig. 4.6).

In the book: “Creating 3D Game Art for the iPhone with Unity iOS” written by Wes McDermott mentions that: *“You will always need to profile for performance to match your game budget. Faster hardware is a plus and allows you to do more, but you must remember that building for the cutting-edge hardware will inevitably alienate a good degree of your potential market due to users with older models.”* (1, 2011, p. 8)

This means that there are three factors that correlate between the hardware of the iPhone and the content being created for the mobile device, the factors are as follows:

- Performance
- Game Budget
- Marketing

4.2 GPU: Geometry

Optimizing model geometry is the process of modifying the geometrical structure to make some aspect of it work more efficient by using fewer triangles.

In “Unity’s Manual, Practical Guide to Optimization for Mobiles” they mention that British computer scientist Michael A. Jackson is often quoted for his “Rules of Program Optimization”, he said: *“Considering how fast computers are, and how quickly their speed is increasing, there is a good chance that if you program something it will run fast enough. Besides that, if you try to optimize too heavily, you might over-complicate things, limit yourself, or create tons of bugs”* (8)

This statement still applies today, however when developing content for mobile games, there is another factor that comes into play. The hardware which the mobile phones are running on is still powerful for its current age. However, it is still very limited compared to current generation high end computers.

Ultimately this means when creating 3D content for mobile applications, the main aspect should be optimization, so the risk of creating something that will not work on the mobile device will diminish.

On Unity’s website in the section: “Practical Guide to Optimization for Mobiles – Graphics Methods” it mentions: *“The more you respect and understand the limitations of the mobile devices, the better your game will look, and the smoother it will perform. If you want to make a high-class game for mobile, you will benefit from understanding Unity's graphics pipeline.”* (8)

This statement is very true, and it will help you utilize every aspect of a 3D engine for Unity when creating optimized content for mobile graphics. Optimizing 3D assets usually is a very tedious process, however if it is implemented early on in the pipeline it can make for a better mobile game, performance wise.

In the article “Beautiful, Yet Friendly Part 2: Maximizing Efficiency” published in Game Developer, July 2013. June 2013. Guillaume Provost speaks about the topic of optimization, he says: *“Whether they are vertices, texels, objects, or textures, it's more about uniformly distributing them than about plucking out detail. This is a very powerfully intuitive concept: things that are smaller on-screen should get less detail than things that are bigger on screen.* (20)

Guillaume Provost also goes on to add that: *“Programmers can always optimize their code to go just a little bit faster. But there's a hardware limit they can never cross without sacrificing visual quality. If you are pushing the limits of your system, chances are that it is your content -- not code -- that drives the frame rate in your game.”* (20)

This is most likely always the case in some games, the hardware is doing its bit, so is the code, if the game is running poorly it is most likely due to un-optimized content which needs to be looked over and optimized for its specific platform. The high end console graphics are not able to run on a mobile device, not yet anyway, therefore the mobile platform require a lot of optimization regarding texture size, texels, drawcalls, and polygon count.

4.2.1 Polygon Triangulation

All 3D objects that we see on a computer screen consist of geometrical objects often called primitives. Quadrilaterals, triangles, n-gons etc. are example of primitives.

In game engines, everything is converted to triangles, the reason for this is because all geometrical objects can be split into triangles, however a triangle cannot split to anything else than triangles. This concludes that drawing the triangles is a lot easier performance wise than dealing with different geometrical primitives.

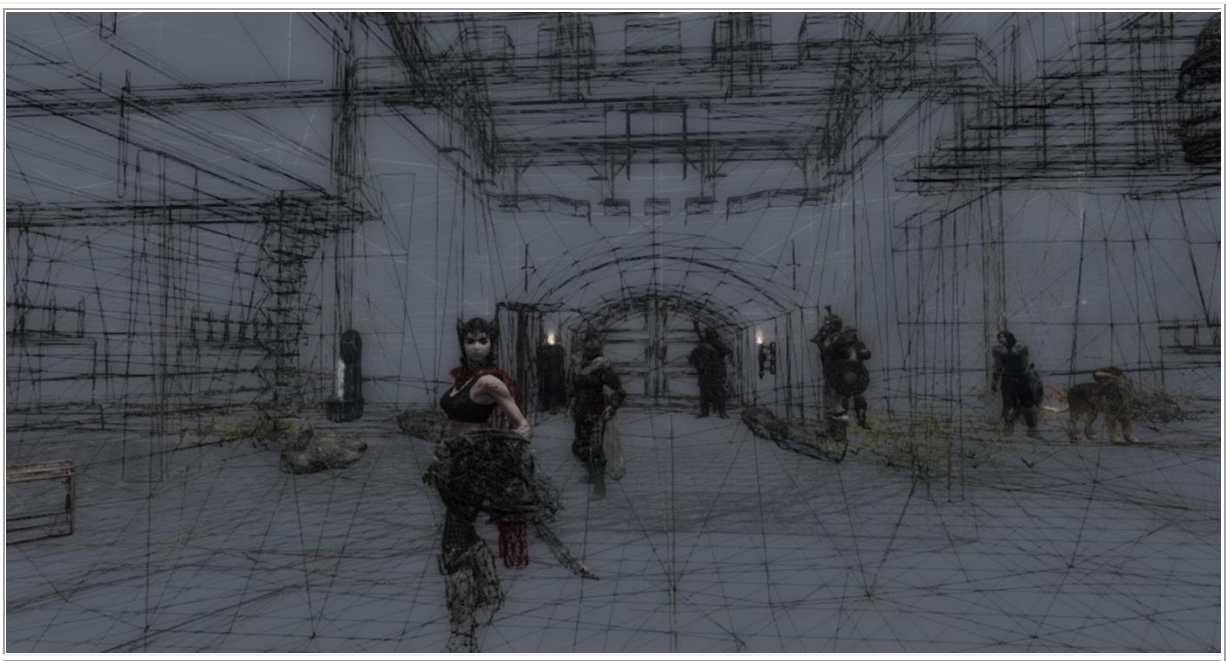

Figure 4.6 Real-time triangulated world

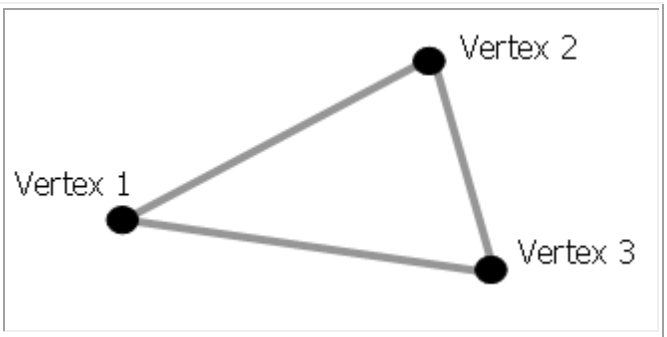

Figure 4.7 Triangle

The advantages of using triangles are that they are the simplest primitive, therefor faster for a game engine to utilize and draw (Fig 4.7).

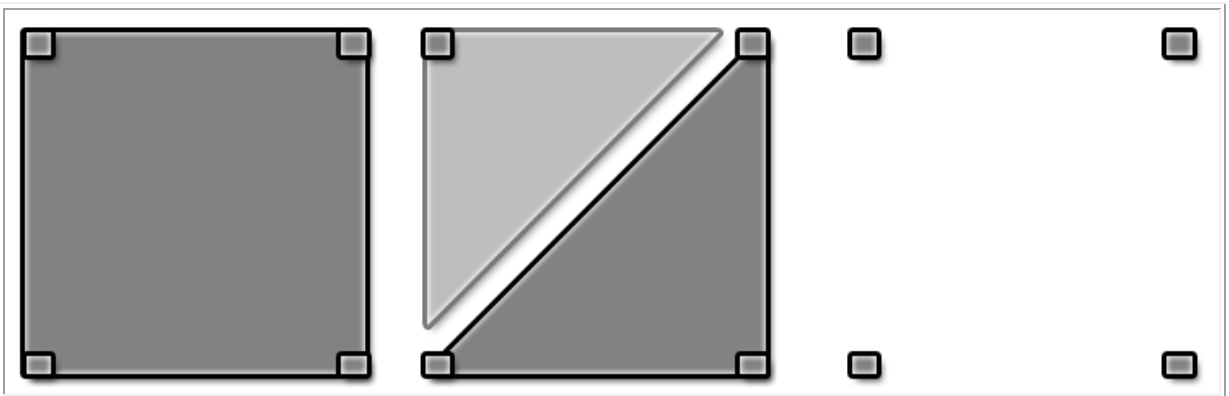

Figure 4.8 Quad, Triangle, Vertex

The image illustrates a polygon, triangle and vertex (Fig 4.8).

- A quad consists of four vertices and two triangles
- A triangle consist of thee vertices
- A vertex is the singular form

Type	Vertices
Vertices	1
Triangle	3
Quad	4
Polygon	5 (or more)

4.2.2 Polygonal Budget

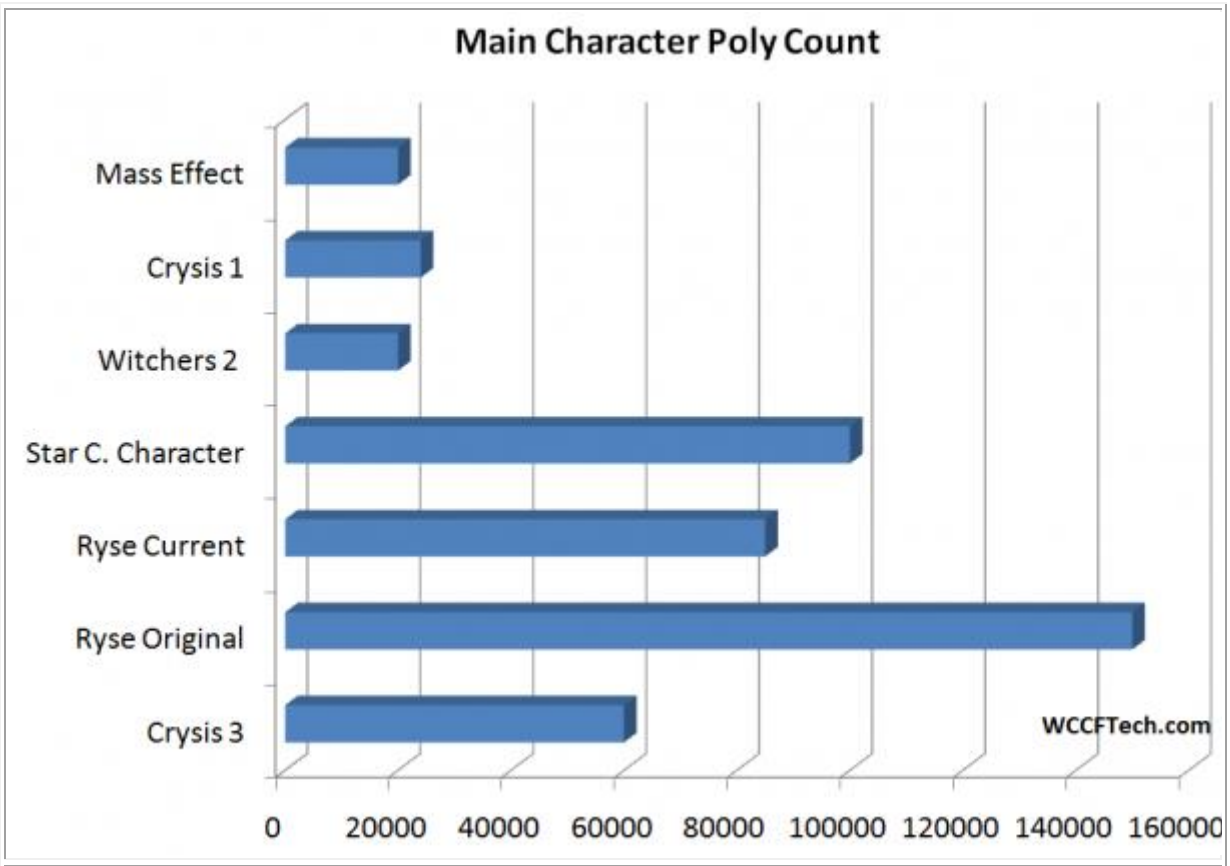

Figure 4.9 Main character poly count

Fig. 4.9 showcases a set of main characters from video games either on the PC, Xbox 360, PS3 or the latest Xbox One console. In the statistics the main character's polycount vary a lot, the determining factor here is the hardware capabilities.

Suggested polygon count for performance (total visible): 7000 vertices
Texture memory: 24MB
Max texture size: 1024px x 1024px
Screen colour depth: 16-bit (65,536 colours)

Picture Format: PNG
Suggested texture size: under 512 x 512

According to Unity's Manual regarding Optimized Graphic Methods: *"The number of polygons you should use depends on the quality you require and the platform you are targeting. For mobile devices, somewhere between 300 and 1500 polygons per mesh will give good results, whereas for desktop platforms the ideal range is about 1500 to 4000. You may need to reduce the polygon count per mesh if the game can have lots of characters onscreen at any given time. As an example, Half Life 2 used 2500-5000 triangles per character. Current AAA games running on the PS3 or Xbox 360 usually have characters with 5000-7000 triangles."* (9)

Polygon budgets are always hard to establish, however in terms of budgeting it may be more helpful to think in percentages. E.g. the environment gets 50% of your polygon budget, characters get 25%, and additional effects will get the remaining 25%.

Area	Polygon Budget
Environments	50%
Characters	25%
Effects	25%

The process of budgeting can also go into greater detail; you can start breaking down those numbers more. The main character might have twice as many polygons as enemy characters, but you might want 10 enemies on screen at once, so that gives roughly 2% per enemy and 5% for lead character.

Area	Polygon Budget
Main Character	5%
Enemy	2%
Non Player Character	2%

Polygons are to be spent where they are needed a general rule of thumb is to give your main character twice the budget of your regular enemy. However, a boss may require greater detail therefor dependent on its size it could use twice the budget of a regular enemy and more closer to the main character.

Area	Polygon Budget
Main Character	5%
Enemy	2%
Boss	5%

A soft limit, which can be used to define the iPhones polygonal budget is 7000 visible vertices per scene, however, the total amount of vertices can go up to number closer to 300000 with the use of LODs, occlusion culling and other various optimization techniques to filter out geometry that is not being used.

Wes McDermott mentions in the book: Creating 3D Game Art for the iPhone with Unity iOS that: *"Your game budget is the blueprint or guide through which your game content is created."* (1, 2011, p. 1)

He also goes on to state that: *"For instance, if you have a fast-paced game concept, you'll need to denote a high frame rate in your game budget such as 30–60 frames per second (fps), and all of the content you create must be optimized to allow the game to meet this frame rate budget."* (1, 2011, p. 1)

In the end, this means that depending on what type of game is to be created, the process of the concepts needs to be evaluated. If it is a game that requires a lot of assets shown at once, then the polygonal budget needs to be much lower for every given asset.

Wes also notes that: "*You might want to give your hero character a bit more resolution in terms of vertex count while reducing the count for the enemies. This budget is subjective to your game requirements, but you can see that without understanding the constraints of the hardware, it would be impossible to create assets that run smoothly on the iPhone and iPad."* (1, 2011, p. 1)

As Wes noted, it all comes down to subjective thought regarding the budget, however the game needs to be tied to the requirements, if the developer does not understand that constraints of the hardware, it would be impossible for the application to run on the device.

4.2.3 Optimizing the use of Triangles

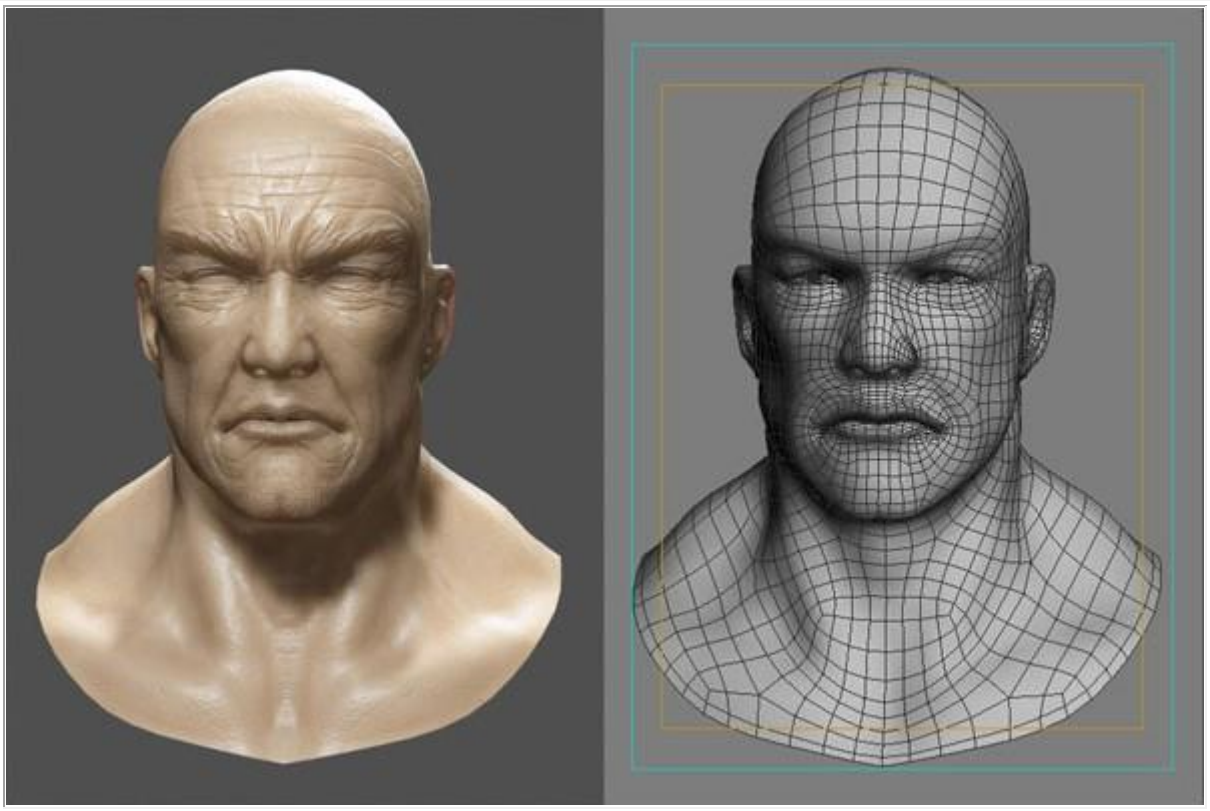

Figure 4.10 Optimizing triangles

The way you create models can have a massive effect on the amount of polygons you end up with, especially when the model is not optimized. As a rule of thumb, every triangle should have an impact on the silhouette and shape of the model; if not then you should probably re-evaluate the structure of the model (Fig. 4.10).

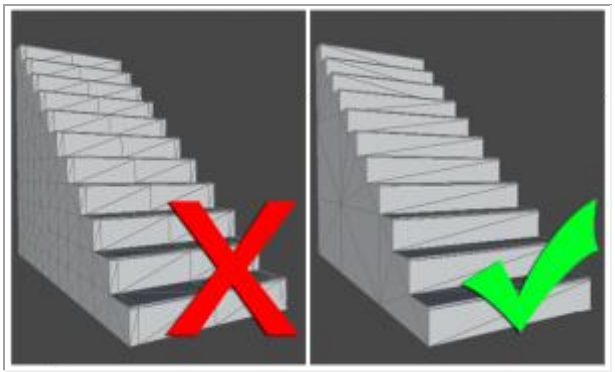

Figure 4.11 Polygonal Triangulation

The model on the right has 726 triangles, while the one of the left has 156.

In Fig. 4.11 the amount of triangles used in this particular model might not seem like a lot, but imagine if this staircase was used a hundred times in a game for measure, if you multiply 156x100 and 726x100 you end up with 15600 and 72600 triangles

In the end this does a lot, every triangle matters, maybe not to the extent where you should have extremely simplistic models, but if it comes to the point where you can take away 500 triangles, and the model looks basically the same, it might be worth considering for the video games performance.

Guillaume Provost 3D graphics programmer at Pseudo Interactive says in the article "Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck" that *"Balancing level complexity and prop complexity is generally the combined job of the game designers and level designers, but as an artist you'll benefit greatly from knowing in which contexts objects are used. Carefully optimizing meshes can be a tedious, thankless task. Making sure you pay attention to those objects that count most first will get you the most out of the system for the least amount of work."* (19)

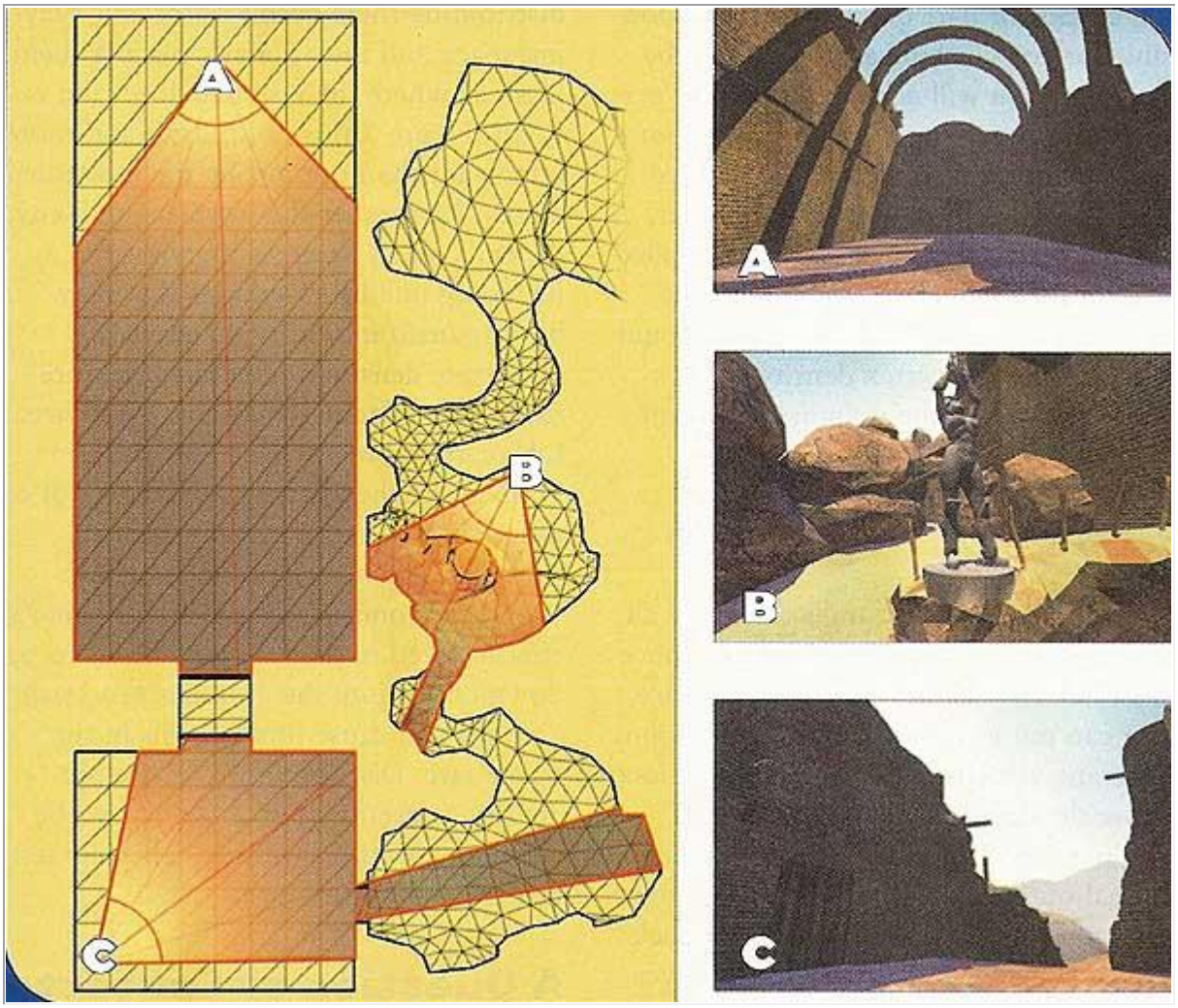

Figure 4.12 Level design in correlation to polygon density

Provost also goes on to add: "*The visibility spectrum in a portal-enabled visibility engine. In camera A and C, the visibility spectrum is closed off using a door that disables the portal. In camera B, transition zones keep the player from seeing into the next area. The very small visibility spectrum in B lets us place a highly detailed statue that we could not afford in the other areas".* (19)

4.2.4 Hard vs. Smooth Edges

Smoothing groups can be applied to various polygons which will help disguise the limited amount of geometry, this is very important when modelling for games to utilize every performance aspect.

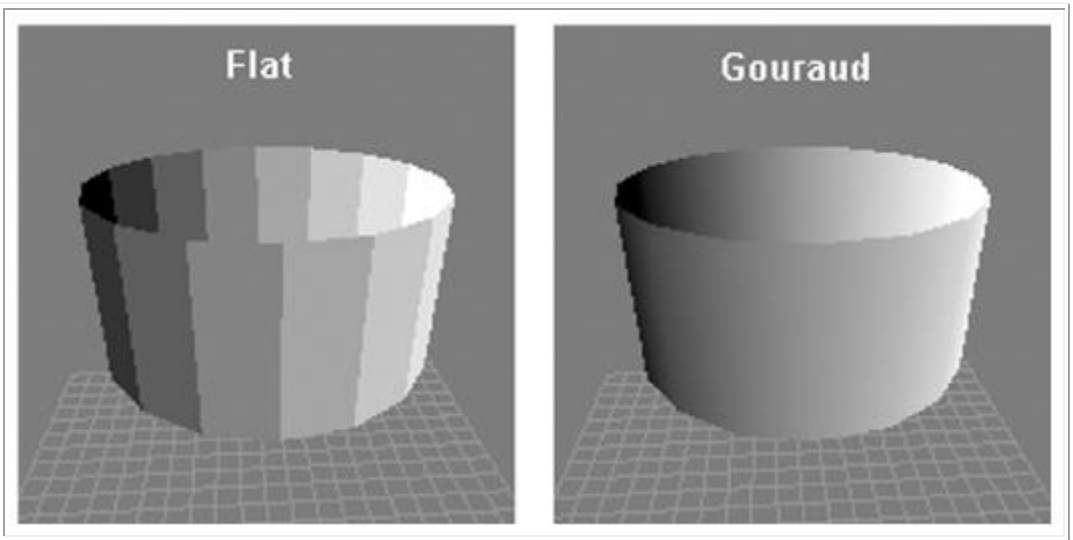

Figure 4.13 Flat shading and Smoothing Groups

Knowing the difference between having a hard edge as compared to a smoothed edge is important in some areas, and when deciding where these boundaries will lie is a determining factor when assigning smoothing groups.

Figure 4.14 Faceted and smooth

The image on the left has faceted edges, and the one the right has smoothed edges.

4.2.5 Level of detail

Level of detail centers around decreasing the geometrical surfaces of 3D models as the viewer moves away from the object, using LODs fluently will enable a real-time application to make the most of all assets.

It is important to use LODs effectively to achieve different levels of detail for different distances. An object that is far as way does not require the same level of detail as an object that is close to the camera, see Fig. 4.15.

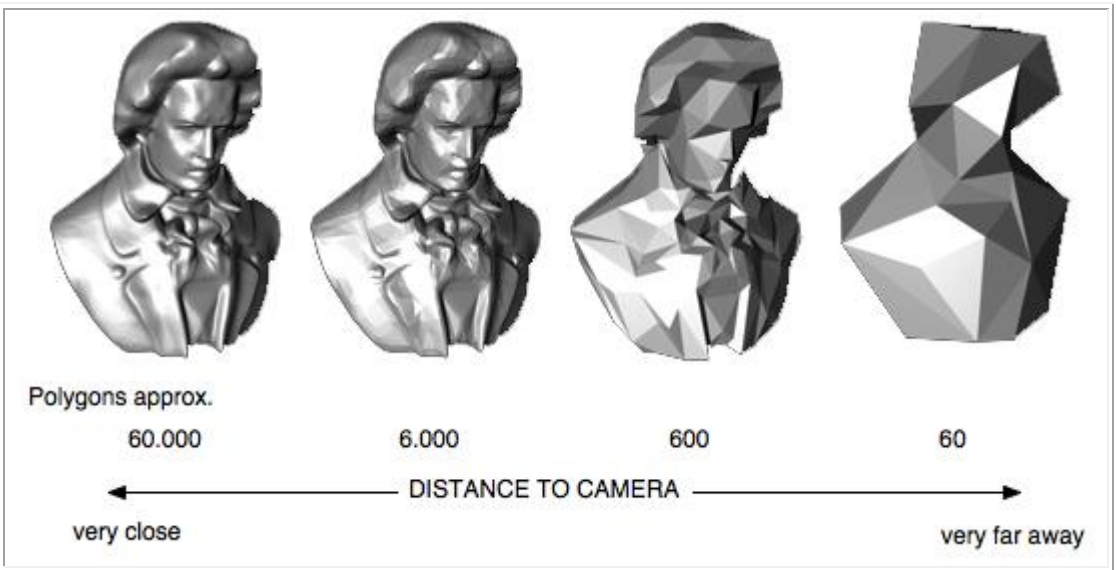

Figure 4.15 Level of detail setup

If the player, or user is not near the object or is very far away, the 3D model will based on the calculations switch to a lower level of detail for optimized performance.

Figure 4.16 Level of detail used on range

When the player moves further away from a specific asset the level of detail will diminish, the LOD transition should be next to impossible to catch.

The LOD enables there to be a more total of 3D models on scene, which is crucial for creating believable environments.

4.3 GPU: Texture

In the iOS developer library they state that: "*Texture data is often the largest portion of the data your app uses to render a frame; textures provide the detail required to present great images to the user. To get the best possible performance out of your app, manage your app's textures carefully.*" (15)

In this section of the analysis, we will analyze what key points should be focused on while deciding on what sort of textures to use for a mobile application and how to use them when opting to have good performance on the mobile device.

Some key points that should be prioritized when working on textures for mobile devices is to reduce the amount of memory on the device which is spent on textures, this can be done through combing smaller textures into a bigger texture atlas.

4.3.1 Basic Texture Overview

Figure 4.17 Basic Texture Overview

There are different texture maps that can be used to portray different qualities of an object, the most common used in a game engines are the following: Diffuse Maps, Local Normal Maps, Specular Maps, Glow Maps and Ambient Occlusion Maps, See Fig. 4.17.

Depending on the complexity of the specific 3D asset in mind, it varies how many are to be used. However, the most common texture is the Diffuse (Colour Map); the diffuse map is a texture which defines a surface's main colour.

The normal map (bump map) is used to predetermine how light reacts to the surface of a model, by giving it fake bumps and groves with high areas being lightly colored and low areas being darker which is controlled by the values of the color used (typically a light violet/blue color). The most common way to create a normal map is to "bake" the details from a high polygon mesh to a low polygon mesh.

However, it is important to note that on mobile devices texture take a big chuck of the GPU's memory, so it is vital to prioritize when choosing which 3D models are given a normal map.

Unity's Manual regarding Optimized Graphic Methods suggests: *"Use it sparingly, only on the most important characters or objects. Anything that can take up the whole screen probably shouldn't be bump mapped."* (9)

This is often the case, and they also recommended to overlaying the normal into the diffuse for objects which do not have the same focus as main characters or objects.

"Instead of using bump maps, bake more detail and contrast into the diffuse texture. The effect from League of Legends is an interesting example of this being used successfully in the industry." – Unity Manual/Practical Guide to Optimization for Mobiles - Graphics Methods (9)

When baking a High resolution mesh to a low poly mesh, see Fig. 4.18. The pipeline typically looks like this:

- Create a high-resolution mesh
- Retopologize the model using the high-resolution sculpt as reference
- UV-unwrap the new low poly mesh
- Bake the normal map
- Apply the normal map as part of the texturing process

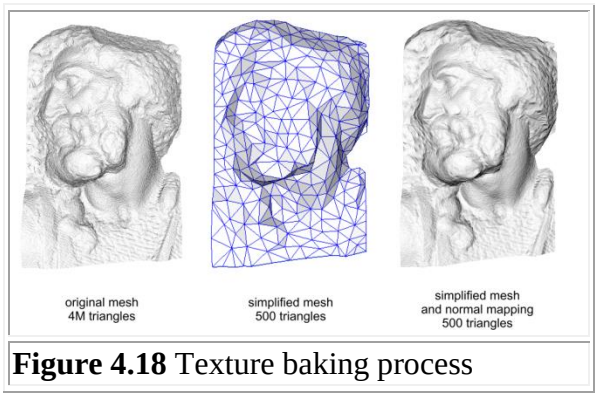

Figure 4.18 Texture baking process

Specular maps are used to define how shiny a surface is of a given object, the higher value of a pixel on the specular (from white to black) will result in a shinier surface. When looking to achieve a dry surface such as stone or fabric, one should try to produce a very dark specular map, while one is trying to achieve a material with similar qualities of chrome or plastic it should be of a lighter contrast.

Unity's Manual regarding Practical Guide to Optimization for Mobile – Graphic Methods suggest when working with specular to overlay them into the diffuse because: *"the human eye doesn't notice that they don't actually line up with the reflected light and view directions - they are simply high-contrast details on the texture, completely faked, yet they end up looking*

great. This is a common cheating technique which has been used in many successful games." (8)

Glow maps are used to make a surface appear like they are emitting a light. Surfaces which are meant to glow should have a strong light vibrant colour, while surfaces which have no such quality should be covered in black.

Ambient Occlusion or the light map creates a soft shadowing without a direct light source; this is useful because it means the complex lighting from a high poly model can be baked to a low polygonal model to save lighting being calculated in real-time.

While all textures that are mentioned above, they all serve a purpose, but due to technical limitations on current generation mobile devices some of them are often simulated by merging textures or prioritized on 3D models which have a focus in the setting. See Fig 4.19, in this case an Ambient Occlusion pass and a diffuse texture is merged to simulate lights in the raw colour.

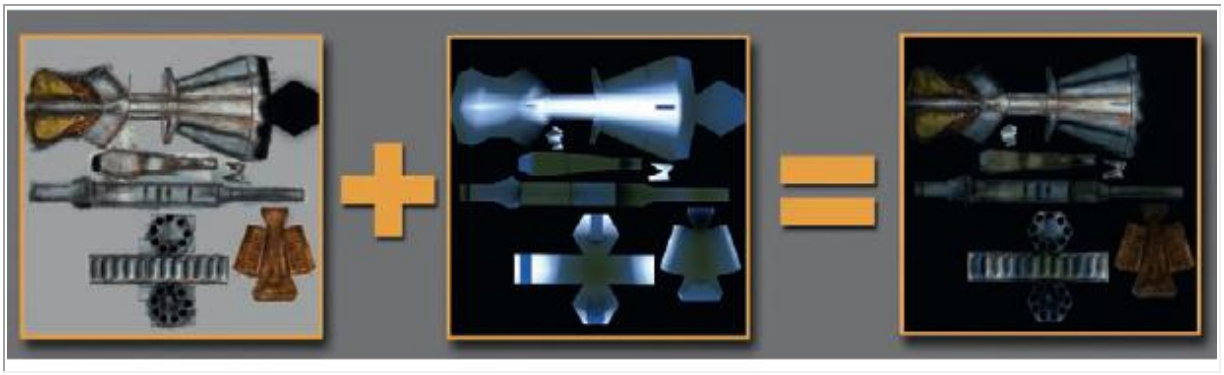

Figure 4.19 Ambient Occlusion and Diffuse Overlay

Wes McDermott suggest: *"Instead of Using a Lightmap Shader, You Could Manually Combine Your Lightmap and Diffuse Texture in Photoshop and Only Use One Texture."*

4.3.2 Compression and Mipmaps

Mipmaps are pre-calculated, optimized collections of images that belong to a main texture; they are a smaller version of the main textures which are used when the textures are displayed in a far of distance. In order to optimize performance Mipmaps are a great way for speed up the rendering time and maintain the texture mapping quality for objects which are far away from the users' camera.

Figure 4.20 No Mipmapping / With Mipmapping

At the website NVIDIA Graphic Visions they use Fig. 4.20 and they explain the Mipmapping procedure: *"Take a look at the Mip mapping example above. The left part is done without mipmapping. You might think that it looks sharper in the screenshot here. It is, but without mipmaps all textures flicker with a large amount of noise - this looks awful when the scene is set in motion. Simplifying the Mipmaps are different versions of one and the same texture that is available in various sizes to fit it in a proper place (depth) inside 3D graphic environment. Just try to imagine that you are standing in a long highway and you are focusing on the road texture beginning from your legs to the all way to the same horizon. To secure as much realistic appearance as it possible."* (25)

This means that Mipmaps allow the texture to be rendered in a very low detail, when the object where the texture is mapped on becomes small. This is valuable because 3D video games often contain massive amounts of textures and information. Therefor having low detail texture; will save performance in the long run.

In Unity's Documentation Manual regarding textures it is stated: *"Using mip maps uses 33% more memory, but not using them can be a huge performance loss. You should always use mipmaps for in-game textures; the only exceptions are textures that will never be minified (e.g. GUI textures)."* (26)

The exchange being that the mipmapped textures require 33% memory is a valuable exchange for being able to have a smoother experience on mobile devices.

Mipmaps main functions are to:

- Speed up rendering times
- Improving the overall quality
- Reducing stress on the GPU

4.3.3 Tileable/Seamless Textures

Seamless or tileable texture are maps which can be placed side-by-side each other without creating a noticeable boundary between two copies of the image.

When a seamless texture is repeated the boundaries of the texture are unnoticed because of the seamless transition, making them extremely valuable for a game engine given the fact that the materials can easily be reused. The seamless textures are typically used in instances such as where a ground, floor or wall pattern is required.

Instead of using a massive texture for the wall which would be 4096x4096 you can use a 512x512 seamless texture and make it tile, in order to save GPU performance.

4.3.4 Vertex Based Painting

Vertex based painting or texture blending can also be combined with the use of seamless texture, which results in being able to create “unique” looks in a 3D world, which also is reasonably cheap. Vertex blending is often used because it is a very efficient way to blend textures without using more texture memory.

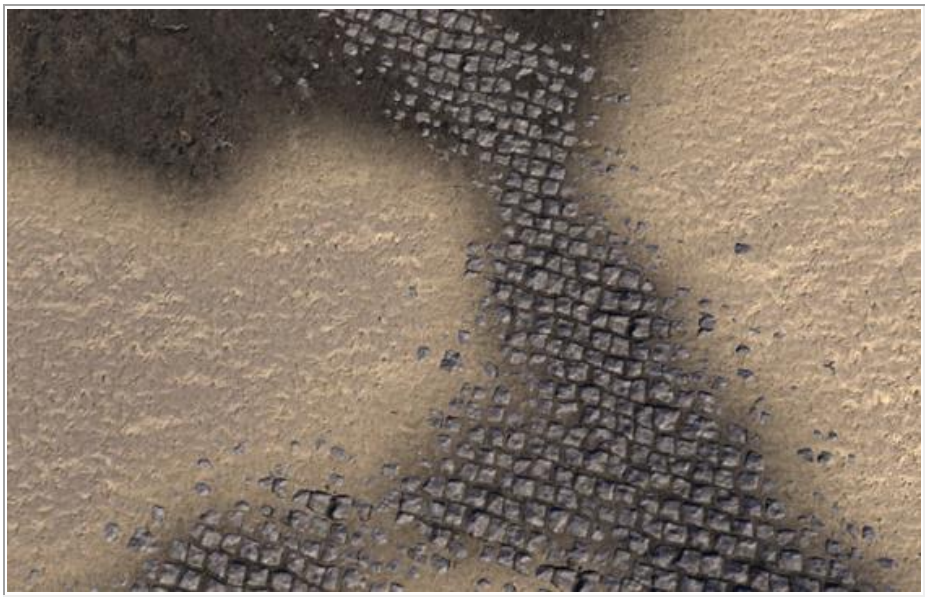

Figure 4.21 Vertex based painting

In Fig. 4.21 vertex based painting is used to create a unique look, by sampling a multitude of different seamless textures.

4.3.5 UV- Mapping

UV- Mapping is the process of making a 2D image representation of a 3D model, this process projects a texture map onto a 3D object by utilizing UV coordinates, as shown in Fig 4.22.

The letters UV were used to describe the 2D coordinates to avoid confusion regarding the X, Y and Z axis which are used to deal with 3D geometry coordinates. When UV Maps are being generated or created, the mapping coordinates are being adjusted and not the geometry itself.

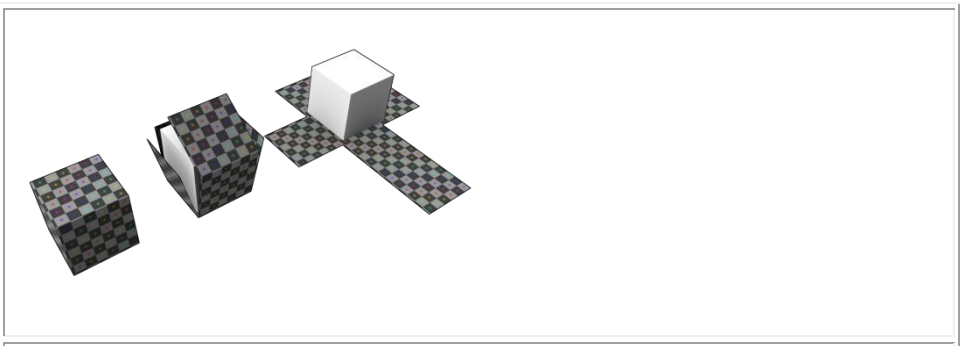

Figure 4.22 Cube representative UV wrapping

UV maps can either be generated automatically by the software application or created manually by the artist, usually the latter is preferred.

3D objects need to have a "UV" maps that specify how a 2D texture is going to be applied on the 3D surface, without a UV Map textures are not useable, see Fig. 4.23.

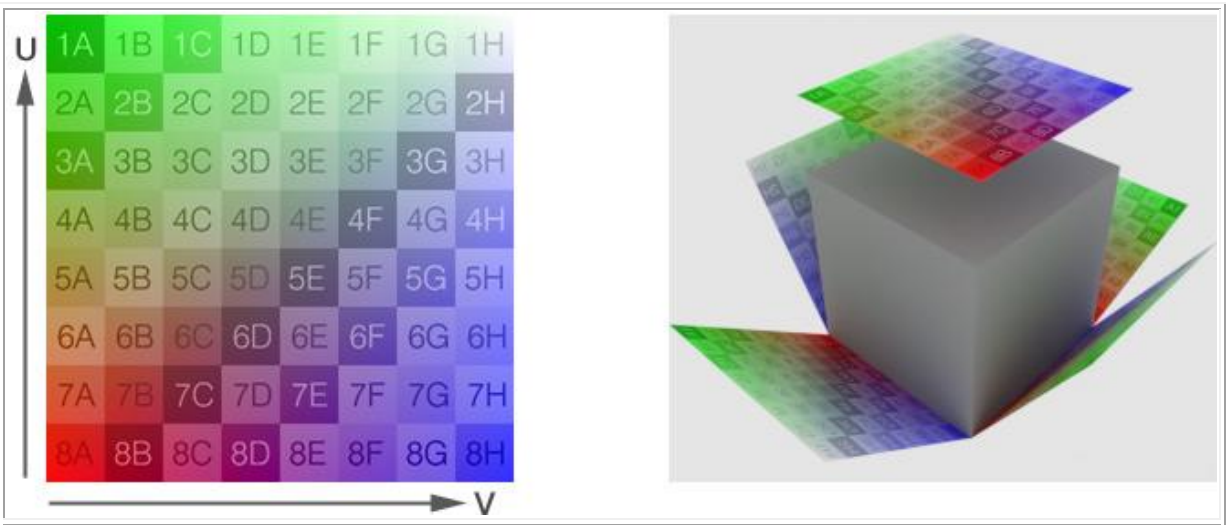

Figure 4.23 Grid UV cube

4.3.6 Optimized UV – Map

The main priority when making an optimized UV map for a 3D asset. As a general rule of thumb while creating UV layout for textures for environment assets:

- Maximize space for the parts you want to show off
- Reuse UVs as much as possible, by mirroring and overlapping
- Seams along natural boundaries, preferably in hidden or less-seen areas
- Use evenly-sized texels to avoid different resolutions across the mesh
- Try to map straight lines either vertical or horizontal
- Remove bleeding by keeping a decent gutter between UVs and along edges.

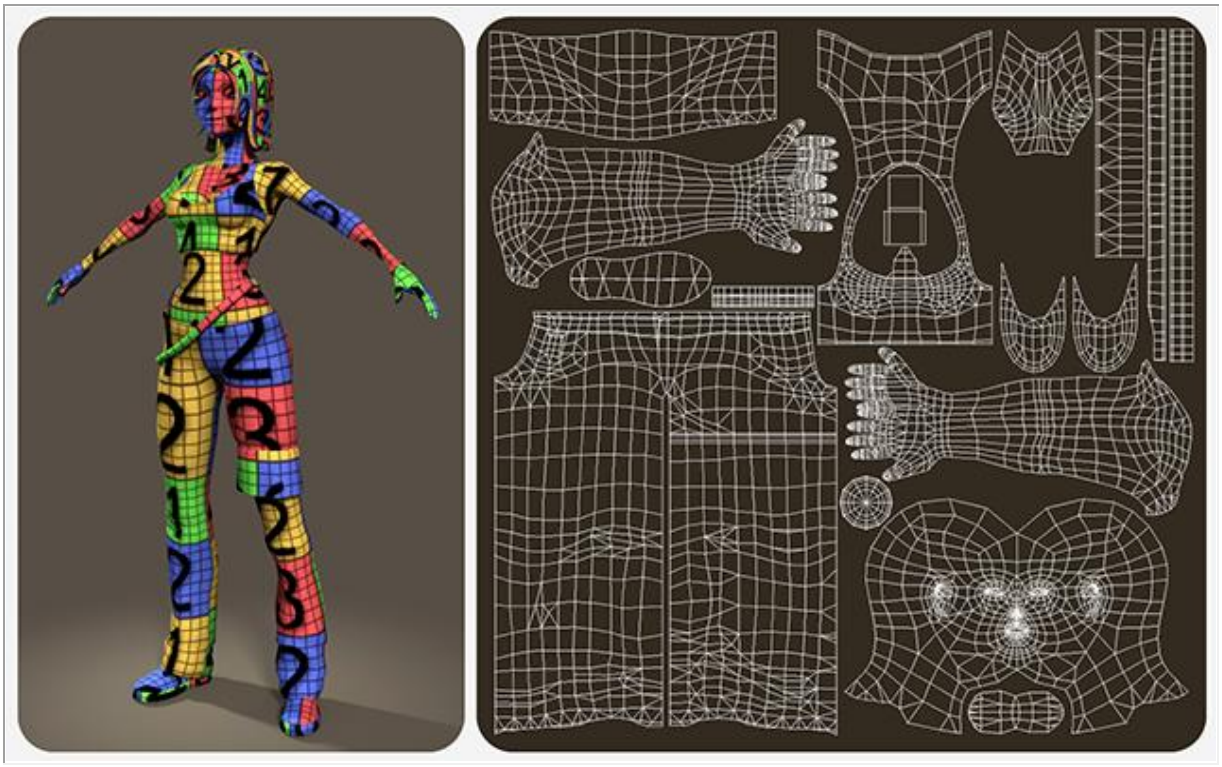

Figure 4.24 UV Map

The main focus while creating an optimized UV map is:

- Use as much UV space as possible
- Reduce as much stretching as much as possible
- Allow ease of texturing and lack of obvious seams.
- Sharing UV-Space between assets

It is crucial to plan ahead when creating UV-maps for 3D assets, depending on what sort of asset you are creating, you will require to approach the map differently, see Fig 4.24 for example of efficiently used in UV-space.

In this example piece I produced for the purpose of showing an example on how to apply Optimizing in terms of creating an optimized UV – Map.

4.3.7 Optimized UV - Layout

Figure 4.25 3D Character

This is an example of a fully utilized UV setup, the face is being sampled for both the right and left side of the face, so is the sword, the texture samples both sides of the blade, the eye is sampled right and left, see Fig. 4.25 and 4.26.

Figure 4.26 Texture Map

4.3.8 Texture Size

The maximum size on Textures on the iOS devices always depend on the hardware. Pre-3GS is 1024, after that is 2048, and in later generations it is capable to use 4096x4096 textures, see Fig. 27.

Keeping texture sizes down will improve the general frame rate, and helps the total build size of the application down. Having compressed textures will make it easier for the device to render scenes; there will be less memory strain, which will make the experience smoother.

It is also important to utilize the size limit, according to "ITunes Connect Developer Guide" the "*iOS App binary files can be as large as 2 GB, but the executable file (app_name.app/app_name) cannot exceed 60MB. However, consider download times when determining your app's size. Minimize the "file's size as much as possible, keeping in mind that there is a 100 MB limit for over-the-air downloads.*" (24)

Sizes	iPhone	iPhone Retina	iPhone 5	iPad	iPad Retina
Devices	iPhone 1g-3GS iPod Touch 1g-3g	iPhone 4, 4S iPod Touch 4g	iPhone 5, 5C, 5S iPod Touch 5g	iPad iPad 2 iPad Mini	iPad Air iPad Mini Retina
Resolution	480 x 320	960 x 640	1136 x 640	1024 x 768	2048 x 1536
% of next size up*	50%	94%	?	50%	100%
% of iPad Retina*	24%	47%	?	50%	100%
% of iPhone*	100%	200%	?	213%	427%
Default file suffix	file.png or file**~iphone**.png	file**@2x**.png or file**@2x~iphone**.png	file**-568h@2x**.png	file**~ipad**.png	file**@2x~ipad**.png
Cocos2D file suffix	file.png	file**-hd**.png	file**-iphone5hd**.png	file**-ipad**.png	file**-ipadhd**.png
Aspect ratio	3:2	3:2	71:40	4:3	4:3
Icon size iOS7(6)[1]	(57 x 57)	120 x 120(114 x 114)	120 x 120(114 x 114)	76 x 76(72 x 72)	152 x 152(144 x 144)
Max texture size[2]	1024 x 1024	2048 x 2048	2048 x 2048	2048 x 2048	4096 x 4096
iOS7 Wallpaper[3]	n/a	1,040 x 1,360	1,040 x 1,536	1,168 x 1,424	1,936 x 2,448

Figure 4.27 iOS resolution quick reference

It should be taken into consideration that optimization should be the main factor and it depends on what the purpose of the specific texture is. The size of the 3D asset should have an appropriate size texture. Another factor is which console generation of the mobile devices is the video game being created for, outdated, current, or future generation.

4.3.9 Texture Atlas

In computer graphics, a texture atlas is the process of composoting smaller images into a atlas texture. The atlas texture contains many smaller sub-images, each of which are part texture for 3D objects.

Atlas texture are essential for creating optmized video game applications, it is often more efficient to store the texture as one instead of a collective of smaller images. The atlas texture, when loaded by the GPU is calcuated as one draw call resulting in the hardware being able to calculate the pre-dertimined textures for the area quicker.

Creating Atlas textures always benefit from using power of two textures (mipmapping) such as “8", "16", "32", "64", "128", "256", "512", "1024", "2048”. They are regarded as being valid and properly optimised for quick loading into a game.

Inside the Unity Manual in the subsection Asset Import and Creation on the topic of Texture 2D they mention that: *Non power of two texture sizes generally take slightly more memory and might be slower to read by the GPU, so for performance it's best to use power of two sizes whenever you can.* (26)

On a mobile device such as the iPhone it is very important to keep draw calls and texture swapping to a minimum. In order to manage materials it is a sensible to combine textures onto texture atlases. Also using UV space optimally is crucial to get the most out of one atlas texture.

It is crucial to plan when creating texture atlases, depending on the size of the particular object and texture, reasonable detail should be equally divided between all objects that are tied to the atlas texture.

4.4 CPU: Optimizing Draw Calls

4.4.1 Material Batching

Unity can combine a number of objects at runtime and draws them together with a single draw call. This operation is called "batching". The more objects Unity can batch together, the better rendering performance (on the CPU side) you can get.

Every time a material is loaded, a draw call set into effect, it is crucial to keep this number down to maintain good performance.

In the book Creating 3D Game Art for the iPhone with Unity iOS written by Wes McDermott he describes a draw call in the following way: *“The draw call can be thought of as a “request” to the GPU to draw the objects in your scene and can be the area in which the CPU causes a bottleneck in performance.”* (1, 2011, p. 3)

3D models which share the same material can be batched together as one, if you want to achieve good batching; you need to share as many materials among different objects as possible.

If you have two identical materials which differ only in textures, you can combine those textures into a single big texture – otherwise known as a texture atlas.

Once textures are in the same atlas, you can use single material instead, which will result in less draw calls and less stress on the CPU.

5. Discussion

In this part of the thesis I will state my interpretations from the analysis, and my findings and explain the implications of my findings, and discuss suggestions for future research in the field of optimized 3d modelling for hand-held devices.

5.1 Murphy's Law

Murphy's law is an epigram that is typically stated as: Anything that can go wrong will go wrong.

Guillaume Provost 3D graphics programmer at Pseudo Interactive mentions in "Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck" that: *"Performance is like laundry: it's a chore, and people only notice it if it's a problem. If your clothes are clean 95 percent of the time, people will only remember -- and judge you by -- those 20 days out of the year where your clothes were not clean."* (19)

This statement often true when it comes to graphics performance, if your game runs smoothly most of the time, but there is one instance where the performance goes downhill because of poorly optimized content, you will be judged by those remaining 5 percent.

Provost also writes: *"Frame rate is a direct function of all objects in view at a given time, and as such it is most likely to go down when the visual and environmental stimuli are at their peak. Since, in most games, that also happens to be both when players are most enjoying themselves and when they require the most responsiveness out of the system, performance hits can be significant sources of player frustration."* (19)

This means that you need to be very careful when building the art assets of video games, getting good performance is more about avoiding the worst case scenarios then making things go faster. Provost also says that you should always: *"always assume players will position themselves in the worst vantage point possible in terms of performance. lf there's a single spot in the entire level the player can sit at and bring all its glorious complexity into view ' at one time, you can rest assured that players will strive to get to it".* (19)

This is something you need to plan and optimize in order to get the most "bang for your buck", if you are to have a whole scene where every asset is to be visible, the assets in scene should have LODs, and be sharing draw calls for the sake of the hardware being able to handle the performance peak.

5.2 Internal Profiler (GPU vs CPU)

Textures and polygons are both inexpensive and expensive, it depends on several factors but as a general rule of thumb neither of the two pools should be flooded with draw calls.

When deciding upon adding larger textures or more vertices, it is the same concept as apples and oranges; they are both drawn from the same pool in a sense.

In the end you do not want the GPU waiting for the CPU, or vice versa. You need to manage your memory pool level creation, both factors need to be taken into consideration and carefully optimized.

In order to know how you game is currently doing it is important to keep an eye on the internal profiler. It will reveal vital statistics about the performance of the video game.

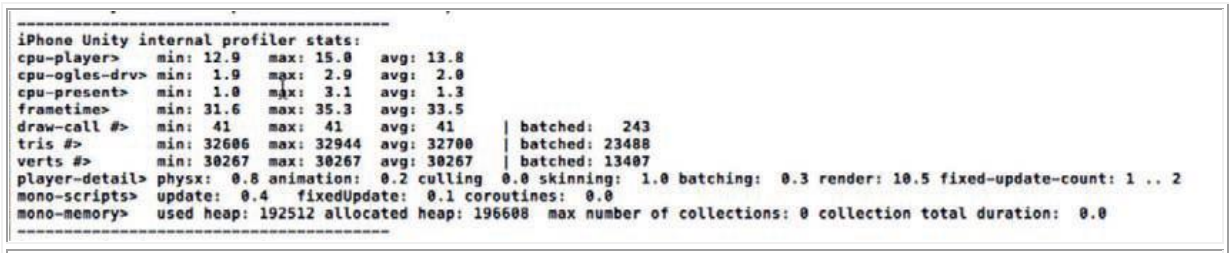

Figure 5.1 Internal Profiler

Inside the Unity Manual in the subsection Measuring Performance with the Built-in Profiler it mentions regarding the internal profiler that: *"These messages are written every 30 seconds and will provide insight into how the game is running. Understanding what these messages mean is not always easy, but as a minimum, you should quickly be able to determine if your game is CPU or GPU bound."* (12)

This will give relevant information to the developer which will help reallocate assets if needed or fix the designated hotspots.

6. Conclusion

Current research appears to validate the idea that 3D models are a large and growing factor in games performance. On the basis of the research it seems fair to suggest that managing the polygonal count and the amount of draw calls will result in a faster frame rate, resulting in a smoother experience.

In Unity's Practical Guide to Optimization for Mobiles, it is mentioned that*: "All mobile devices are not created equal"*, they also go onto add that: "*The computational capability of mobile devices is increasing at an alarming rate. It's not unheard of for a new generation of a mobile GPU to be five times faster than its predecessor. That's fast, when compared to the PC industry.* (8)

In order to produce optimized 3D models for the iPhone it is important to understand that a lot of factors come in to play to, however it is important to note that making optimization a design consideration and not a final step is vital when creating 3D applications for mobile devices.

The hardware used on mobile devices on the market today is very limited compared to the high end computers, which most people are familiar with. This means that the Game creator must respect the limitations of mobile devices in order to create a better game.

Wes McDermott states in the book: *Creating 3D Game Art for the iPhone with Unity that: "Getting to Know the iDevice Hardware and Unity iOS, when dealing with a mobile device such as the iPhone and iPad, you must always be aware of the limited resources available. Like an old miser, constantly counting every penny, you must watch over each vertex and make sure not a one goes to waste."* (1, 2011, p. 19)

In order to produce a working game on a mobile device, it is vital to understand the platform upon which the application is designed to run. Hardware specifications plays a huge part in making a successful video game, as does having a well-balanced game budget in terms of triangle and texture usage. Performance is critical for games, and one of the foundations upon which many games stand, therefore it is vital to make the most out of the devices hardware and software in order to be well-balanced to and for the GPU and CPU.

The analysis provides confirmatory evidence that in order to create optimized content for mobile devices it is advisable to:

- Adjust the polygon count according to the GPU's specifications
- Lower the amount of Draw Calls
- Compress textures
- Texture Priority

Adjusting the polygonal count will help relieve stress on the GPU, lowering the amount of draw calls will optimize the games flow, reducing texture sizes will increase performance and that prioritizing textures for certain key assets will provide for a smoother experience.

There are also three factors that are determined throughout the specifications of the hardware of the iPhone and the content being created for the mobile device, the factors are as follows:

- Performance
- Game Budget
- Marketing

These three factors will determine which modelling budget will be at disposal, if you look at the hardware specifications for GEN1 compared to GEN6, you will notice that GEN1 has 128MB of memory, while GEN6 has 1GB memory which means there is a vast difference in what the devices are capable of.

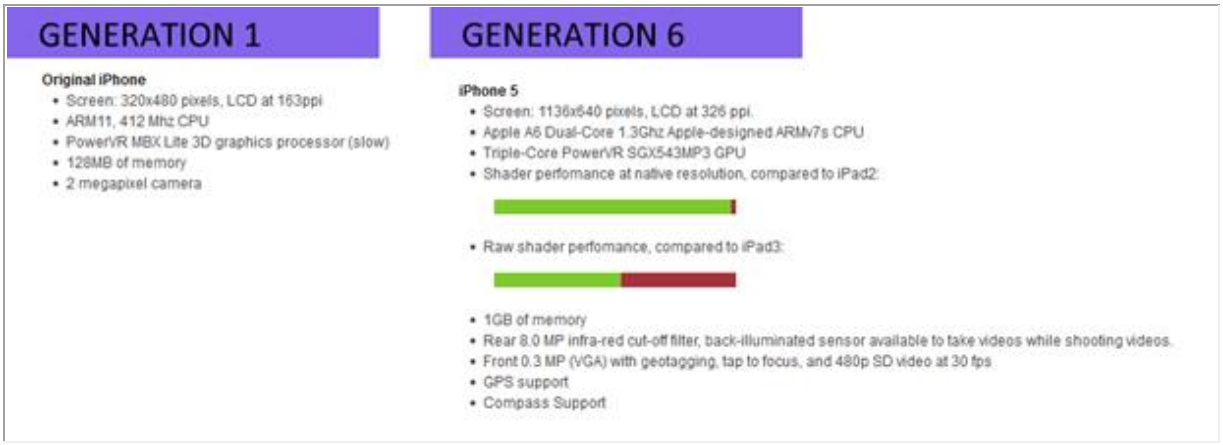

Figure 5.1 iOS hardware comparison

"Don't make your players wait" – Unity's - Practical Guide, Performance [8]

On logical grounds it is worthwhile concluding that: smooth frame rate in games will affect the player experience in a positive way. Given the centrality of the issue of optimizing content for mobile applications; it is important to note that the games will perform better at runtime if reduced graphics bottlenecks are cut down when optimizing is an early design consideration, not as a final step.

The available evidence seems to suggest that when dealing with the limited resources available every performance aspect should be closely looked into when creating 3D content. It is vital to optimize the content and correlate your game budget accordingly to the marketed device. In the end it will make for a better game.

7. References

Literature

(1) McDermott. 2011 Creating 3D Game Art for the iPhone with Unity, Focal Press

(2) Omernick, M., 2004. *Creating the Art of the Game.* New Riders Publishing

(3) Derakshani R. Derakshani D. 2012 3DS Max 2012 Essentials. Sybex

(4) Kennedy. R. 2013 How To Become A Video Game Artist. Watsil –Guptill Publications

Internet

(5) Unity. (2013-07-18). *iOS Hardware Guide.* Available: http://docs.unity3d.com/Documentation/Manual/iphone-Hardware.html. Last accessed 22th May 2014.

(6) Unity. (2011-11-04). *Modeling Optimized Characters.* Available: https://docs.unity3d.com/Documentation/Manual/ModelingOptimizedCharacters.html. Last accessed 21th May 2014.

(7) Unity. (2012-29-07). *Optimizing Graphics Performance.* Available: https://docs.unity3d.com/Documentation/Manual/OptimizingGraphicsPerformance.html. Last accessed 10th May 2014.

(8) Unity. (2012-29-07). *iPhone-Practical Guide.* Available: https://docs.unity3d.com/Documentation/Manual/iphone-PracticalGuide.html. Last accessed 13th May 2014.

(9) Unity. (2012-11-06). *Optmized Graphics Methods.* Available: https://docs.unity3d.com/Documentation/Manual/iphone-OptimizedGraphicsMethods.html. Last accessed 29th April 2014.

(10) Unity. (2012-11-06). *iphone-FutureDevices.* Available: https://docs.unity3d.com/Documentation/Manual/iphone-FutureDevices.html. Last accessed 25th May 2014.

(11) Unity. (2013-07-16). *Practical Rendering Optimizations.* Available: https://docs.unity3d.com/Documentation/Manual/iphone-PracticalRenderingOptimizations.html. Last accessed 13th May 2014.

(12) Unity. (2013-10-10). *iPhone Internal Profiler.* Available: https://docs.unity3d.com/Documentation/Manual/iphone-InternalProfiler.html. Last accessed 17th May 2014.

(13) Unity. (2013-01-31). *DrawCall Batching.* Available: https://docs.unity3d.com/Documentation/Manual/DrawCallBatching.html. Last accessed 20th May 2014.

(14) Unity. (2013-07-04). *Reducing Filesize.* Available: https://docs.unity3d.com/Documentation/Manual/ReducingFilesize.html. Last accessed 21th May 2014.

(15) iOS Developer Library. (03-10-2014). *Guide for iOS.* Available: https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/TechniquesForWorkingWithTextureData/TechniquesForWorkingWithTextureData.html#//apple_ref/doc/uid/TP. Last accessed 22th May 2014.

(16) Matt Wuebbling. (April 20, 2012). Graphics moving toward console level. Available: http://blogs.nvidia.com/blog/2012/04/20/mobile-graphics-moving-toward-console-level/. Last accessed April 25, 2014.

(17) (-). *iOS Resolution Quick Reference.* Available: http://www.iosres.com/. Last accessed 21th May 2014.

(18) Apple. (-). *iPhone Resolution.* Available: http://www.iphoneresolution.com/. Last accessed 21th May 2014.

(19) Guillaume Provost. (June 01, 2003). *Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck.* Available: http://www.ericchadwick.com/examples/provost/byf1.html. Last accessed 21th May 2014.

(20) Guillaume Provost. (July 01, 2003). *Beautiful, Yet Friendly Part 2: Maximizing Efficiency.*
Available: http://www.ericchadwick.com/examples/provost/byf1.html. Last accessed 21th May 2014.

(21) Unity. (February 19, 2013). *Unity games sweep App Store awards.* Available: http://blogs.unity3d.com/2013/02/19/unity-games-sweep-app-store-awards/. Last accessed 21th May 2014.

(22) Editor. (Nov 23, 2012). *Unity 3D: iOS Performance Tips.* Available: http://www.kwalee.com/2012/11/23/unity-3d-ios-performance-tips/. Last accessed 24th May 214.

(23) Splash Damage. (-). *Basic Texture Overview.* Available: http://wiki.splashdamage.com/index.php/Basic_Texture_Overview. Last accessed 24th May 2014.

(24) Apple - iTunes Connect Developer Guide. Available: http://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/iTunesConnect_Guide.pdf. Last accessed 25th May 2014.

(25) NVIDIA Graphic Visions. Available:
http://www.shinvision.com/155
Last accessed 25th May 2014.

(26) Unity. (2007-11-16). *Texture 2D.* Available:
http://docs.unity3d.com/Documentation/Manual/Textures.html
Last accessed 25th May 2014.

(27) Wikipedia (2014-05-15) UV-Mapping
http://docs.unity3d.com/Documentation/Manual/Textures.html
Last accessed 25th May 2014

(28) Wikipedia (2014-05-15) Game Engine
http://en.wikipedia.org/wiki/Game_engine
Last accessed 25th May 2014

(29) Unity (2014-05-15) Unity 3D
http://unity3d.com/
Last accessed 25th May 2014

(30) Wikipedia (2014-03-04) Smartphone
http://en.wikipedia.org/wiki/Smartphone
Last accessed 25th May 2014

(31) NVIDIA (2014-05-25) NVIDIA
http://www.nvidia.com/
Last accessed 25th May 2014

(32) Wikipedia (2014-05-15) System on a chip
http://en.wikipedia.org/wiki/System_on_a_chip
Last accessed 25th May 2014

(33) Apple (2014-05-12) iOS
http://www.apple.com/ios/
Last accessed 25th May 2014

(34) Wikipedia (2014-05-15) Polygon Triangulation
http://en.wikipedia.org/wiki/Polygon_triangulation
Last accessed 25th May 2014

(35) Wikipedia (2014-05-20) Resolution
http://en.wikipedia.org/wiki/Display_resolution
Last accessed 25th May 2014

(36) Wikipedia (2014-05-07) CPU
http://en.wikipedia.org/wiki/Central_processing_unit
Last accessed 25th May 2014

(37) Wikipedia (2014-02-25) GPU
http://en.wikipedia.org/wiki/Graphics_processing_unit
Last accessed 25th May 2014

(38) Wikipedia (2014-04-12) Real-Time
http://en.wikipedia.org/wiki/Real-time_computer_graphics
Last accessed 25th May 2014

Software

Autodesk 3DS Max
Available: http://www.autodesk.com/products/autodesk-3ds-max/overview

Photoshop CS5
Available: http://www.adobe.com/se/products/photoshop.html

NDO2
Available: http://dev.quixel.se/

Unity
Available: https://unity3d.com/

Images

Figure 1.1 Thesis Outline
Unity Logo on Unity. (2014).
Retrieved from https://unity3d.com/
Apple iPhone Logo on Apple. (2014).
Retrieved from https://apple.com

Figure 2.1 Unity App Store 2012
Unity App Store 2012 on Unity. (2013).
Retrieved from: http://blogs.unity3d.com/2013/02/19/unity-games-sweep-app-store-awards/

Figure 2.2 Real-Time Graphics in Video Game The Last Of Us
Rogelio Olguin Environment Artist on The Last Of Us. (2013). Studying.
Retrieved from: http://www.polycount.com/forum/showthread.php?t=122359

Figure 2.3 Game world rendered in 3D geometry
Rogelio Olguin Environment Artist on The Last Of Us. (2013). Studying.
Retrieved from: http://www.polycount.com/forum/showthread.php?t=122359

Figure 4.1 iPhone System-on-a-chip (SOC)
Apple A4 on Wikipedia. (2014).
Retrieved from: http://en.wikipedia.org/wiki/Apple_A4

Figure 4.2 NVIDIA Graphic Chart – Graphics Performance
Matt Wuebbling 2012 on NVIDIA. (2012).

Retrieved from: http://blogs.nvidia.com/blog/2012/04/20/mobile-graphics-moving-toward-console-level/

Figure 4.3 iMac vs. iPhone Comparison
iMac vs. iPhone Comparison. 2012 on iPhone. (2010).
Retrieved from: https://www.flickr.com/photos/x1brett/4742540168/

Figure 4.4 iPhone Resolution
iPhone Resolution on iOS. (2014).
Retrieved from: http://www.iphoneresolution.com/

Figure 4.5 iPhone RAM
iPhone Ram on IOS. (2012).
Retrieved from: http://www.geek.com/apple/iphone-5-vs-iphone-4s-how-the-specs-compare-1515347/

Figure 4.6 Real-time triangulated world
Wireframe on Nexusmods. (2013).
Retrieved from: http://www.nexusmods.com/skyrim/images/143994/?

Figure 4.7 Triangle
3D Triangle. (2013).
Retrieved from:

Figure 4.8 Polygon, Triangle, Vertex
Robin Powell. (2014).

Figure 4.9 Main Character Polycount
Ryse Polygon Count in Comparison with Other AAA Titles. (2014).
Retrieved from: http://wccftech.com/ryse-polygon-count-comparision-aaa-titles-crysis-star-citizen/

Figure 4.10 Optimizing Triangles
3D Total on Wireframe. (2013). Studying.
Retrieved from:
http://www.3dtotal.com/admin/new_cropper/tutorial_content_images/268_tid_fig02.jpg

Figure 4.11 Polygonal Triangulation
Unity App Store 2012 on Unity. (2013). Studying.
Retrieved from: http://blogs.unity3d.com/wp-content/uploads/2011/09/GeomWrongRight-300x179.png

Figure 4.12 Level Design in correlation to Polygon Density
Guillaume Provost on Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck. (2013).
Retrieved from: http://www.ericchadwick.com/examples/provost/byf1.html

Figure 4.13 Flat shading and Gouraud shading (Smoothing Groups)
Gouraud Shading on Wikipedia. (2013).
Retrieved from: http://en.wikipedia.org/wiki/Gouraud_shading

Figure 4.14 Faceted and smooth
Warhammer 40K on Smoothing Groups. (2013). Studying.
Retrieved from: http://forums.relicnews.com/showthread.php?244744-TOOL-Santos-Tools-2-UPDATED-31-01-Scripts-v1-2-2-released!/page3

Figure 4.15 Level of detail setup
Level of Detail on OpenSG. (2014).
Retrieved from: http://www.opensg.org/projects/opensg/wiki/Tutorial/OpenSG2/NodeCores

Figure 4.16 Level of detail used on range
Appropriate distances by James Dargio. (2014).
Retrieved from: http://www.thegnomonworkshop.com/news/2013/03/video-games-film-whats-the-difference-its-still-3d-modeling-right/

Figure 4.17 Basic Texture Overview
Basic Texture Overview, Splash Damage. (2014).
Retrieved from: http://wiki.splashdamage.com/index.php/Basic_Texture_Overview

Figure 4.18 Texture baking process
Normal Map on Wikipedia. (2013).
Retrieved from: http://en.wikipedia.org/wiki/Normal_mapping

Figure 4.19 Lightmap and Diffuse Overlay
Unity Documenation on Unity. (2014). Studying.
Retrieved from: http://docs.unity3d.com/Documentation/Manual/index.html

Figure 4.20 No Mipmapping / With Mipmapping
Unity Documenation on Unity. (2014). Studying.
Retrieved from: http://www.shinvision.com/wp-content/uploads/2009/06/mipmapping-egz.png

Figure 4.21 Vertex Based Painting
Unity Documenation on Unity. (2014).
Retrieved from: http://forum.unity3d.com/threads/168629-Does-anybody-know-how-to-use-ATS-Color-map-Shaders

Figure 4.22 Cube Representative UV Wrapping
UV_Mapping on Wikipedia. (2014).
Retrieved from: http://en.wikipedia.org/wiki/UV_mapping

Figure 4.23 Grid UV Cube
Unity Documenation on Unity. (2014). Studying.
Retrieved from: http://docs.unity3d.com/Documentation/Manual/index.html

Figure 4.24 UV Map
Game Character Creation Series: Kila Chapter 3 – UV Mapping. (2013).
Retrieved from: http://cgi.tutsplus.com/articles/game-character-creation-series-kila-chapter-3-uv-mapping--cg-26754

Figure 4.25 3D Character
Polycount, SDK Goblin. (2010).
Retrieved from: http://www.polycount.com/forum/showthread.php?t=69806

Figure 4.26 Texture Map
Polycount, SDK Goblin. (2010).
Retrieved from: http://www.polycount.com/forum/showthread.php?t=69806

Figure 4.27 iOS Resolution Quick Reference
iOS Resolution. (2014).
Retrieved from: http://www.iosres.com/

Figure 5.1 Internal Profiler
Wes McDermott, Creating 3D Game Art for the iPhone with Unity. (2011).
Retrieved from: Creating 3D Game Art for the iPhone with Unity, page 232.