DEGREE PROJECT

VR Performance for mobile devices

Optimization vs. visual shift against realism

Fredrik Tall

Computer Graphic Arts, bachelor's level
2017

Luleå University of Technology
Department of Arts, Communication and Education

LULEÅ UNIVERSITY OF TECHNOLOGY

Preface

This thesis summarizes my Bachelor's degree in Computer Graphics at Luleå University of Technology in Skellefteå, Sweden.

I would like to take the opportunity to thank my colleagues for feedback and inspiration. A big thank you to LTU and my instructors Samuel Lundsten, Stefan Berglund, Arash Vahdat and Håkan Wallin who made the education possible.

Fredrik Tall

Sammanfattning

Denna rapport kommer att gå igenom prestanda och begränsningar av VR på mobila enheter för att uppnå realism. Det kommer att börja med en teoretisk del som enheten och VR-headsetet, och sedan en praktisk del där prestanda testas för att hitta relevanta kostnader för pixlar ändras i en redan skapad inomhus kontorsmiljö.

Det ultimata målet för VR är Holodeck i Star Trek men utan behov av Starship Enterprise som en stor dator. Använda solglasögon och datorn har alla redan har i fickan, smartmobilen. Tekniken i smartmobiler rör sig fort, komponenter blir mindre och kraftfullare varje år. Det är inte ovanligt att de dubblar sin kapacitet från föregående års modell.
I denna avhandling diskuteras begränsningarna av realistisk VR-grafik och deras kostnader för prestationsbegränsade enheter som finns i smartmobiler. Resultaten visar visuell effekt kontra prestanda i en kontorsmiljö skapad för Unity, den kommer att täcka...

- Vilka optimeringar i VR skapar mer autentiska pixlar, jämfört med deras kostnad i FPS när du måste ta hänsyn till en mycket mer limiterad prestationsbudget?

- Vilka parametrar är viktiga att arbeta med för att komma närmare en trovärdig rekreation av en verklig världsmiljö jämfört med kostnaden i FPS.

Abstract

This report will go through the performance and the limitations of VR on mobile devices for achieving realism. It will start off with a theoretical part such as the device and VR headset, and then a practical part where performance is tested to find relevant cost for pixels changed in an already created indoor office environment.

The ultimate goal of VR is the Holodeck in Star Trek but without the need for the Starship Enterprise as a number cruncher. But to use a slick pair of sunglasses and with the computer everyone already has in their pocket, the smartphone. With phones and technology moving as fast as it is and components getting smaller and more powerful every year, it is no longer impossible to think that it's achievable.
In this thesis we will be discussing the limitations of realistic VR graphics and their costs for performance limited devices that exist on today's phones. The results will show the visual effect versus performance in an office environment created for Unity, it will cover…

- What optimizations in VR creates more authentic pixels versus their cost in FPS when you have to consider a much stricter performance budget?

- What parameters are important to work with to get closer to a credible recreation of a real world environment versus the cost in FPS.

Table of Contents

1 INTRODUCTION

1.1 Background

Creating a realistic VR game takes a lot more computing power than regular games. The basics are that VR games has two cameras, one for each eye. This means that the game essentially needs to be run twice, one game for each eye. People are sensitive to micro delays or latency which are common in games when calculating a lot of information 30 times a second. To avoid getting nauseous the game has to update three times as fast as regular games.[1] Also, since almost nothing can be constructed in VR with a certain viewpoint or distance in mind, every 3D object has to be detailed from every angle and up close.

The data below illustrates the amount of pixels per second that has to be calculated before it can be seen on the screen. A newly released AAA game on the PlayStation4 for example, Dark Souls III is running at 1080p at 30 FPS. That means that at least 62 million pixels per second are processed.

- 720p at 30 FPS: **27** million pixels/sec
- 1080p at 60 FPS: **124** million pixels/sec
- 4k Monitor 4096x2160 at 30 FPS: **265** million pixels/sec
- VR 1512x1680x2 at 90 FPS: **457** million pixels/sec

Pixels/sec = Width * Height * FPS[2]

1.2 Question Formulation

Optimization in VR versus cost in FPS when you have to consider a much stricter performance budget and which parameters are the most important for closing the gap for recreating a realistic environment?

1.3 Purpose

The purpose of this thesis is to experiment with an office environment for testing VR performance and judge graphical parameters. This thesis should:

- Correlate between FPS and pixels changed on visual changes.
- Investigate an objective way of preforming tests for the office environment in a VR environment.

---

[1] Timothy J. B., Dennis A. V., and John E. D. (2012) *"The Effect of Apparent Latency on Simulator Sickness While Using a See-Through Helmet-Mounted Display: Reducing Apparent Latency with Predictive Compensation"* Human Factors and Ergonomics Society

[2] Vlachos, A. (2015) *"Advanced VR Rendering" VALVE*

1.4 Limitations

- This thesis won't go into the technical aspects of shader writing or other ways of optimizing the real-time rendering pipeline directly.
- The office environment has already been created in Unity for mobile devices. I will not change the visual integrity of the underlying 3D models or textures.

Programming the real-time pipeline or editing the underlying shaders, models and textures will result in a production centered thesis. Focused on either the programming or producing 3d assets for a VR application/game.

2 Theory

2.1 Virtual Reality

Coined in 1985 by Jaron Lanier, the term denotes *"The use of computers and human computer interfaces to create the effect of a three-dimensional world containing interactive objects with a strong sense of three-dimensional presence"* - (S. Bryson, 1996)

Compared to desktop computer systems, virtual reality facilitates a high level of immersion into virtual space. This is supported by the means of special display systems covering a large portion of the user's visual field of view and interactions devices sensing the user's physicals movements. Virtual reality allows the user to step into situations, which would be expensive, dangerous or even impossible to experience in the real world such as the recently opened 'VR Zone Project I Can' in Tokyo.

2.1.1 Headset

The most basic components of a VR headset are two lenses and a casing to hold them, the more expensive models also have two screens built in with adjustable distance relationships from lens to lens, lens to eyes and finally lens to their respective screens. They also have different ways of tracking the headsets movement.

One of the first commercially available headsets was the Forte VFX1 which was announced at CES in 1994. The VFX-1 had stereoscopic displays, 3-axis head-tracking and stereo headphones.[3]

The HTC Vive is the top of the line, and one of the most expensive ones but also the one that is pushing the technology furthest with its room scale tracking.

Oculus Rift is also on the cutting edge with the same type of technology but limited tracking is by rotation from a fixed position.

Gear VR is a harness designed for smartphones, it utilizes the phone as the computer and screen.

Google Cardboard is the most inexpensive and basic headset that exists, it is designed for smartphones. It only has two glass lenses, folded pieces of cardboard, double sided tape to hold it together and Velcro to mount your phone in place making it act like a screen to display the VR content.

[3] *Nathan Cochrane (1994). "VFX-1 VIRTUAL REALITY HELMET by Forte".* Game Bytes Magazine.

Figure 2:1 **HTC Vive**[I]

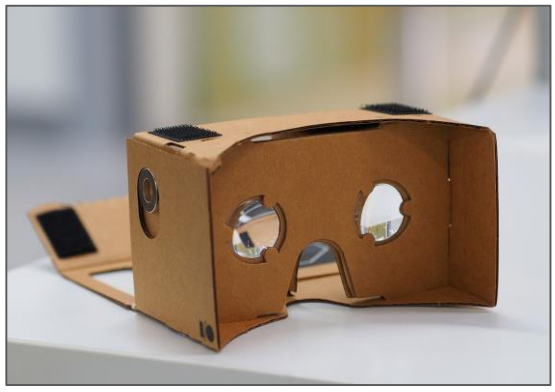

Figure 2:2 **Google Cardboard**[II]

If the alignment of the real and virtual world is done at such fidelity that our senses are unable to perceive any visual differences, it's possible to feel present in the virtual world. This sense of presence is the goal of many VR technologies. However, it is very hard to achieve presence and very easy to break it.[4]

For example, if the user touches something in the real world that is not present or properly aligned in the virtual world there is an obvious mismatch between our sense of touch and vision. How the brain interprets information when there is a mismatch between senses depends on how reliable the information is perceived to be. In most cases our sense of touch and proprioception is deemed more reliable than vision. So if you sit down and feel the floor with your feet about 1.2m below your eyes, your brain will not trust visual information telling it that the eyes are 1.7m above the virtual floor. In an attempt to match the differing information it might instead assume that everything at eye level is 1.2m above ground, resulting in the virtual world being interpreted as smaller than it actually is.[5]

Mismatch of our sensory information have a more severe effect than making us perceive the world around us at an incorrect scale, it can also result in nausea.[6]

A common case is when your head moves and the image of the display doesn't follow that movement. When this happens the visual information lags behind causing motion sickness. The brain's own feedback loop between these sensory systems is very fast, which in turn demands a stable high frame rate and low latency on the device. To mitigate the effect of motion sickness the user's perception of the virtual environment should match the user's other senses.

[4] Martijn J. Schumie et al. *"Research on Presence in Virtual Reality: A Survey".*
[5] Tom Forsyth. *"Connect: Developing VR Experiences with the Oculus Rift".*
[6] Eugenia M. Kolasinski. *"Simulator Sickness in Virtual Environments".*

2.1.2 Virtual reality sickness or Cybersickness

Cybersickness is a term to denote a motion sickness like symptoms when exposed to a virtual environment. The most common symptoms are: [7]

- General discomfort
- Headache
- Nausea
- Vomiting
- Sweating
- Fatigue
- Drowsiness
- Disorientation
- Loss of balance

VR sickness occurs when the refresh rate of on-screen images is not high enough. Because the refresh rate is slower than the brain's processing speed, it causes a discord between the processing rate and the refresh rate, which causes the user to perceive glitches on the screen. When these two components do match up, it can cause the user to experience the same feelings as simulator and motion sickness, as mentioned above.[14]

The expected quality on animation can also cause users to experience this phenomenon. When animations do not meet the users' expectations, it causes another type of discord between what is expected and what is actually happening on the screen. When onscreen graphics do not keep the pace with the users' head movements, it can trigger a form of motion sickness.[14]

Another trigger of virtual reality sickness is when there is disparity in apparent motion between the visual and vestibular stimuli. Essentially what happens is there is a disagreement between what the stimuli from the eyes send to the brain and what the stimuli from the inner ear are sending to the brain. This is what is essentially at the heart of both simulator and motion sickness. In virtual reality, the eyes transmit that the person is running and jumping through a dimension. However, the ears transmit that no movement is occurring and that the body is sitting still. This type of discord between the eyes and ears cause motion sickness.[8]

[7] Laviola, J. J. Jr (2000). *"A discussion of Cybersickness in virtual environments".* ACM SIGCHI
[8] Kolasinski, E. M. (1995). *"Simulator sickness in virtual environments".* U.S. Army Research Institute for the Behavioral and Social Sciences

2.2 Smartphone performance

Since the introduction of the smartphone, mobile graphics technology has advanced considerably. By 2007 mobile graphics technology took off, and it is expected to close the performance gap with previous console generation in the next few years.[9]

That said, comparing a current smartphone and a current console is like comparing a Ferrari with a Smart Car. A console is much bigger and purpose built for generating computer graphics and a smartphone is not.[4]

In figure 3.3 we can see the lines of consoles (gray) and mobile (dark green) are getting closer together and is moving past last-gen consoles over the last few years. The iPhone 6 was released in September 2014 and can be compared to half of a PlayStation 2. In April 2015, less than half a year later, Samsung's new Galaxy S6 was released and is five times as powerful as the PlayStation 2.[10]

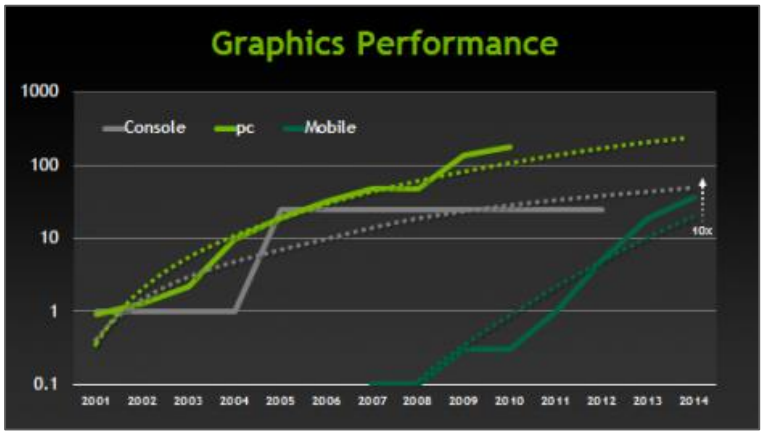

Figure 2:3 NVIDIA Graphics performance chart[III]

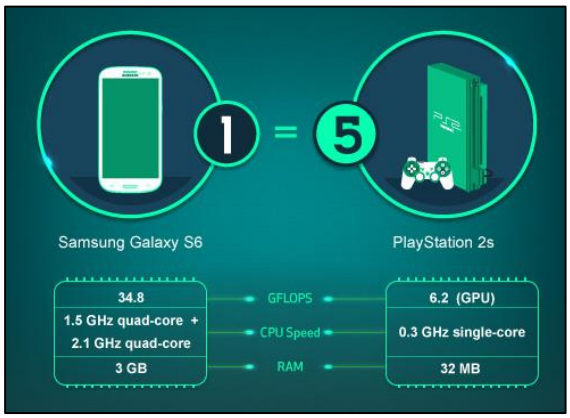

Figure 2:4 Processing power compared[IV]

[9] Matt Wuebbling (2012). *"Mobile graphics moving toward console level".* NVIDIA

[10] Experts Exchange (2015). *"Processing power compared - Visualizing a 1 trillion-fold increase in computer performance".* Experts Exchange

2.2.1 Limitations

There are three main factors to consider when it comes to GPU and CPU performance: The proficiency of the chip designer, the size of the chip, and the chip's maximum power consumption or thermal design power, TDP for short.

If you have two identical designed CPUs with 1 billion transistors, but one chip has a 5W TDP and the other has a 20W TDP, it's safe to assume that the 20W TDP chip will be significantly more powerful. The same goes for the size of the chips, one CPU with 2 billion transistors is going to be a lot more powerful than one with 1 billion transistors. This is a simplification that ignores other important factors, but simply put; Bigger is better.[4]

2.2.2 CPU & GPU

The physical processor cores inside smartphones are on par with that of a PlayStation 4 or Xbox One. An iPhone 6 for example has enough physical space for two cores, that are run on 3-4W from a phone battery that is commonly charged once a day. The CPU has to be run at a speed that won't generate too much heat since there is no real cooling of the components inside the phone.

The PS4 on the other hand has eight cores that are driven by 50 times (140W) that of the iPhone 6 and the power is provided by your wall outlet, which provide an unlimited amount of electricity to the system. Cooling is installed in the system to prevent the hardware from overheating. [9.1&9.2]

All in all, this means that faster cores equal higher power consumption, which mean higher hardware temperatures that in turn requires cooling. With zero cooling, limited size and power, a new smartphone is yet to stand a chance to a new console.

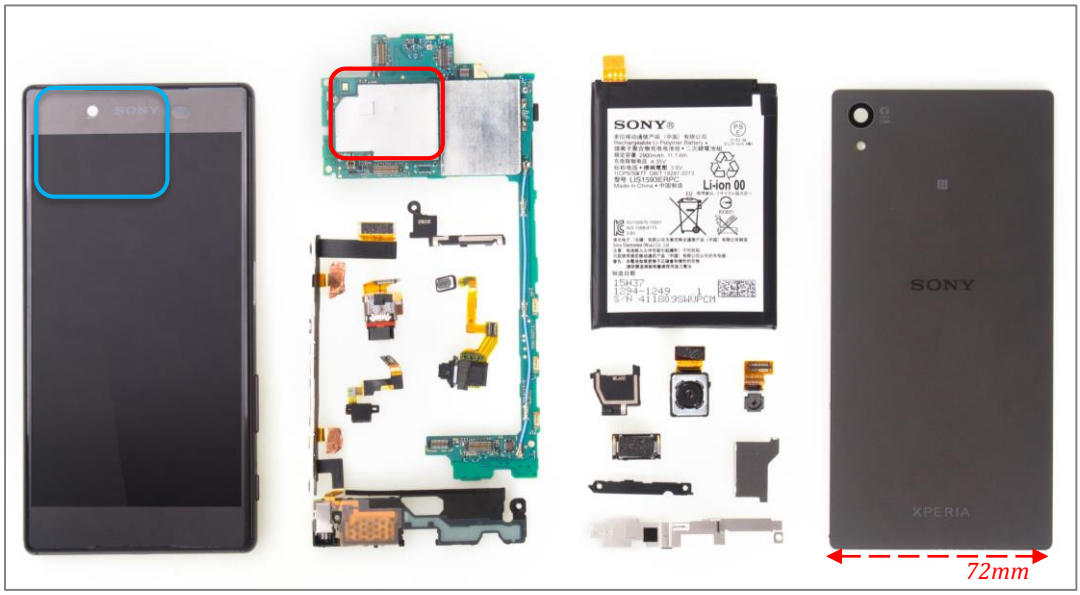

Figure 2:5 Sony Xperia Z5 teardown[V]

- Cooler for the APU - Two small heat pipes on the back, that transfer heat away from the chip
- APU - Octa-core (4x1.5 GHz & 4x2.0 GHz) CPU and Adreno 430 GPU

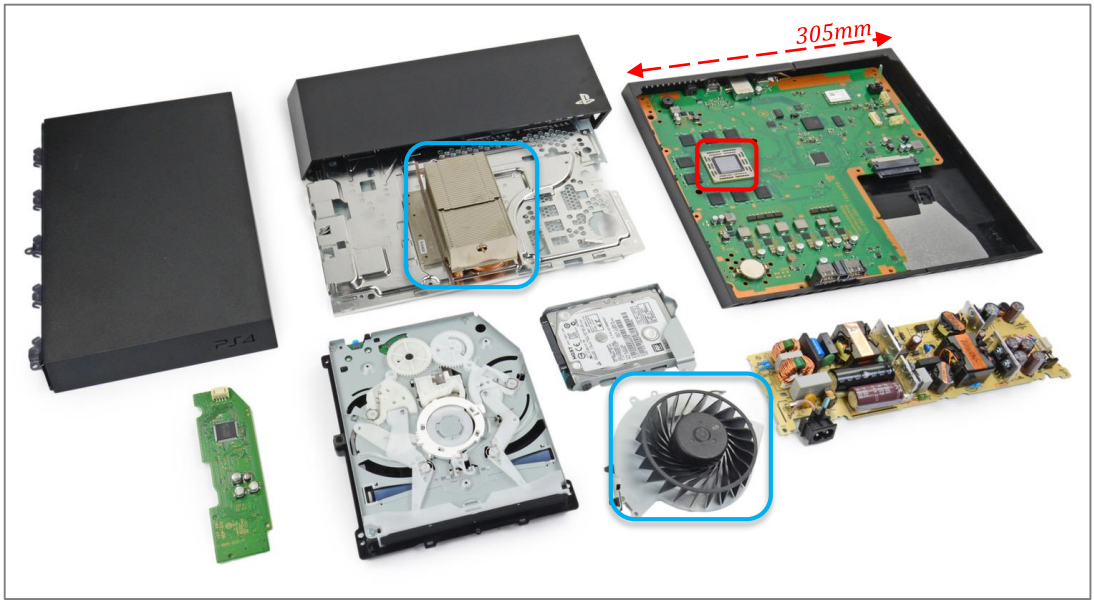

Figure 2:6 **PlayStation 4 teardown**[VI]

- Heat sink for the APU – transfers all the heat up away from the chip onto a wide surface area and the fan blows the hot air out
- APU - 2 Quad-core AMD "Jaguar" Cores 1.6 GHz CPU and AMD Radeon Graphics GPU

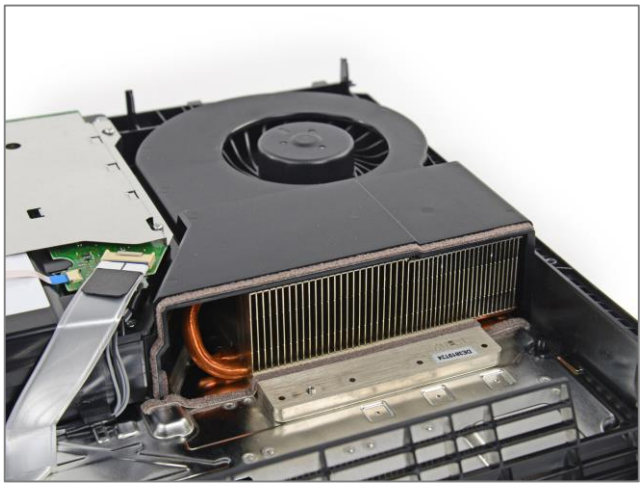

Figure 2:7 **PlayStation 4 APU cooler**[VII]

[11] Sebastian Anthony (2014). *"Does the iPhone 6 actually have console-quality graphics?".*
[9.1] Joel Hruska (2013). *"Reverse engineered PS4 APU reveals the console's real CPU and GPU specs".*
[9.2] Sebastian Anthony (2014). *"Apple's A8 SoC analyzed: The iPhone 6 chip is a 2-billion-transistor 20nm monster".*

2.3 Frame rate

The rate at which images are displayed is measured in frames per second (FPS) or Hertz (Hz). At one frame per second there is little sense of interactivity; the user is painfully aware of the arrival of each new image. At around 6 FPS, a sense of interactivity starts to grow. An application displaying at 15 FPS is certainly real-time; the user focuses on action and reaction.[10]

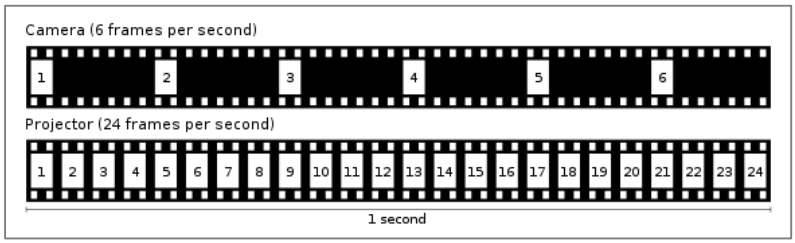

Figure 2:8 Illustration of FPS[VIII]

Barfield (1998) studied the perceived level of presence within a virtual environment as a function of input device type and update rate. Although the type of input device had limited effect, it was found that an update rate of at least 15 FPS was a critical value in order to experience a sense of presence. Regarding ease and comfort of navigation, interactivity and smoothness of motion, an update rate of 15 FPS was considered equally important. The study further revealed consistently higher ratings for all factors when the update rate was increased to 20 FPS.[12.1]

In a more recent study, Claypool (2007) investigated the impact of frame rate on player performance in first person shooter games. For movement tasks it was found that an increase of frame rate from 7 FPS to 15 FPS significantly improved player performance. Unfortunately, no data were collected beyond 15 FPS. For shooting tasks the frame rate was found to be even more important. Although the rate of improvement was steeper below 15 FPS, player performance was increased all the way up to 60 FPS.[13]

Looking at the computer games industry it becomes clear that 15 FPS is not enough to satisfy player demands. For so-called first-person shooter games 30 FPS is generally considered the absolute minimum, and many game developers even target 60 FPS in order to give players a smooth and responsive experience.[14]

---

[10] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". AK PETERS
[12.1] Barfield, W., Baird, K.M., Bjorneseth, O. J. (1998). *"Presence in Virtual Environments as a function of type of input device and display update rate".*
[13] K. T. Claypool (2007). *"On frame rate and player performance in first person shooter games".*
[14] Rubino, C., & Power, J. (2008). *"Level design optimization guidelines for game artists using the epic games: Unreal editor and unreal engine 2".* Computers in Entertainment (CIE)

2.4 3D Graphics Rendering Pipeline

The main function of the pipeline is to generate or render a two-dimensional image. The rendering pipeline is the underlying tool for real-time rendering. The locations and shapes of the objects in an image are determined by their geometry, the characteristics of the environment, and the placement of the camera in that environment. The appearance of the objects is affected by material properties, light sources, textures and shading models.[15]

A coarse division of the real-time rendering pipeline can be divided into three conceptual stages: Application, geometry, and rasterizer. These make up the core of the pipeline. Each of these stages may be a pipeline in itself.[16]
Also the slowest of the pipeline stages determines the rendering speed or the update rate of the images, FPS or Hz as mentioned above.

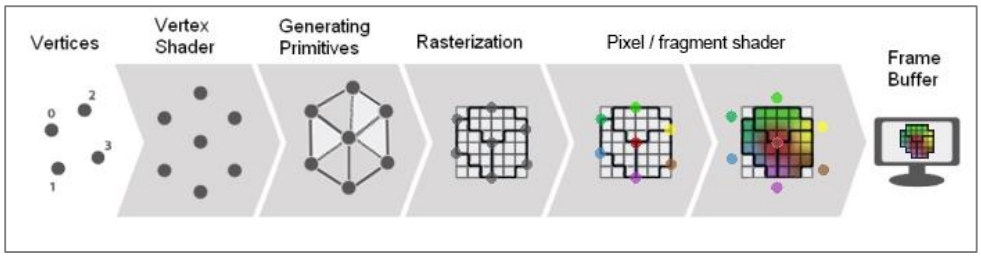

Figure 2:9 Simplified 3D graphics rendering pipeline

The Application Stage

Tasks performed by the application stage include collision detection, physics simulation and many others, such as controls and configures future stages of the pipeline like blending, clipping, etc.

The application stage is driven by a program and is therefore implemented in software such as DirectX or OpenGL running on the CPU. A consequence of the software-based implementation is that it is not divided into substages like geometry and rasterizer stages.
The most important tasks of the application stage are to download huge lists of vertex information to the GPU, and then bind that data together.

Geometric data or vertex buffers, i.e., points, lines and triangles are stored in every draw call. A draw call separates all objects in the scene into manageable chunks, one object per draw call. Imagine dividing every object to a new projector slide, and to view the whole image all slides are stacked in front of the projector.[17]

---

[15] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 11-14 AK PETERS
[16] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 12 AK PETERS
[17] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 11-15 AK PETERS

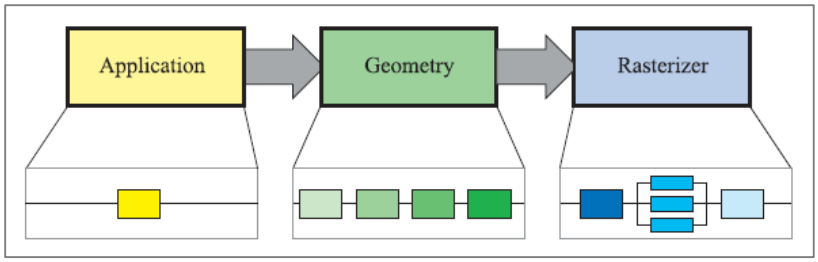

Figure 2:10 **Basic construction of the rendering pipeline**[IX]

The Geometry Stage

To produce a realistic scene, it is not sufficient to render the shape and position of objects, their appearance must be modeled as well. The geometry stage is responsible for the majority of the per- polygon and vertex operations. This stage is generally divided into the following stages: Model and view transform, vertex shading, projection, clipping and screen mapping.

Model and view transform are different coordinate spaces. By moving everything from one space to another, clipping and projection operations are simpler and faster to calculate.[18]

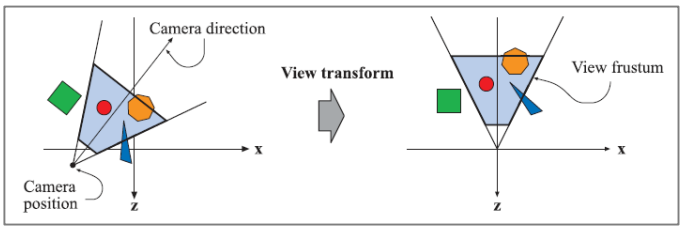

Figure 2:11 **Moved into the view transform coordinate space**[X]

The vertex shading operation handles the processing of individual vertices such as their position and the effect light has on its material, also known as shading. Vertex shading results which can be colors, vectors, texture coordinates are then sent to the rasterization stage.
A variety of material data can be stored in vertex. The point's location, a normal, a color or any other numerical information that is needed to compute the shading equation.[19]

Projection is performed to define what we want to view. Objects within or partially within the volume will be rendered and objects outside of it won't. There are many ways to project a 3D volume onto a 2D screen. Two commonly used methods are orthographic- and perspective projection. Projection transforms the view volume into a 1x1x1 unit cube and depending on which method is used, either the parallel lines converge at the horizon. Making objects appear smaller the farther away from the camera it lies, mimicking the way we

[18] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 16-17 AK PETERS
[19] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 17 AK PETERS

perceive objects' sizes in real life. Or lines remain straight removing the illusion of distance no matter how far apart the objects are.[20]

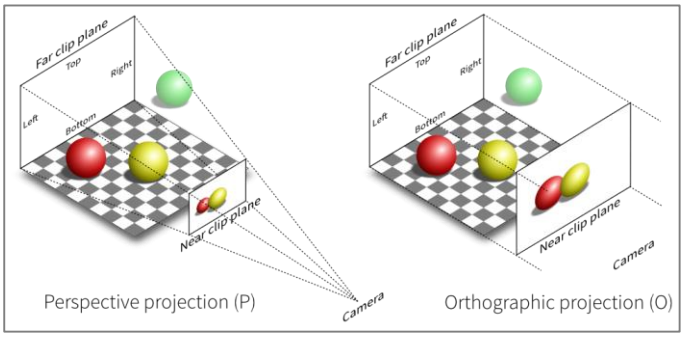

Figure 2:12 **The yellow and red spheres are within the volume while the green one is not and does not appear on the projection[XI]**

The final stages in the geometry stage are concerned with preparing the geometric shapes for rasterization, the final stage. This is done in two steps:

1. Clipping is performed so that only the primitives entirely or partially inside the view volume need to be passed on to the rasterizer stage, which then draws them on the screen. A primitive that lies completely inside the view volume is passed on just as is and primitives that lie outside are not. Primitives that are partially inside the view volume require clipping.

   For example, a line or triangle such as in figure 3.12 that has one vertex outside and one inside the view volume should be clipped against the view volume, so that the vertex that is outside is replaced by a new vertex that is located at the intersection between the line and the view volume. All vertices outside will be discarded.[21]

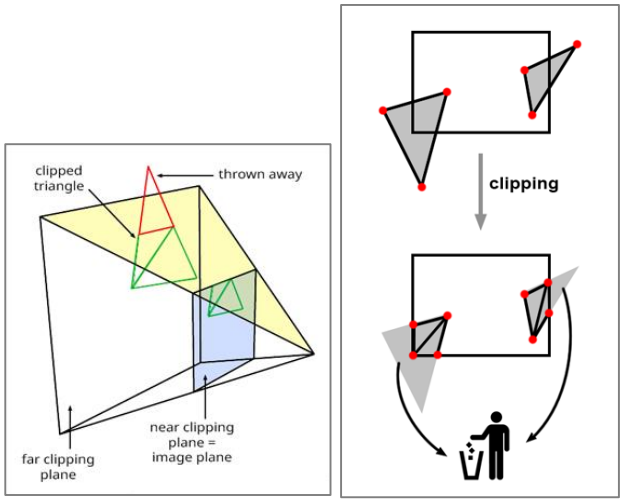

Figure 2:13 & 2:14 **Primitives intersecting with the unit cube or view frustum are clipped and new vertices are generated, all outside are discarded[XII&XIII]**

[20] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 18 AK PETERS
[21] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 19 AK PETERS

2. After clipping has occurred the primitives are sent to the screen mapping stage. Here the unit cube's and primitives x- and y-coordinates are mapped to the pixel dimensions, the width and height of the viewport or screen.

   The new coordinates called screen coordinates and the unchanged z-coordinates are passed on to the rasterizer stage.[22]

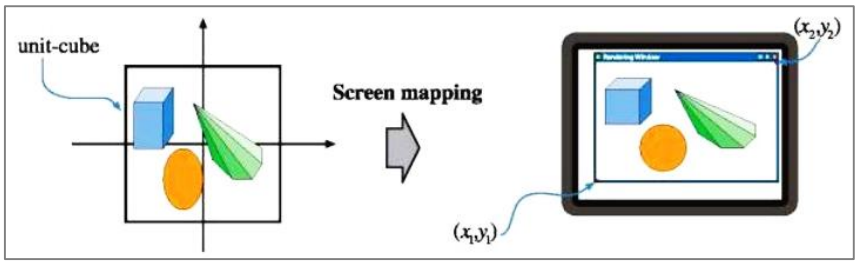

Figure 2:15 The primitives lie in the unit cube after the projection transform, and the screen mapping procedure takes care of finding the coordinates on the screen[XIV]

The Rasterizer Stage

Similar to the geometry stage, this stage is divided into the following stages:

- Rasterization
- Pixel shading
- Merging

In this stage, all primitives are rasterized, i.e., converted into pixels on the screen. Each visible line and triangles in each object enters the rasterizer in screen space, ready to convert. Those triangles that have been associated with a texture are rendered with that texture applied to them. Visibility is resolved via the z-buffer, along with alpha or transparent triangles. Each object or draw call is processed alone and the final image is then displayed on the screen.[23]

Figure 2:16 & 2:17 Grab a draw call that is projected and ready for the rasterization stage. The blue slide is a draw call that previously been through the rasterizer stage[XV&XVI]
***Note that a single draw call is processed through the entire pipeline, and then per-pixel is checked against the previous call if there is a change in the pixel.**

[22] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 20-21 AK PETERS
[23] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 21-26 AK PETERS

Rasterization checks whether a pixel overlaps the triangle, if it does; save that pixel. All other pixels are ignored and do not need to be calculated. This is not whole truth, because checking all pixels of the entire screen is too expensive when we'll only want to check a small area of the image. By grabbing the lowest and maximum values of the triangle, also known as a bounding box, only pixels within that area are calculated.[24]

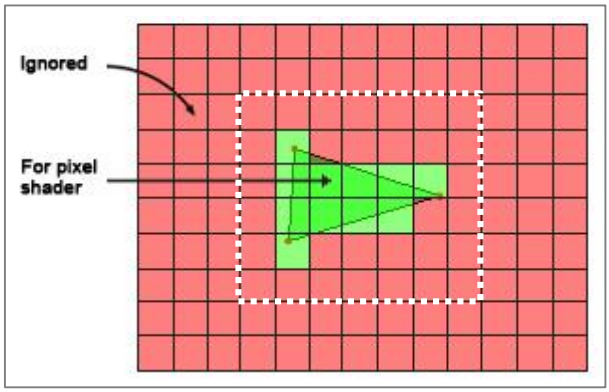

Figure 2:18 White box representing the bounding box of the triangle to optimize the calculation.

Pixel or fragment shading is a program that gets every pixel from the rasterizer in a list, each pixel can be seen as a train track for the rest of the pipeline. That means from here on out each pixel will be handled the same way our triangles were, every pixel is independent of each other.
Imagine the amount of pixels sent through the pipeline as train tracks, in figure 3:19 the pixel shader was sent 11 pixels that represent the triangle from the rasterizer stage. This means 11 train tracks that will be calculated simultaneously, all trains will leave the station at the same time. Fortunately, the GPU excels at handling a lot of smaller tasks.

The program can be programmed in a lot of ways to create different post-processing effects, such as creating real-time ambient occlusion, reflections, anti-aliasing. But its primary use is for texturing, coloring pixels by looking up the colors on the UV coordinates in the vertices.[25]

Post processing effects

The process of applying full-screen filters and effects to a camera's image buffer before it is displayed to screen. It can drastically improve the visuals of your product with little setup time.

---

[24] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 21-37 AK PETERS and J. C. Prunier (2015) *"Rasterization: a Practical Implementation".* Scratchpixel
[25] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 21-37 AK PETERS

Real-time ambient occlusion for example is a simple rectangle plane placed in front of the camera, the pixel shader is programmed to color pixels black and white depending on their distance to each other by looking up the z-depth data. Adding directional information provides even more accurate results with little computational cost.[26]

Anti-aliasing by super-sampling means that each full frame is rendered at double (2x) or quadruple (4x) the display resolution, and is then down-sampled to match the display resolution. A 2x FSAA would render 4 super-sampled pixels for each single pixel of each frame. Rendering at larger resolutions will produce better results; however, more processor power is needed, which degrade performance and frame rate.[27]

You can use post-processing effects to simulate physical camera and film properties too, such as; Bloom, Depth of Field, Chromatic Aberration or Color Grading.

[26] Mittring, M. (2007) *"Finding next gen: Cryengine 2. In SIGGRAPH '07: ACM SIGGRAPH".* P. 97–121 New York, NY, USA. ACM
[27] Gregory J., Lander J. (2009) *"Game Engine Architecture".* P. 39 AK PETERS

Figure 2:19 To the left is the pixels sent from the rasterizer. To the right is the pixels colored after the pixel shader.

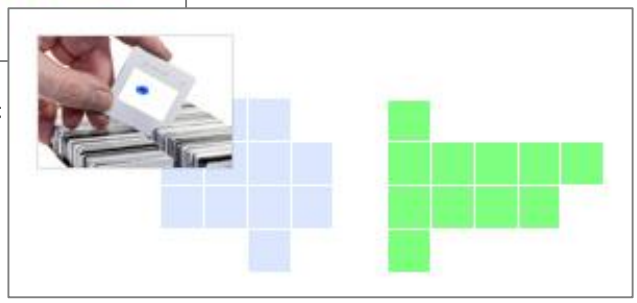

Figure 2:20 To the left is the current pixels as they are, but the entire list of objects(right) is not treated fully yet. So the pixels have to be checked with the Z-depth to determine which pixel should be drawn, blue or green.

Merging is the next station on all 11 track trains, and as the name suggests, it merges each new draw call's pixels. It also checks the z-buffer to see which pixels are overlapping another, not necessarily the order in which they are in the list from the application stage.[28]

In figure 2:20 the left illustration is the current pixels which we are merging with the new from the triangle. Merging checks with the z-depth data which pixel is closest to the camera, either the blue or green should be on top based on the 3D model's location that we want to render. The closest pixel data will be rendered visible, in this case it is the green pixel.

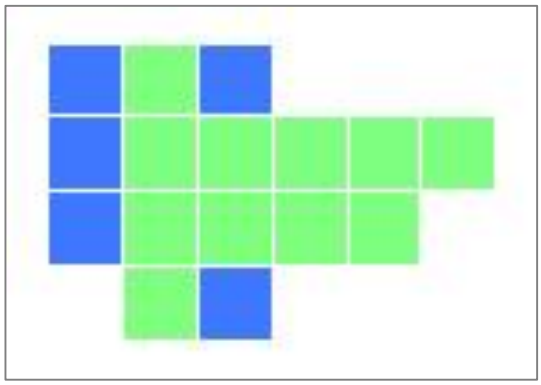

Figure 2:21 Final result displayed to the screen.

[28] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 21-37 AK PETERS and J. C. Prunier (2015) *"Rasterization: a Practical Implementation".* Scratchpixel

2.5 Optimization

The graphical parts of your game can primarily impact two systems of the computer: the GPU and the CPU. The first rule of any optimization is to find where the performance problem is, because solutions for optimizing for GPU vs. CPU are handled differently.[29]

The general idea behind pipeline optimization is to remove the bottlenecks without reducing the possible amount of geometry to process. For example, it is often the case that an application is CPU-bound as opposed to GPU-bound. What that means is that the GPU is waiting for the CPU to feed it with new data and commands. In essence, the GPU becomes underutilized.[30] The work that must be carried out by the CPU in order to render a frame is divided into three categories:[31]

- Determining what must be drawn
- Preparing command for the GPU
- Sending command for the GPU

In order to reduce draw calls in CPU-bound situations the main way is geometry batching. By batching the idea is to combine geometry that share the same state, for example materials, in order to form a few large "chunks" of geometry to render. By doing so the same amount of geometry can be rendered, but with much fewer draw calls.[32]

GPU performance is often limited by a fill rate, especially on mobile devices, but memory bandwidth and vertex processing can also be concerns. Fill rate refers to the number of pixels render per second by the GPU. If our game is limited by fill rate, this means that our game is trying to draw more pixels per frame than the GPU can handle.

The pixel shader that tells the GPU how to draw a single pixel calculates a lot of data that might not be needed. A pixel shader can be programmed with that in mind, for example ignoring all cast shadows.
Fill rate can be solved by lowering the rendered resolution, this however has huge visual impact.
If our game is limited by memory bandwidth, this usually means that we are using textures that are too large for the GPU to process at the required speed.

Vertex processing refers to the work that the GPU must do to render each vertex in a mesh. The cost of vertex processing is impacted by two factors: the number

---

[29] Unity Technologies. (2017). *"Optimizing graphics performance".* Unity manual
[30] Hillaire, S. (2012). *"Improving Performance by Reducing Calls to the Driver".* OpenGL Insights, A K Peters
[31] Unity Technologies. (2017). *"Optimizing graphics rendering in Unity games".* Unity learning platform
[32] Wloka, M. (2003). *"Batch, Batch, Batch: What Does It Really Mean?"*. Presentation at Game Developers Conference 2003

of vertices that must be rendered, and the number of operations that must be performed on each vertex.[33]

2.6 Modelling

Modelling is the action of creating a mesh of geometry in a virtual three dimensional space with a 3D-modelling software. Models are created from polygons, multi-sided planes that are connected to each other to make a form. A simple 3D-object, for example the cube in figure 2:22 is comprised of vertices, edges and faces. Vertices are points in space, the corner of the cubes. An edge is the line which connects two points. Faces are the surfaces that connects to four points.

A single vertex holds more than x-, y- and z-coordinate data; Normal direction is one that is important for the visual quality of the model. Each vertex has three normal directions to describe shading. A softened normal direction creates a smoother surface and a hardened normal direction creates a sharper surface.[34]

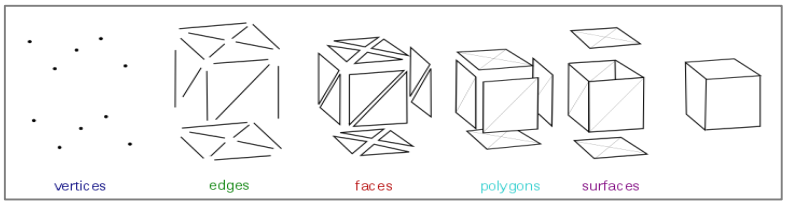

Figure 2:22 Illustration of a cube and its components[XVII]

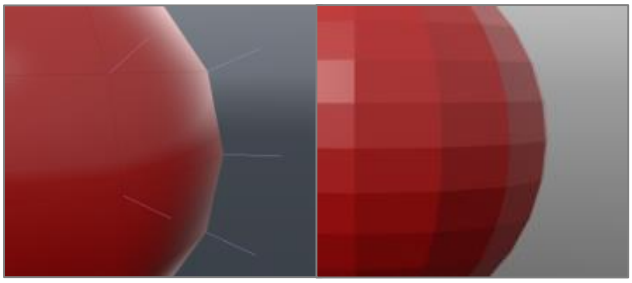

Figure 2:23 Illustration of a sphere with soft- and hardened normal directions[XVIII]

A 3D-model starts as a primitive shape and through stages of refinement is made into an object.[35] There are different modelling techniques. Box modelling, image-based modelling and digital sculpting are some. Polygons should be spent on refining the geometry's outer edges, because shading can fake the impression of a detailed surface but not a silhouette.[36]

---

[33] Unity Technologies. (2017). *"Optimizing graphics rendering in Unity games".* Unity learning platform

[34] *Kennedy, S. M. (2013). "How to become a video game artist: The insider's guide to landing a job in the gaming industry". Watson-Guptill Publications.*

[35] *Solarski, C. (2012). "Sponsored feature: Drawing basics and video game art: Character design". Gamasutra*

[36] *McGrath, T. (2008). "Creating efficient next gen environment art". Gamasutra*

2.7 Texturing

As explained in the paragraph Real-Time Rendering, "A surface's texture is its look and feel", e.g. the texture of an oil painting.

Textures are expensive in video games. They are of large file size, and each object usually require four or more to create a material. Atlas maps are a combination of more textures into one big file, meaning the computer only has to load and process one file to color multiple objects all will with the same material properties. The downside to using atlas maps is the increased file size required to not lose texture quality of our objects.[37]

[37] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed*". P. 147 AK PETERS

3 Methodology

3.1.1 Data collection

This thesis was based on data collection through the following:

1. Literature and thesis paper studies on 3D Game Art & Android Devices
2. Unity's documentation (User Manual and Learning platform)[38]
3. Visual comparisons between the real-time rendered office environment and the real-life photograph provided
4. Experiments with Unity's available graphic settings to compare the FPS with pixels changed in the rendered image

Figure 3:1 Real-time rendered office.

Figure 3:2 Photograph of real office.

[38] Unity Technologies. (2017). *"Optimizing graphics performance".* Unity manual

The tests are running on the device in VR as shown in figure 3:3. However all future images of the environment will be shown though a camera placed at the same location as shown in figure 3:1. The FPS is captured from the device running the office environment in VR.

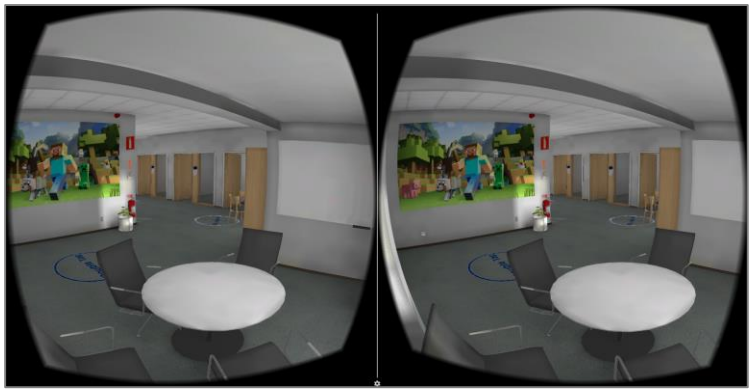

Figure 3:3 VR point of view.

VR tests were made on a Xperia Z2 with the Google Cardboard headset. Tests will be broken down into two sections, visual (rasterizer stage) and nonvisual changes (application and geometry stages). Visual changes will be compared by subtracting one result from the other. The change to VR was made with Google Cardboard's Unity package.

Statistics from the original (Non-VR) office environment:

Objects	Triangle count	Vertex count	*Materials	*Textures	Lightmaps
126	13200	19800	22	13	13

*Materials: 4 standard- and 18 mobile optimized materials
*Textures: 1 detail atlas map and 12 individual maps

This study draws research from Unity's own guide and sources referenced by Unity staff "Unity's-Practical Guide to Optimization for Mobiles", articles such as, "Beautiful, Yet Friendly Part 1: Stop Hitting the Bottleneck", "Beautiful, Yet Friendly Part 2: Maximizing Efficiency".[39]

Methods of optimization that will be analyzed are:

- Draw call batching
- Triangle count
- Vertex count
- Texture quality
- Materials
- Lighting
- Post-effects

The aim of this will be to achieve some insight regarding the production of optimized content for handheld real-time applications and apply them to VR.

[39] Unity Technologies. (2017). *"Optimizing graphics performance".* Unity manual and E. Chadwick (2003) *"Beautiful, Yet Friendly Part 1 & 2: Stop Hitting the Bottleneck & Maximizing Efficiency".*

3.2 Draw call batching

This is a nonvisual change to the office environment. There are two methods which are used to optimize the draw calls, and as they relate to arranging the objects into sets or groups to be rendered together.

Static batching was used to combine non-moving objects into a single model. Models in which have the same material inside Unity are automatically separated into different draw calls. By knowing this combine the second method, material batching, by exporting static objects which share materials together.

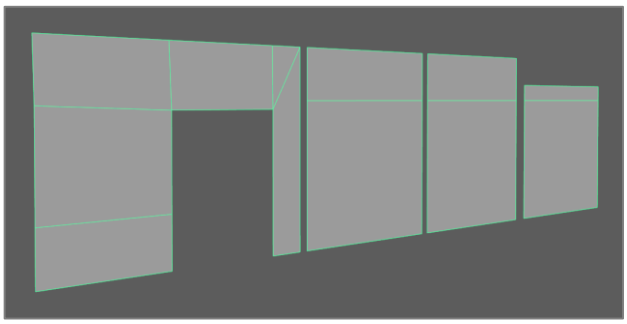

Figure 3:4 Four combined wall models.

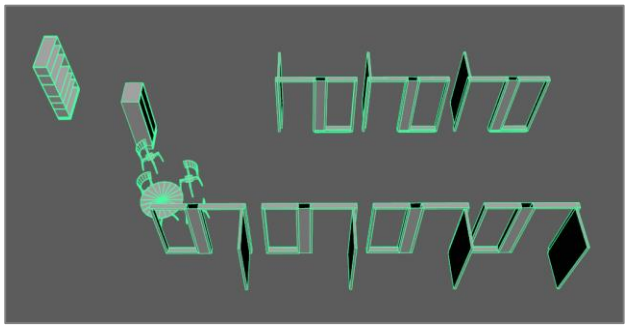

Figure 3:5 Twenty-one combined wood material models.

The in-built static batch tick box tries to do combine these objects together by itself. It has conditions that the models have to match up to and if they don't, these objects will be ignored in the batching process. This method was used in the original (Non-VR) office environment.

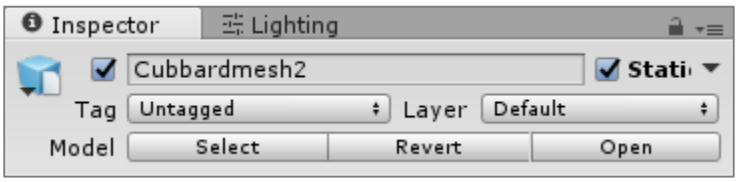

Figure 3:6 Static tick box in the top right.

3.3 Triangle count

This will not result in a visual change; it will be demonstrative with a higher density of polygons that are not optimized. Therefore, it's a nonvisual change to the office environment.

Subdivision of the geometry in Maya was used as a method to test the effect of increasing the polygon density. The new office environment scene models replaced the original inside Unity's file structure, everything except the model's polygon count stays the same.

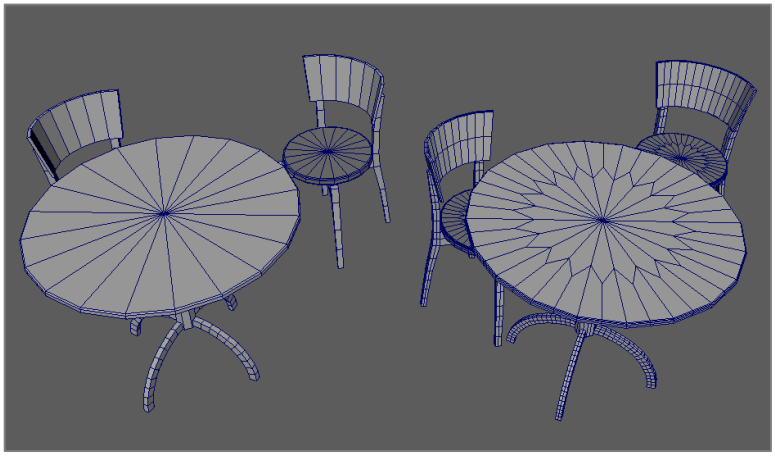

Figure 3:7 The models to the left are the original models, with a total of 1568 polygons. The ones on the right are the new, with a total of 6528 polygons.

3.4 Vertex count

This is a nonvisual change to the office environment. The method to test vertex count without a change in triangle count was done by separating each vertex normal creating a hard edge between each line, as shown in figure 3:28. The new office environment scene models replaced the original inside Unity's file structure, everything except the models' normal directions stay the same.

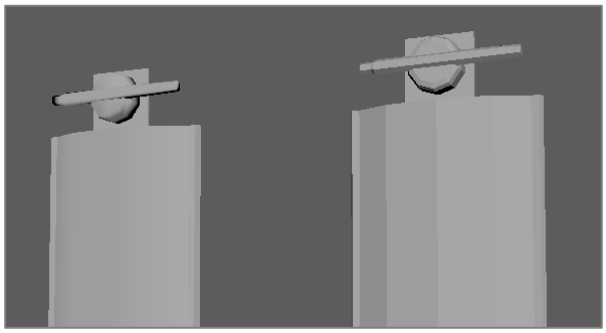

Figure 3:8 Left geometry uses soft vertex normals and the right geometry uses hard vertex normals.

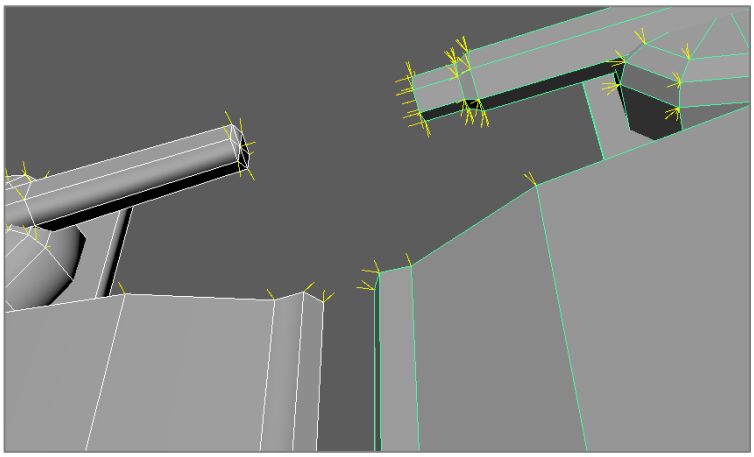

Figure 3:9 **The yellow lines represents the angle of the normal.**
Left illustration has softened- and right illustration has hardened normals.

3.5 Texture quality

This is a visual change to the office environment. The method changes the compression of the textures created for the environment.

Render quality is lowered to a quarter of the resolution in Unity's quality settings in the original office environment. Full and quarter resolution are tested.

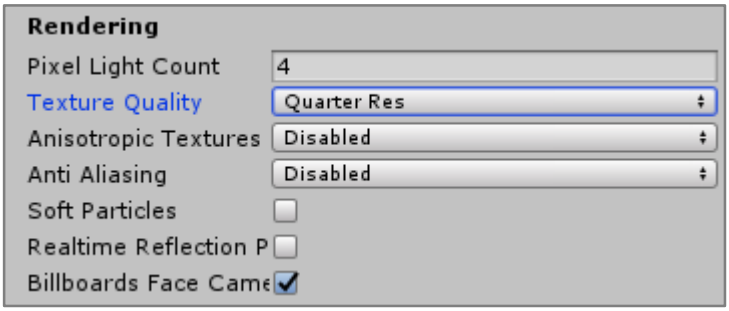

Figure 3:10 **Texture quality options. Full, half, quarter and eighth resolution available.**

3.6 Materials

This is a visual change to the office environment. The two methods used to the test the materials effect in the environment were to convert the four non-optimized Unity materials from the original office environment to mobile materials.

The second was to convert all the mobile optimized materials to the Unity standard, non-optimized for mobile devices.

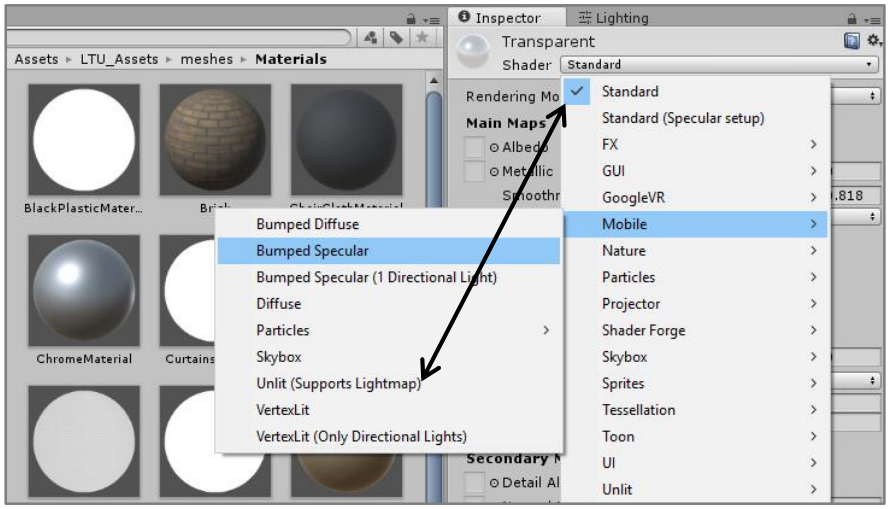

Figure 3:11 **Non-optimized materials from the office environment are converted into mobile versions. As well as converting all materials to the standard Unity material. Standard material and Unlit (Supports Lightmap) materials were tested.**

3.7 Lighting

This is a visual change to the office environment. The method used to the test lighting was to add real-time shadows for a single light source and increase shadow quality settings to high.

Mobile materials used in the original office environment are unlit with support for Lightmaps. This means no shadow will fall on any surface. All materials have to be converted to another less optimized material.

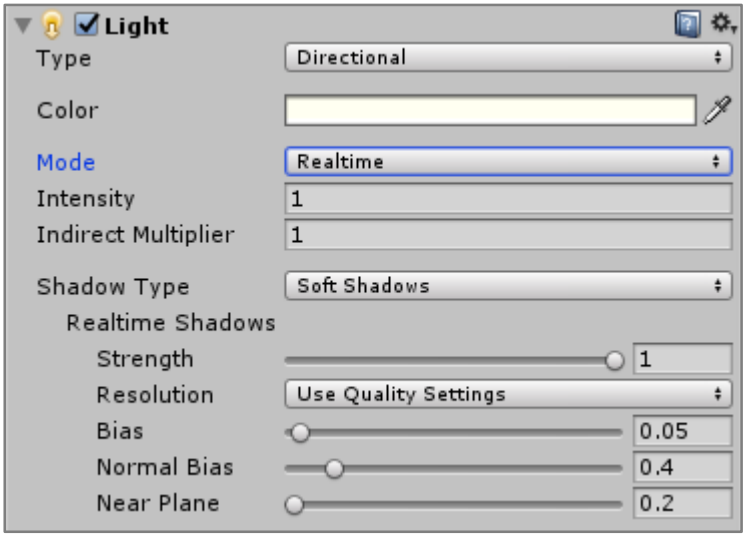

Figure 3:12 **Direction light set to real-time instead of baked.**

Figure 3:13 **Shadow resolution set to high resolution.**

3.8 Post-processing effects

3.8.1 Anti-Aliasing

This is a visual change to the office environment. The method used to the test anti-aliasing was to active it in both the cameras and in the render settings. This is a camera effect applied inside Unity.

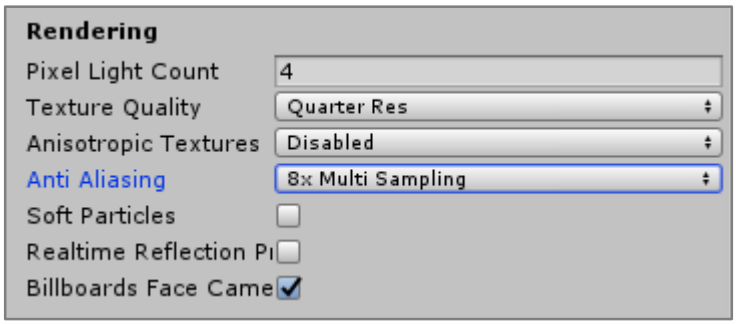

Figure 3:14 **Anti-Aliasing set to 8x.**

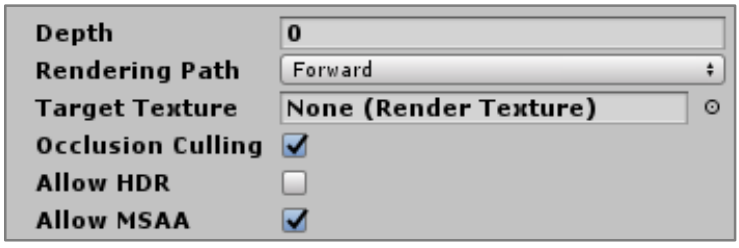

Figure 3:15 **Activate MSAA on both cameras (eyes).**

3.8.2 Ambient Occlusion

This is a visual change to the office environment. The method used to the test real-time ambient occlusion or AO was to active it in both of the cameras and to turn off the baked AO from the original office environment. This is a camera effect applied directly inside Unity.

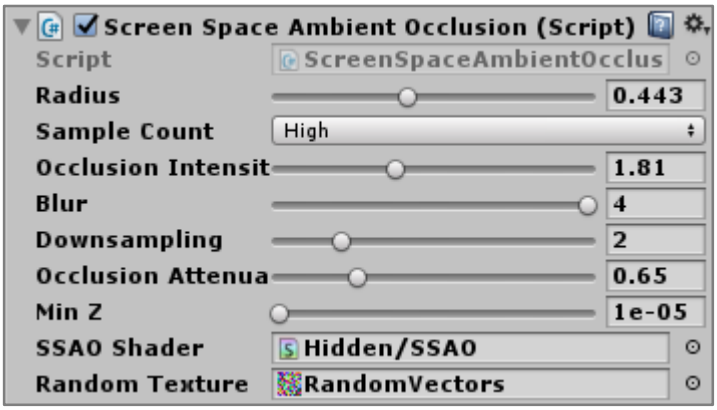

Figure 3:16 Activate real-time AO on both cameras (eyes).

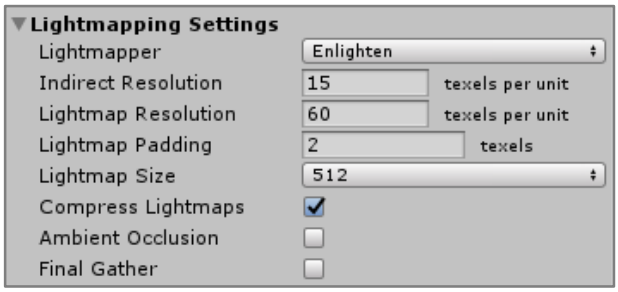

Figure 3:17 De-activate baked AO on both cameras (eyes).

3.9 Critique of Method

The method used will not represent the whole picture when performing the tests. Collecting changes in FPS data does not represent for example: Increased loading times instead of the data being handled in real-time.

Capturing and breaking down all the data from Unity's profiler would result in a more accurate, real life reading of the optimization methods used.

4 Result

Original office environment scene fps through a VR camera.

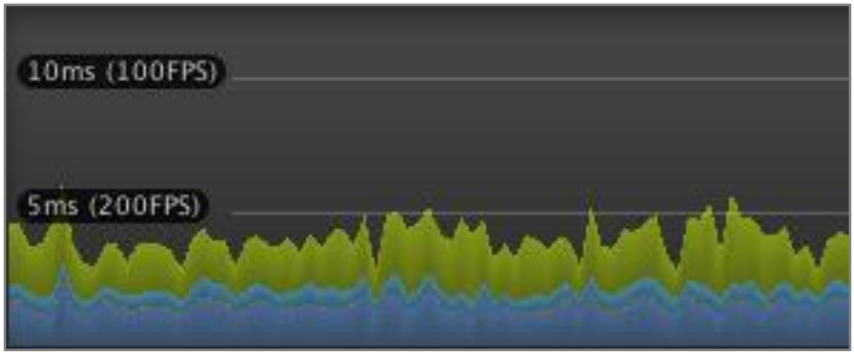

Figure 4:1 Original office environment FPS in VR. (bigger value is better)

4.1 Draw call batching

Optimization	Draw calls	FPS
In-built static batching	238	*210
Static & material batching	61	*230

*In-built static batching results in figure 4:1

*Static and material batching results in figure 4:2

Batching both material and static objects resulted in a FPS increase of 10% for lowing the draw calls with 1/3 of the original method used.

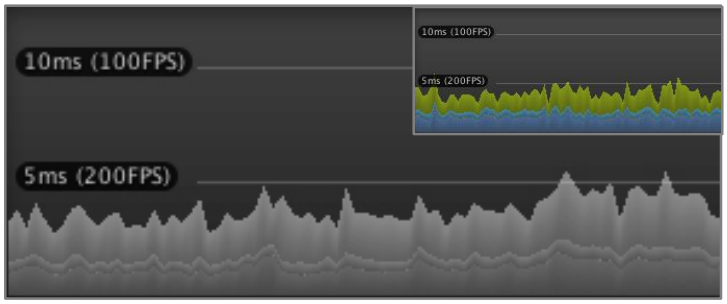

Figure 4:2 FPS test Static and material draw call batching, 61 draw calls. In-built static batching to the right (bigger value is better)

4.2 Triangle count

Optimization	Triangle count	FPS
Original triangle density	26,400	*210
Increased triangle density	146,000	*110

*Original triangle density on models results in figure 4:1
*Increased triangle density on models results in figure 4:4

Increased the triangle count resulted in a FPS cost of 45% for 5,5 times more triangles.

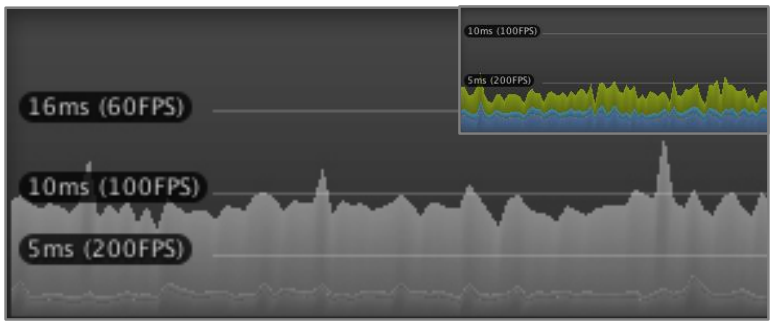

Figure 4:4 FPS test after a 5,5 times increase in triangle count. Original triangle density to the right. (bigger value is better)

4.3 Vertex count

Optimization	Vertex count	FPS
Original vertex count	39,600	*210
Increased triangle density	55,600	*180-200

*Original vertex count results in figure 4:1
*Increased vertex count results in figure 4:5

Increased vertex count resulted in a FPS drop of 5% for 1,4 times more vertex count.

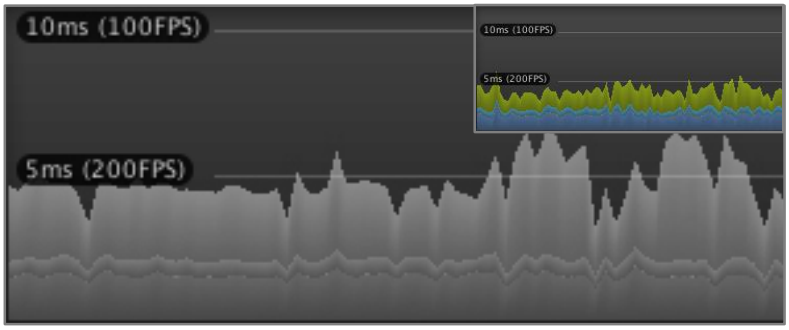

Figure 4:5 FPS test after a 1,4 times increase in vertex count. Original vertex count to the right. (bigger value is better)

4.4 Texture quality

Figure 4:6 & 4:7 Quarter resolution to the left and full resolution to the right.

Figure 4:8 Subtraction between quarter and full resolution to find pixels that are affected. Deleting all white pixels and comparing pixels to determinate the percent.

Optimization	Pixels affected	FPS
Quarter resolution	-	*210
Full resolution	16%	*213

*Quarter resolution results in figure 4:1 and 4:6
*Full resolution results in figure 4:7, 4:8 and 4:9

Increased texture quality to full resolution results in a FPS drop of 0 to 3% for affecting 16% of the pixels on the screen.

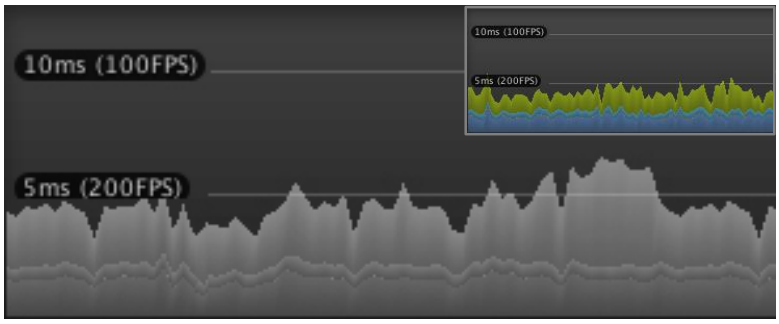

Figure 4:9 FPS test after a texture quality set to full resolution. Quarter resolution to the right. (bigger value is better)

4.5 Materials

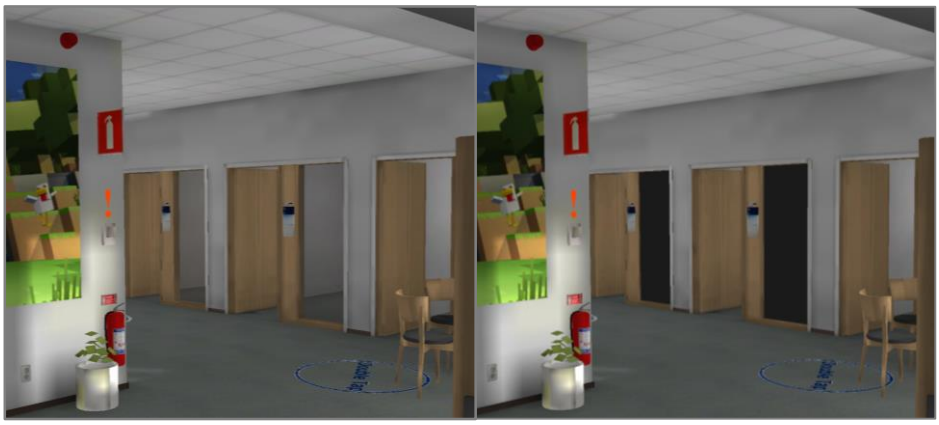

Figure 4:10 & 4:11 Original office environment to the left. Result after materials are optimized for mobile devices to the right.

Figure 4:12 Subtraction between mobile and standard Unity optimized materials to find pixels are affected. Deleting all white pixels and comparing pixels to determinate the percent.

Optimization	Materials	FPS
Standard Unity material	4	*210
Standard Unity material	0	*190

*4 standard Unity materials results in figure 4:1 and 4:10
*Standard Unity materials converted into mobile optimized results in figure 4:11, 4:12 and 4:13

Optimizing standard Unity materials to mobile friendly materials resulted in a FPS drop of 10% for affecting 20% of the pixels on the screen.

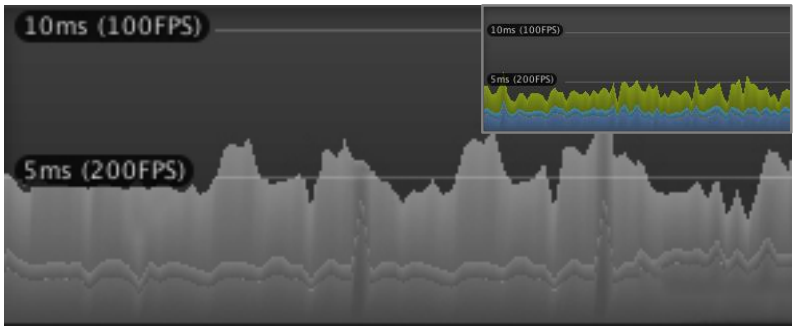

Figure 4:13 FPS test after all materials are optimized for mobile devices. Standard Unity material to the right. (bigger value is better)

4.6 Lighting

Figure 4:14 & 4:15 Original office environment to the left. Result after real-time shadows and supported materials are assigned to the right.

Figure 4:16 Subtraction between baked and real-time shadows to find pixels are affected. Deleting all white pixels and comparing pixels to determinate the percent.

Optimization	Pixels affected	FPS
Light baking	-	*210
Real-time shadows	36%	*160

*Light baked shadows results in figure 4:1 and 4:14
*Real-time shadows and converted into supported mobile optimized materials results in figure 4:15, 4:16 and 4:17

Real-time shadows resulted in a FPS drop of 26% for affecting an average of 36% of the pixels on the screen.

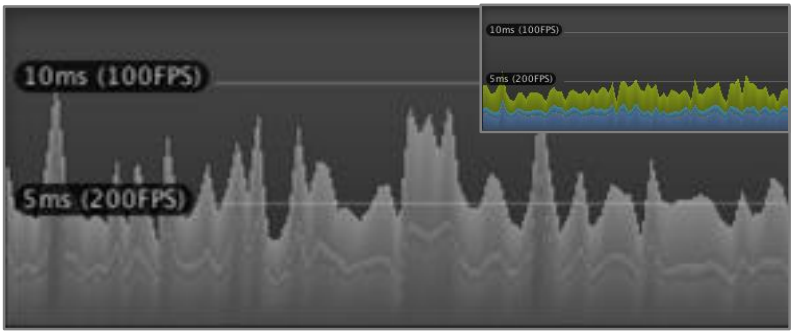

Figure 4:17 FPS test after all materials are optimized for mobile devices. Light baking to the right. (bigger value is better)

4.7 Post-effects

4.7.1 Anti-Aliasing

Figure 4:18 & 4:19 Original office environment to the left. Result after 8x Anti-Aliasing to the right.

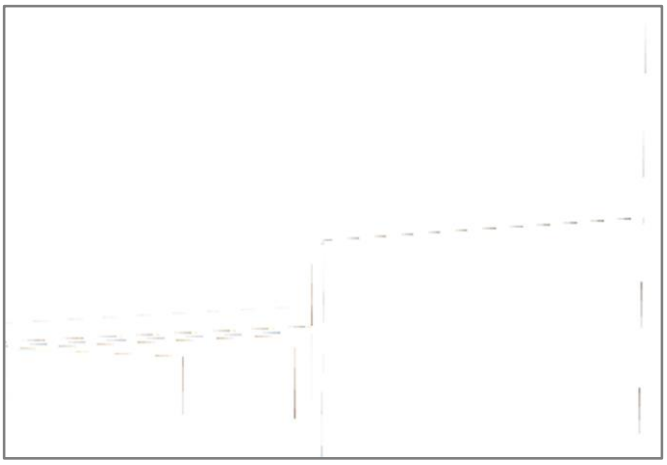

Figure 4:20 Subtraction between no anti-aliasing and 8x to find pixels are affected. Deleting all white pixels and comparing pixels to determinate the percent.

Optimization	Pixels affected	FPS
No Anti-aliasing	-	*210
8x Anti-aliasing	1-2%	*110

*No anti-aliasing results in figure 4:1 and 4:18

*8x anti-aliasing results in figure 4:19, 4:20 and 4:21

Increased anti-aliasing to 8x resulted in a FPS drop of 50% for affecting an average of 50% of the pixels on the screen.

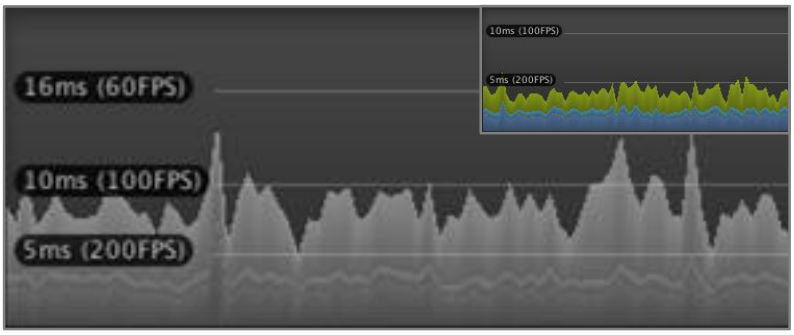

Figure 4:21 FPS test results after 8x anti-aliasing. Original office environment to the right. (bigger value is better)

4.7.2 Ambient Occlusion

Figure 4:22 & 4:23 Original office environment to the left. Result after real-time ambient occlusion to the right.

Figure 4:24 Subtraction between real-time and baked ambient occlusion to find pixels are affected. Deleting all white pixels and comparing pixels to determinate the percent.

Optimization	Pixels affected	FPS
Baked ambient occlusion	-	*210
Real-time ambient occlusion	37%	*170

*Baked AO results in figure 4:1 and 4:22

*Real-time AO results in figure 4:23, 4:24 and 4:25

Real-time AO resulted in a FPS drop of 20% for affecting 37% of the pixels on the screen.

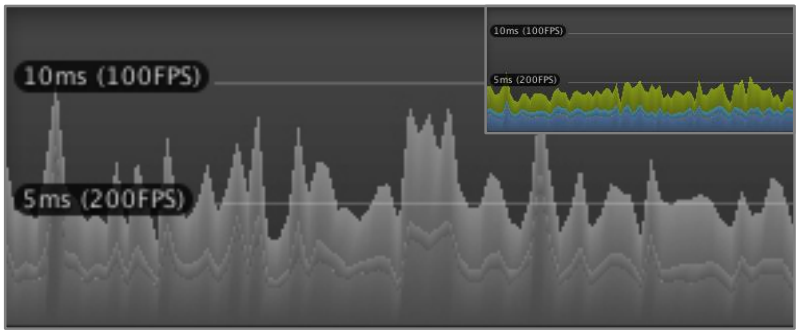

Figure 4:25 FPS test results after real-time AO. Baked AO to the right. (bigger value is better)

5 Discussion

The original question posed was; The cost vs performance – visually, and which parameters were most important for a realistic result.

This thesis has investigated the amount of pixels modified and their cost in FPS. The methods used are the typical solutions when optimizing graphics, some of which have been successfully and some which have not for this VR environment.

The visual results versus impact for the end user is a subject not touched upon that would skew the results heavily from the results in this thesis. For example, how much does the end user notice the 8x anti-aliasing versus no anti-aliasing. As well as other important factors such as if memory bandwidth were taken into consideration.

Non-Visual results

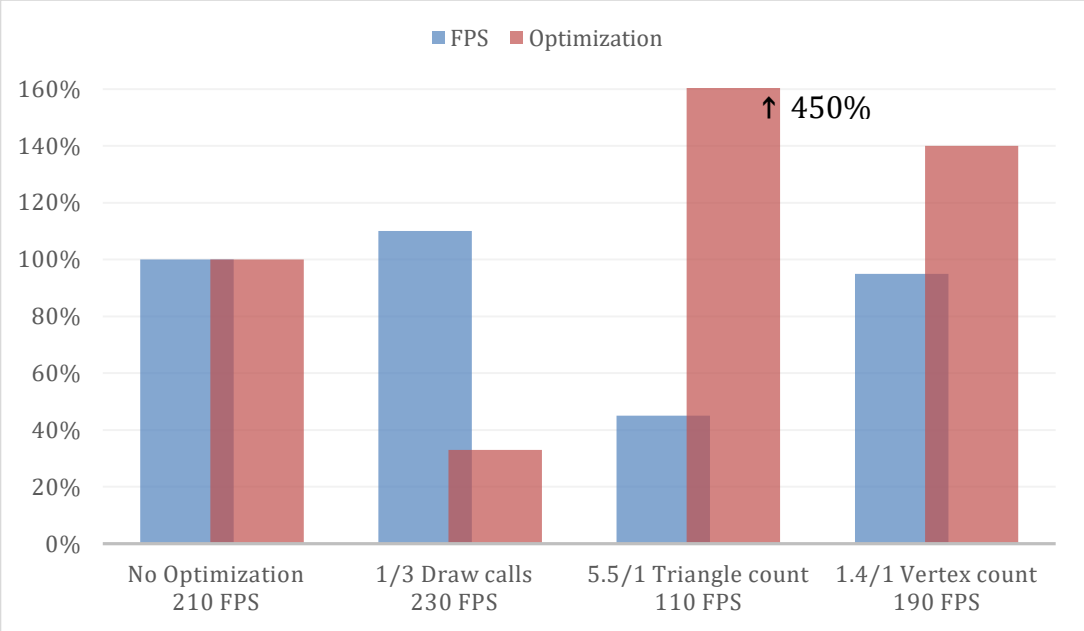

Figure 5:1 Chart illustrates all non-visual test results. 100% displays the original office environment FPS with no optimizations made. Red pillar is the percent difference in an optimization "setting". A bigger blue column means higher FPS (better) and red means an unoptimized test (worse).

Non-Visual results are optimization on the CPU. In the application and geometry stages it comes down to optimizing as much data as possible before sending it further in the pipeline, to the rasterizer where the GPU takes over.

Optimization made by Samuel Lundsten for the office environment LTU project were aggressive and to test performance going any lower would impact the visual integrity of the models. Hence the triangle and vertex count tests are not optimization for better FPS. But worse to compare to the already optimization made by S. Lundsten.

Visual results

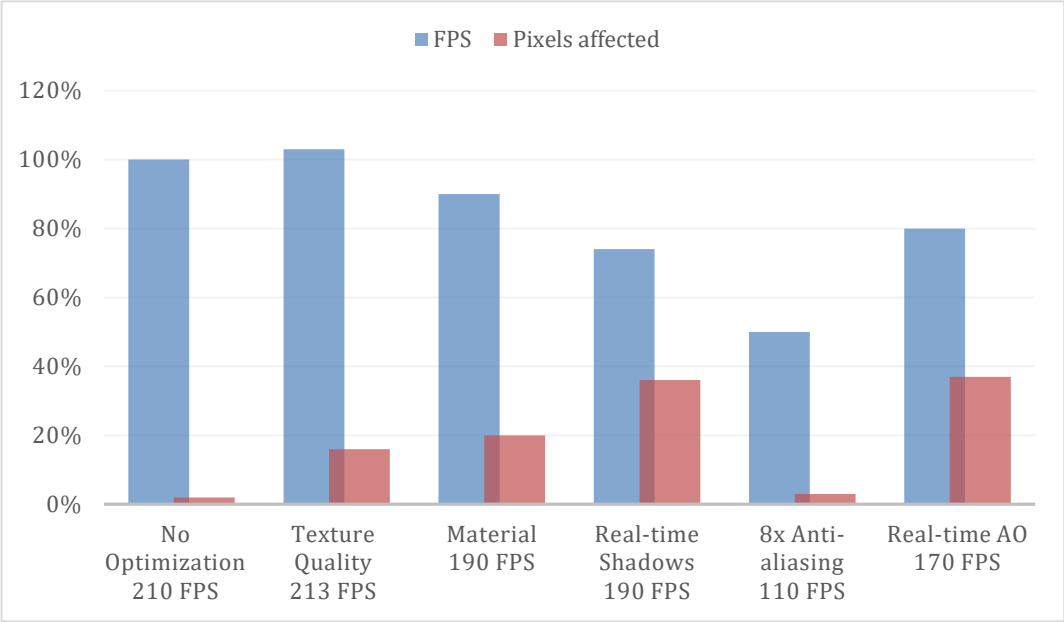

Figure 5:2 Chart illustrates all visual test results. 100% displays the original office environment FPS with zero pixels affected. Red pillar is the percent difference in pixels. A bigger blue column means higher FPS (better) and red means more pixels affected (bigger change).

Visual results are optimization on the GPU. In the rasterizer stage and more specifically, the pixel shader. Drawing the pixels and handling everything on a per-pixel level. The visual result of anti-aliasing was surprising in that such as small change in pixels has such a huge effect on FPS and overall realism of the images.

5.1 Anti-Aliasing test

The results of the anti-aliasing test created the worst ratio of pixels changed versus FPS. When comparing the original and the test together with a photograph of real-life office the anti-aliasing is needed to achieve a more realistic result.

A 1-2% modification of the entire screens pixels for the cost of half your FPS. Not surprising when you consider what happens when applying 8x anti-aliasing when drawing the pixels in the rasterizer stage. 8x AA would render 16 super-sampled pixels for each single pixel in each frame. Essentially rendering at eight times the display resolution.

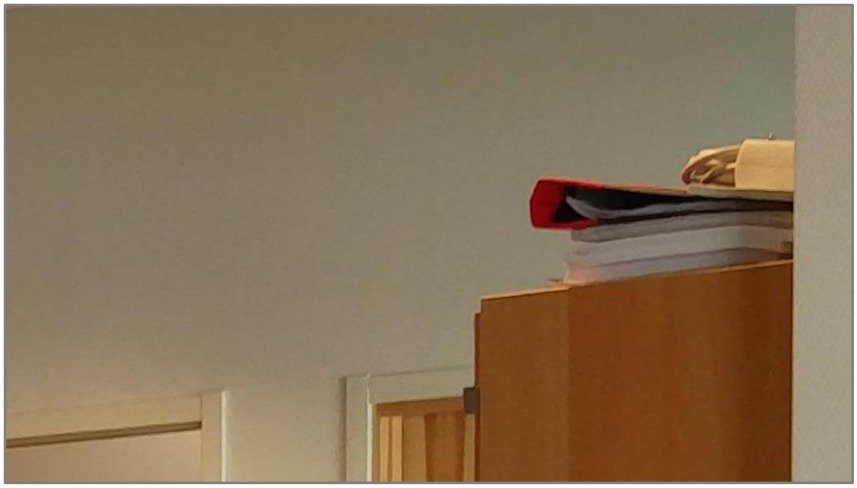

Figure 5:1 Photograph of the "anti-aliasing" in the real-life office environment

Figure 5:2 & 5:3 Original office environment to the left. Result after 8x Anti-Aliasing to the right

5.2 Texture quality test

The results of the texture quality test created by far the best of pixels changed versus FPS. Comparing the original and the test together with a photograph of real-life office it is clear that an increase in texture quality is needed to achieve a more realistic result.
This however is not completely true, if memory bandwidth or loading time were taken into consideration the it would increase a lot with this test.

A modification of 16% the entire screens pixels for the cost of 0-3% FPS.
The FPS will remain the same because the file is loaded into the graphics memory of the device before the application/game is rendered. The data transfer speed on graphics memory is fast enough for textures to be handled in such a way, affecting only loading times.

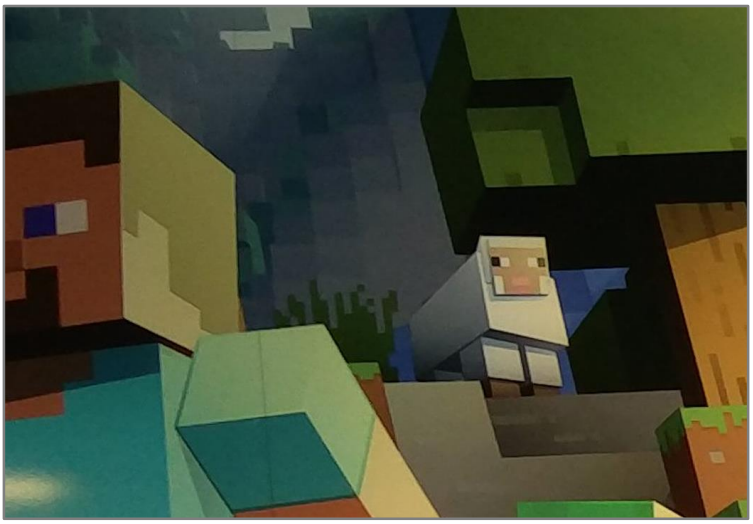

Figure 5:4 Photograph of the "texture quality" in the real office environment

Figure 5:5 & 5:6 Quarter resolution to the left and full resolution to the right

This thesis discussion concludes with a few points to think about that are also confirmed by Unity's documentation.

- All mobile devices are not created equal. There are much slower and much faster phones.
- Both artists and programmers need to know the limitations of the platform, and the methods that are used to get around them.
- Choosing methods of optimizing or game design that fits your application/game. For example, in VR applications/games some anti-aliasing is needed to remove the jagged lines and most importantly a stable 90 FPS is a must.

- Profile early and often to discern which optimizations will pay off with big performance increases and which are a waste of time. Spending your resources in the right
- Don't optimize too early, especially in application/game development for smartphones. The computational capability of mobile devices is increasing at an extraordinary rate. It's not unheard of for a new generation of a mobile GPU to be five times faster than its predecessor. That's incredibly fast when compared to the PC industry.

Often quoted for his rules of program optimization:

"The first rule of program optimization: don't do it. The second rule of program optimization (for experts only!): don't do it yet."

- *Michael A. Jackson*

6 Conclusion

Optimization of performance limited devices is in itself hard. By adding VR which is still in its infancy onto that, you create some very hard questions for your game/application. Questions such as, how do you get the biggest impact visually versus performance cost. What visuals changes are noticed by the end user, and so on.

Throughout this thesis a more interesting question is always approaching in the mind; *Is it needed, and if so, how much?*

How much is needed until the brain is "sold" on the VR world. Immersion in a game world is a way to get past the details in the picture that is the game. In the same way nostalgia puts a rosy glean around an old video game. How far do you have to go to achieve a realistic result?

Further work would include:

- Measuring the effect visually in pixels and breaking down all relevant technical factors such as, CPU usage, memory and rendering.
- How to create realistic transitions of LODs in VR and their performance.
- LODs in the peripheries of the eyes. Taking advantages of the focus of the eyes in VR, some sort of eye tracking inside the VR headset is needed for something like this.
- A look at where immersion breaks and the limits of believability. (believability is not necessarily photorealistic)

7 References

[1] Timothy J. B., Dennis A. V., and John E. D. (2012) *"The Effect of Apparent Latency on Simulator Sickness While Using a See-Through Helmet-Mounted Display: Reducing Apparent Latency with Predictive Compensation"* Human Factors and Ergonomics Society

[2] Vlachos, A. (2015) *"Advanced VR Rendering" VALVE*

[3] Cochrane, N. (1994) *VFX-1 VIRTUAL REALITY HELMET by Forte*. Game Bytes Magazine

[4] Martijn J. Schumie et al. *"Research on Presence in Virtual Reality: A Survey".*

[5] Tom Forsyth. *"Connect: Developing VR Experiences with the Oculus Rift".*

[6] Eugenia M. Kolasinski. *"Simulator Sickness in Virtual Environments".*

[7] Matt Wuebbling (2012). *"Mobile graphics moving toward console level".* NVIDIA

[8] Experts Exchange (2015). *"Processing power compared - Visualizing a 1 trillion-fold increase in computer performance".* Experts Exchange

[9] Sebastian Anthony (2014). *"Does the iPhone 6 actually have console-quality graphics?".*

[9.1] Joel Hruska (2013). *"Reverse engineered PS4 APU reveals the console's real CPU and GPU specs".*

[9.2] Sebastian Anthony (2014). *"Apple's A8 SoC analyzed: The iPhone 6 chip is a 2-billion-transistor 20nm monster".*

[10] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed"*. AK PETERS

[10.1] Barfield, W., Baird, K.M., Bjorneseth, O. J. (1998). *"Presence in Virtual Environments as a function of type of input device and display update rate".*

[11] K. T. Claypool (2007). *"On frame rate and player performance in first person shooter games".*

[12] Rubino, C., & Power, J. (2008). *"Level design optimization guidelines for game artists using the epic games: Unreal editor and unreal engine 2".* Computers in Entertainment (CIE)

[13] Laviola, J. J. Jr (2000). *"A discussion of Cybersickness in virtual environments".* ACM SIGCHI

[14] Kolasinski, E. M. (1995). *"Simulator sickness in virtual environments".* U.S. Army Research Institute for the Behavioral and Social Sciences

[15-23] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed"*. Chapter 1 AK PETERS

[24] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed"*. P. 21-37 AK PETERS and J. C. Prunier (2015) *"Rasterization: a Practical Implementation".* Scratchpixel

[25] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed"*. P. 21-37 AK PETERS

[26] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed"*. P. 21-37 AK PETERS and J. C. Prunier (2015) *"Rasterization: a Practical Implementation".* Scratchpixel

[27] Unity Technologies. (2017). *"Optimizing graphics performance".* Unity manual

[28] Hillaire, S. (2012). *"Improving Performance by Reducing Calls to the Driver".* OpenGL Insights, A K Peters

[29] Unity Technologies. (2017). *"Optimizing graphics rendering in Unity games".* Unity learning platform

[30] Wloka, M. (2003). *"Batch, Batch, Batch: What Does It Really Mean?*". Presentation at Game Developers Conference 2003

[31] Unity Technologies. (2017). *"Optimizing graphics rendering in Unity games".* Unity learning platform

[32] Kennedy, S. M. (2013). *"How to become a video game artist: The insider's guide to landing a job in the gaming industry".* Watson-Guptill Publications.

[33] Solarski, C. (2012). *"Sponsored feature: Drawing basics and video game art: Character design".* Gamasutra

[34] McGrath, T. (2008). *"Creating efficient next gen environment art".* Gamasutra

[35] Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "*Real-Time Rendering* 3rd *ed"*. P. 147 AK PETERS

[36] Unity Technologies. (2017). *"Optimizing graphics performance".* Unity manual and E. Chadwick (2003) *"Beautiful, Yet Friendly Part 1 & 2: Stop Hitting the Bottleneck & Maximizing Efficiency".*

8 Appendices

I Figure 2:1 By othree - Google Cardboard, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=40703922
II Figure 2:2 By Adam Patrick Murray – PCWorld, http://www.pcworld.com/article/3051558/hardware/htc-vive-review-reach-out-and-touch-the-virtual-world.html
III Figure 2:3 By Adam Patrick Murray – PCWorld, http://www.
IV Figure 2:4 By Experts Exchange - Experts Exchange, http://pages.experts-exchange.com/processing-power-compared/
V Figure 2:5 By Lenka – IFIXIT, https://www.ifixit.com/Teardown/Sony+Xperia+Z5+Teardown/52300
VI&VII Figure 2:6 & 2:7 By Chipworks – IFIXIT, https://www.ifixit.com/Teardown/PlayStation+4+Teardown/19493
VIII Figure 2:8 By Ilyass – Blogger, http://grainsdecesames.blogspot.se/2010_07_01_archive.html
Figure 2:9 By me – Reference: Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "Real-Time Rendering 3rd ed". Chapter 1 AK PETERS and http://jadrian.org/courses/pa/comp_gfx/2016.0/index.html
IX Figure 2:10 From book – Reference: Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "Real-Time Rendering 3rd ed". Chapter 1 AK PETERS
X Figure 2:11 From book – Reference: Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "Real-Time Rendering 3rd ed". Chapter 1 AK PETERS
XI Figure 2:12 From book – Reference: Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "Real-Time Rendering 3rd ed". Chapter 1 AK PETERS
XII Figure 2:13 From Scratchpixel - https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
XIII Figure 2:14 From Scratchpixel - https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
XIV Figure 2:15 From book – Reference: Akenine-Möller, T., Haines, E. and Hoffman, N. (2008) "Real-Time Rendering 3rd ed". Chapter 1 AK PETERS
XV Figure 2:16 From web - http://www.keywordsuggests.com/6Od2b%7CSorAgWpEMQ*syTGcuV9*UL9Ah8AHfAEkKspdQ/
XVI Figure 2:17 by wiseGEEK - http://www.wisegeek.com/what-is-a-slide-projector.htm#
XVII Figure 2:18 From Scratchpixel - https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
Figure 2:19 By me - Reference: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
Figure 2:20 By me - Reference: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
Figure 2:21 By me - Reference: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
Figure 2:22 By Rchoetzlein - Wikimedia, https://commons.wikimedia.org/wiki/File:Mesh_overview.svg
XVIII Figure 2:23 By The Foundry – Smoothing vertex normals, https://help.thefoundry.co.uk/modo/content/help/pages/uving/vertex_normals.html
Figure 3:1 By me, screenshot: Real-time rendered office.
Figure 3:2 By me, screenshot: Photograph of real office.
Figure 3:3 By me, screenshot: VR point of view.
Figure 3:4 By me, screenshot: Combined 4 brick models from Maya.
Figure 3:5 By me, screenshot: Combined 21 wood models from Maya.
Figure 3:6 By me, screenshot: Static tick box in the top right.
Figure 3:7 By me, screenshot: The model to the left are the original models, with a total of 1568 polygons. The ones on the right are some tested, with a total of 6528 polygons.
Figure 3:8 By me, screenshot: Soft vertex normals to the left and hard to the right.

Figure 4:25 By me, screenshot: FPS test results after real-time AO. Original office environment to the right. (bigger value is better)