

Bachelor's thesis

Information and Communications Technology

2018

Sami Laukkanen

# POST-PROCESSING IN VIDEO GAMES

**TURKU AMK**   
TURKU UNIVERSITY OF  
APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2018 | Pages: 54

Supervisor: Principal Lecturer Mika Luimula, Adj.Prof

Sami Laukkanen

## POST-PROCESSING IN VIDEO GAMES

The thesis is about post-processing in video games. Post-processing is a process where an image on a computer screen is altered using different effects, such as changing the colors or adjusting the brightness. It is a polishing process, used to improve the visual quality of a game and it is usually implemented in the later phases of game development.

The objective of the thesis was to acquire a better understanding on how post-processing works in game development and what kind of challenges it presents. The thesis will also explain how important post-processing is for game visuals and why it should be used.

The studied post-processing theories were used to make a comparison between different game engines to find out how post-processing affects them. In the comparison two aspects were compared, performance and quality. The comparison was achieved by recording the frame rate in identical looking game environments. The results from the comparison should give game developers confidence to choose any of the compared game engines, since they provided very similar results.

These same methods were then used in the case study for a company Focusplan Ltd. The objective was to create a lightweight post-processing solution that could be easily used in all of the company's Unity game engine projects. The end result was three different post-processing profiles for Focusplan to choose from. The process of implementing the profiles into Focusplan's projects is not finished yet and it is completed in the near future.

### KEYWORDS:

Post-processing, Unity, Unreal Engine 4

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintäteknikka

2018 | 54 sivua

Ohjaaja: yliopettaja Mika Luimula, dos.

Sami Laukkanen

## POST-PROSESSOINTI VIDEOPELEISSÄ

Opinnäytetyön aihe on post-prosessointi videopeleissä. Se on prosessi, jossa tietokoneen ruudulla olevaa kuvaa muokataan käyttäen erilaisia tehosteita, kuten värien muuntamista tai kirkkauden säätöä. Post-prosessointi tehdään yleensä vasta pelinkehityksen jälkivaiheilla, ja sitä käytetään pelin visuaalisen ilmeen kohottamiseen.

Opinnäytetyön tavoitteena oli saada parempi ymmärrys, miten post-prosessointi toimii pelinkehitysympäristössä ja minkälaisia haasteita se tuo mukanaan. Työ myös selittää, kuinka tärkeä post-prosessointi on peligrafiikalle ja miksi sitä tulisi käyttää.

Tutkittuja teorioita post-prosessoinnista käytettiin vertailun tekemiseen eri pelimoottoreiden välillä, ja vertailusta saatiin selville kuinka post-prosessointi vaikuttaa niihin. Vertailussa verrattiin post-prosessoinnin suorituskykyä ja laatua nauhoittamalla ruudunpäivitysnopeutta identtisissä peliympäristöissä. Vertailun tulosten pitäisi antaa pelinkehittäjille itsevarmuutta eri pelimoottoreiden valitsemiseen, sillä tulokset eri pelimoottoreista olivat hyvin samankaltaisia.

Näitä samoja menetelmiä käytettiin Focusplan Oy:lle tehdyssä tapaustutkimuksessa. Tavoitteena oli tehdä kevyt post-prosessointiratkaisu, jota voisi käyttää yrityksen kaikissa Unity-pelimoottoria käyttävissä projekteissa. Lopullinen tulos oli kolme erityylistä post-prosessointiprofiilia, joista Focusplan voi päättää mieleisensä. Profiileita ei ole vielä otettu käyttöön yrityksen projekteissa, mutta tämä tullaan toteuttamaan lähitulevaisuudessa.

ASIASANAT:

Post-prosessointi, Unity, Unreal Engine 4

# CONTENT

|   |           |
|---|-----------|
| <b>LIST OF ABBREVIATIONS (OR) SYMBOLS</b>             | <b>6</b>  |
| <b>1 INTRODUCTION</b>                                 | <b>8</b>  |
| <b>2 IMPACT OF POST-PROCESSING</b>                    | <b>9</b>  |
| 2.1 How visuals affect the reception of a game        | 10        |
| 2.2 Problems with bad post-processing and lighting    | 13        |
| 2.3 Key features towards a successful post-processing | 15        |
| <b>3 POST-PROCESSING EFFECTS</b>                      | <b>20</b> |
| 3.1 Shaders   | 20        |
| 3.2 Anti-aliasing                                     | 21        |
| 3.3 Tone mapping                                      | 24        |
| 3.4 Color Grading                                     | 25        |
| 3.5 Depth of Field                                    | 26        |
| 3.6 Motion Blur                                       | 27        |
| 3.7 Film Grain  | 29        |
| 3.8 Vignette  | 31        |
| 3.9 Bloom   | 32        |
| 3.10 Chromatic Aberration                             | 33        |
| 3.11 Ambient Occlusion                                | 34        |
| 3.12 Screen Space Reflection                          | 38        |
| 3.13 Eye Adaptation                                   | 39        |
| <b>4 POST-PROCESSING IN DIFFERENT GAME ENGINES</b>    | <b>41</b> |
| 4.1 Unity   | 41        |
| 4.2 Unreal Engine 4                                   | 42        |
| 4.3 Benchmarking                                      | 42        |
| <b>5 FOCUSPLAN CASE STUDY</b>                         | <b>46</b> |
| <b>6 CONCLUSION</b>                                   | <b>50</b> |
| <b>REFERENCES</b>                                     | <b>52</b> |

## PICTURES

|   |    |
|---|----|
| Picture 1. Image in Unity game engine to show the impact of post-processing. The top side of the image is with post-processing and the bottom is without post-processing.               | 10 |
| Picture 2. A screenshot from the Destiny 2's opening cinematic (Activision 2018).   | 11 |
| Picture 3. A comparison of post-processing using different resolutions.   | 12 |
| Picture 4. A screenshot of cluttering post-processing effects in Battlefield 4 (Electronic Arts 2018).  | 14 |
| Picture 5. Demonstration of bloom gone wrong in the Legend of Zelda: Twilight Princess (Gallant 2008).  | 15 |
| Picture 6. A screenshot of the graphics options in a game called Warframe (Digital Extremes 2018).  | 18 |
| Picture 7. A demonstration how anti-aliasing affects to jagged edges. Left side is without anti-aliasing and the right is with.   | 22 |
| Picture 8. Effects of HDR rendering on a LDR monitor, with and without tone mapping.  | 25 |
| Picture 9. A demonstration how the color grading works in Unity. The left side is without color grading and the right is with. On the very right side is the actual color grading tool. | 26 |
| Picture 10. A demonstration that shows how changing the aperture affects the depth of field. The left side is with an aperture of 1.2 while the right uses a value of 8.                | 27 |
| Picture 11. A demonstration of motion blur affecting rapid camera movement in Unity game engine.  | 28 |
| Picture 12. Digital noise reducing the color banding effect in Unity.   | 30 |
| Picture 13. A comparison between different digital noise settings in Unity.   | 30 |
| Picture 14. A demonstration how vignette effect looks in a bright test environment in Unity.  | 32 |
| Picture 15. A comparison between different bloom settings in Unity.   | 33 |
| Picture 16. A demonstration of chromatic aberration effect in Unity.  | 34 |
| Picture 17. Visualization of a depth buffer in Unity.   | 35 |
| Picture 18. A comparison of different SSAO qualities. The two views on top have only the ambient occlusion effect displayed for visualization purposes.                                 | 37 |
| Picture 19. A comparison between screen space reflections and reflection probes in Unity.   | 39 |
| Picture 20. Eye adaptation effect at different brightness levels in Unity.  | 40 |
| Picture 21. A comparison of post-processing settings between Unity and Unreal Engine.   | 45 |
| Picture 22. A comparison of the different post-processing profiles used in the Focusplan case study.  | 48 |

## TABLES

|  |    |
|--|----|
| Table 1. Performance comparison results of comparing post-processing in Unity and Unreal Engine 4. | 43 |
|--|----|

## LIST OF ABBREVIATIONS (OR) SYMBOLS

|                 |  |
|-----------------|--|
| Unity           | A cross-platform game engine developed by Unity Technologies   |
| Unreal Engine 4 | A cross-platform game engine developed by Epic Games   |
| Image buffer    | A memory designated to contain an image that is going to be worked on by the processor (Roberts 2017).   |
| Pre-rendering   | Pre-rendering is the process in which video footage is not rendered in real-time by the hardware that is outputting or playing back the video. Instead, the video is a recording of footage that was previously rendered on different equipment (typically one that is more powerful than the hardware used for playback). (Wikipedia 2017.) |
| Real-time       | Event or function which is processed instantaneously.  |
| Gameplay        | Gameplay is a term used to define the way players interact with a certain video or computer game. It is further characterized as the way the game is played, including the rules, the plot, the objectives and how to conquer them, as well as a player's overall experience. (Technopedia 2018.)  |
| Avatar          | A digital representation of a person or being  |
| GPU             | A Graphics Processing Unit is a single-chip processor primarily used to manage and boost the performance of video and graphics (Technopedia 2018).   |
| RGB             | The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors (Rouse 2005).  |
| Pixel           | In digital imaging, a pixel is a physical point in a raster image, or the smallest addressable element in an all points addressable display device, so it is the smallest controllable element of a picture represented on the screen (Rouse 2015).  |

|          |   |
|----------|---|
| Vertex   | A vertex in computer graphics is a data structure that describes certain attributes, such as the position of a point in 2D or 3D space, at multiple points on a surface (Math Open Reference 2011). |
| HDR      | Stands for High Dynamic Range. It means better contrast, greater brightness levels and a wider color palette (TechTerms 2017).  |
| LDR      | Stands for Low Dynamic Range. Term that is used for every display device that does not meet the requirements to be called HDR.  |
| FPS      | Frames per second is the frequency rate at which consecutive images called frames appear on a display (Rouse 2010).   |
| PC       | Personal Computer   |
| 3D       | Three-dimensional   |
| 2D       | Two-dimensional   |
| 3D model | In 3D computer graphics, 3D modeling is the process of developing a mathematical representation of any surface of an object in three dimensions via specialized software (Slick 2018).              |
| Texture  | Texture is the graphical skins laid atop of a 3D model so they appear to have a surface detail (Rouse 2010).  |

# 1 INTRODUCTION

Post-processing in video games is a process where full-screen filters and effects are applied to a camera's image buffer before it is displayed to a screen. This results into a much higher graphics fidelity but it can also affect the performance negatively (Unity documentation 2017). Post-processing plays a really large role in modern video game graphics and it is very noticeable when it is missing or implemented incorrectly. That is why, it is good that modern game engines such as Unity and Unreal Engine 4 come packed with post-processing effects, which can be easily activated. This helps artists to use high quality post-processing effects without a need for a game programmer.

The thesis consists of two parts, the theoretical part which shows the importance of post-processing, and also explains different post-processing effects and how they can affect the mood of a game. The thesis will also examine the problems caused by post-processing. Post-processing is known to cause performance problems in video games and the thesis is going to address these issues. Optimizing is a part of this process and it allows post-processing to be available for a higher range of personal computers (PC).

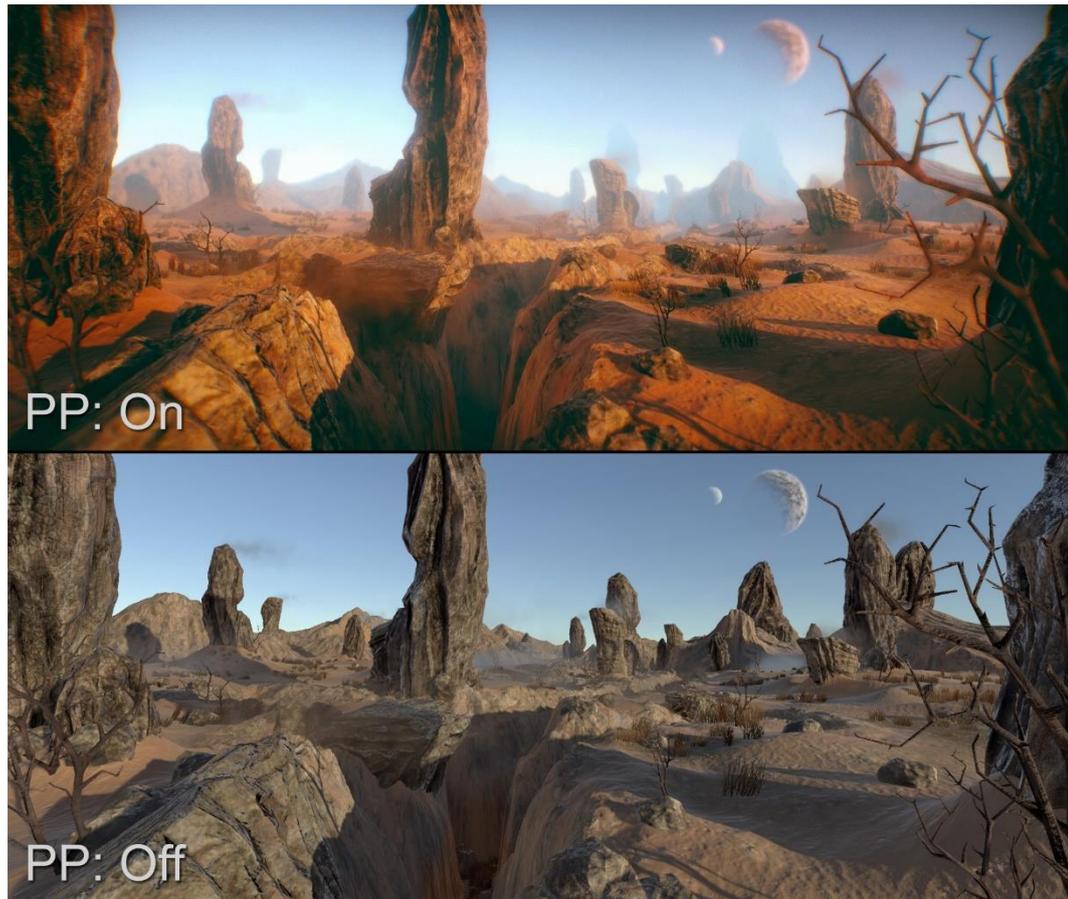
The practical part of the thesis uses the theories from the first part to create a finalized look for a project by Focusplan Ltd. The project is not finished mechanically and the goal is to provide a post-processing profile which can be also used in the other projects developed at Focusplan Ltd.

In the theoretical part the two game engines that are used to compare post-processing are Unity and Unreal Engine 4. Both engines are free to use, but when the games made with these engines start to generate revenue the game developers need to start paying. Both game engines are also well received by the game industry, while Unity is known for being beginner friendly, the Unreal Engine 4 is known for its industry pushing graphics quality. The engine used in the Focusplan's project is Unity and the project is meant to be shown on a CAVE projector system with 3D glasses. CAVE is a multi-projector system where multiple projectors are used to show a wider view than a single projector can deliver. Because the project uses multiple projectors, all of the post-processing effects introduced in the theoretical part cannot be used due to way post-processing handles multiple projectors.

## 2 IMPACT OF POST-PROCESSING

Video game graphics matter and this is even more apparent in the modern video game industry than ever before. Large games are coming out for Sony's and Microsoft's console systems, and almost always these games are pushing the graphics to the next level to advertise the console's processing power to sell more consoles (Usher 2015). Topics that are usually discussed about, are the beautiful looking worlds, lighting, the characters and the technologies behind the game. Taking a backseat in these discussions is the post-processing, if it is mentioned it is usually about how some of the effects are too intrusive and should be removed completely. What video gamers do not realize is that all modern games use at least some post-processing, and many games would be much duller looking without it.

Post-processing is much more than just applying more shininess to the final picture, it is about changing the whole style of a game, the mood and the quality of the graphics (Lane 2017). Post-processing is a crucial part of any game that wants to take itself seriously and not look like a prototype. When post-processing is applied correctly by an artist, it elevates the good looking environments into great looking ones. As shown in the Picture 1 below, post-processing can transform a bland looking environment into something with more charisma and depth. There are multiple different post-processing effects in use to create this polished scene, for example a color grading effect is used to change the environment into a warmer looking desert. Ambient occlusion soft shadows are used to create better ambient shadows around the rocks, and also the post-processed version has a better illusion of depth by using the depth of field and fog effects.



Picture 1. Image in Unity game engine to show the impact of post-processing. The top side of the image is with post-processing and the bottom is without post-processing.

## 2.1 How visuals affect the reception of a game

The first feature that a player will notice when launching a game is the graphics and if there is something wrong with them. This makes it very important to have a graphically flawless first impression (Shankar 2018). Nowadays many of the larger games start with a cinematic intro to immerse the players to the game's world and theme. These cinematics have post-processing effects if the game itself uses them, and as mentioned before, almost all games use at least some post-processing. What makes these cinematics special is that they use predetermined camera placements, angles and are sometimes pre-rendered. This gives artists much more freedom to use all of the post-processing effects at their disposal and do not have to worry about the performance of the game.

One of these games is the Bungie's Destiny 2, which is a role-playing multiplayer first-person shooter published by Activision Blizzard in 2017. When the player starts a new game in Destiny 2, they are presented right away with an opening cinematic. It tells the current situation to the player, but more importantly, it tries to immerse the player into playing the game. This can be achieved with movie quality animations, 3D models, music and most importantly, the post-processing.

The Picture 2 is a screen capture from Destiny 2's opening cinematic. In Picture 2 the post-processing can be seen everywhere, the depth of field, film grain, ambient occlusion and anti-aliasing effects are used to just name a few. Combining them all together creates a better looking and a finalized final image. Without post-processing, the cinematic would simply look unprofessional and not so impactful. This is because post-processing is also used in the film industry and the game cinematics are pulling their inspiration from there. The cinematic used in the Picture 2 is pre-rendered and does not represent the lower in-game graphics quality. In some cases this might cause frustration amongst the players, because the cinematic does not represent the actual game they can play.



Picture 2. A screenshot from the Destiny 2's opening cinematic (Activision 2018).

There are also many other games using cinematics in modern video games, but Destiny 2 was chosen for this example because it is mostly very well received for its graphics style and quality by the players.

Another good example is the new Doom game, a first-person shooter developed by ID software and published by Bethesda Softworks in 2016 for PC, PlayStation 4 and Xbox One. From the perspective of this thesis, these releases are not so interesting, but what

is interesting is the port for the Nintendo Switch console made in 2017. Nintendo Switch is a hybrid console between mobile and stationary modes. It is designed primarily as a home console, with the main unit inserted into a docking station to connect to a television. Alternatively, it can be removed from the dock and used similarly to a tablet computer through its touchscreen. This means that it has much less processing power than its competitors, the PlayStation 4 and the Xbox One game consoles. Converting the game to the Switch console decreases the overall graphics quality of the game, but what is interesting is that the post-processing effects remain almost the same, while the resolution of the textures and screen are reduced drastically. This is good, because without the post-processing effects Doom just would not look the same and when Switch is being played in handheld mode with small screen size, the resolution decrease is not as visible (Leadbetter 2017). The picture 3 is a demonstration of keeping the post-processing intact, while drastically reducing the image and texture resolutions. The resolution decrease is exaggerated to demonstrate the effect.



Picture 3. A comparison of post-processing using different resolutions.

## 2.2 Problems with bad post-processing and lighting

As previously mentioned, most gamers usually appreciate good looking graphics, but sometimes “good looking” might turn into a too overwhelming or indistinct looking environments. In many cases this is because of too heavy use of post-processing effects that obstruct the player’s vision and because of that, the gameplay becomes nauseating. Many times in modern video games, the player avatar’s eyes are treated as cameras and this makes many of the post-processing effects reasonable, but from a realistic standpoint this does not make any sense if the player avatar is not wearing any sort of glasses or lenses. The common effects that are relevant to this issue are usually the lens flare and the lens dirt. The lens flare effect happens when the player looks at a light source and this creates a light scattering effect on the screen and in some cases can be intrusive. The lens dirt is a similar looking effect where dirt particles are shown on the screen, but it can happen from a multitude of reasons. For example, in first person shooters it usually activates from a nearby explosions around the player or when the player is crawling in a pile of mud to enhance the feel of dirtiness. In both cases it would feel realistic if the player would wear glasses, but without them the screen just feels cluttered with effects (Senior 2015). The Picture 4 is from a game Battlefield 4, a first-person shooter developed by EA DICE and published by Electronic Arts in 2013. It was pretty well received by gamers, but received mixed feelings for the game’s overuse of different lens effects. In the Picture 4 the player is aiming towards a light source and this creates a lens flare around that light, but also adds some dirt effects on the screen. It is not helping that there is a pretty severe motion blur effect happening on the edges of the screen. When combining all of these effects together, it creates a pretty hard to read situation for the player.



Picture 4. A screenshot of cluttering post-processing effects in Battlefield 4 (Electronic Arts 2018).

The one effect that creates a lot of debate is bloom. The effect produces fringes of light extending from the borders of bright areas in an image, contributing to the illusion of an extremely bright light overwhelming the camera or eye capturing the scene. The artists really need to be careful when applying this, or otherwise it will transform every object in the game to glowing light bulbs.

The seventh console generation which began in 2005 was infamous for overusing the effect to the point that it was almost used in all games. This was due to consoles that could render graphics with high dynamic range, meaning a larger contrast between very bright and very dark objects. Before this, a normal white color was the brightest which the consoles could render, but with the high dynamic range capabilities the new consoles could now detect a difference in brightness between a white paper and a bright glowing light source (Gallant 2008). Below is a demonstration of overusing this effect, the Picture 5 is from a game Legend of Zelda: Twilight Princess, developed and published by Nintendo in 2006 for Wii video game console. This was right about the time when video games started to use this effect. Some might argue that this effect gives the game a “fairy tale” kind of look, but still it is pretty overwhelming when the bloom effect starts to radiate light from the flowers.



Picture 5. Demonstration of bloom gone wrong in the Legend of Zelda: Twilight Princess (Gallant 2008).

Fortunately game developers have mostly learned from their mistakes and nowadays the bloom is treated more carefully, but there are still some cases of using the bloom very heavy-handedly. The key is being subtle when dealing with post-processing and usually it takes many iterations to make it right.

### 2.3 Key features towards a successful post-processing

The post-processing cannot fix everything. As previously explained, too much post-processing can intervene with the gameplay negatively by obscuring the players vision of the game world. This means that the post-processing cannot just magically fix the game's lacking graphics by increasing the amount of post-processing. The other aspects mentioned are the 3D models, textures, lighting and the general layout of the game environment. In order to create a good looking game, all of the graphical aspects need to be on the same level. The post-processing is used for polishing the game-environment and is not supposed to be the main ingredient for the graphics. This is because post-

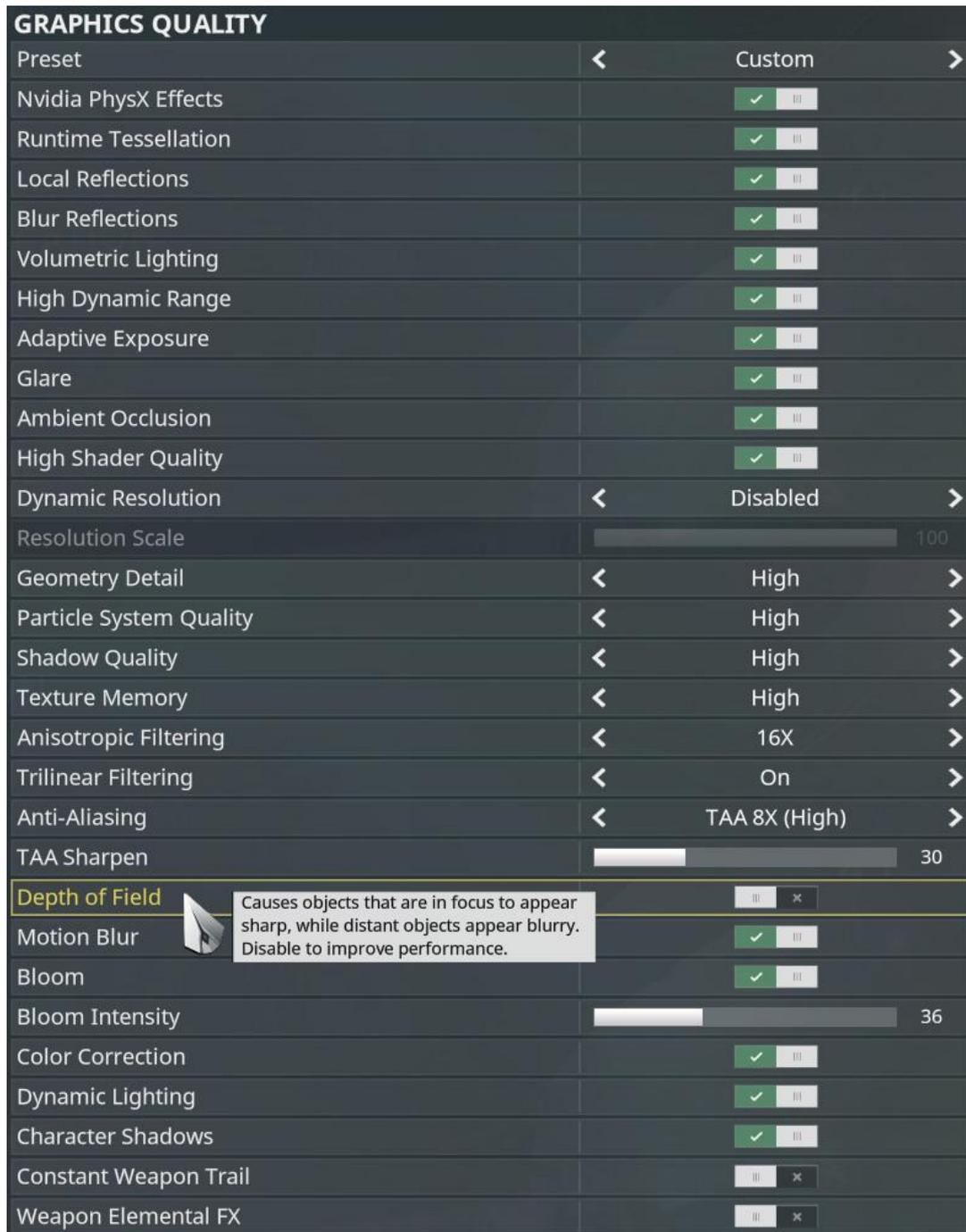
processing is not about adding content to the game environment, it is about modifying the content that is already in the game. (Dries 2016.)

Because video gamers are different, everybody has their own taste for graphics. That is why it is a good to have graphics options for games. Graphics options are mainly used to make games scalable for as wide range of systems as possible, because a five year old PC will not perform as fast as a brand new PC, but it still could run the game if the graphics can be toned down. Also, from the game publishers side, they want to make as much money as possible, so the games cannot only run on brand new PCs or the publishers would lose a large chunk of their audience.

Many games are also multi-platform and this means that they need to work on the consoles as well on the PC. Because consoles are released in cycles, everyone who buys a PlayStation 4 for example, will have the same PlayStation 4 even if they buy it a few years later. This means that developers who develop games for consoles do not have to worry about scalability of the game. This is good, but it usually means that the console versions are lacking completely the graphics options. So if a player does not like the post-processing effects on consoles, there is nothing the player can do to disable them (Morgan 2016). On PC this is completely different, because the developers have to put the graphics settings in the games to have the ability to scale down the most taxing visuals. This usually means that the PC gamers will also receive the ability to adjust the post-processing effects. At this point games have become much more customizable experiences and this is good for the players who could have nauseating side effects from the most intensive post-processing effects. A fitting example of this is a game called Warframe, a free-to-play cooperative third-person shooter, developed and published by Digital Extremes in 2013. It has high speed action gameplay that can cause nausea for some, but also very good graphics options from where the players can turn off most of the nauseating effects if needed.

For most gamers who do not understand the technical aspects behind each graphics settings, changing each of these settings to their liking might be an intimidating task (Lane 2017). That is why many players will not adjust them and instead just rely on the game's default graphics settings or let an automatic detection adjust the settings accordingly. In some cases this might be enough but if the player suffers from a nausea, caused by one of the game's effects, the game cannot know it and automatically turn the effects off. That is why more and more games have started to include descriptions for each of the graphics settings, what the setting does and does it have a large impact on

the performance. Some of the best graphics options have also a demonstration picture, how the setting is going to change the graphics. The Picture 6 below shows the graphics menu from Warframe which has lots of different settings for the players to change and also each of the graphics settings have a description if the player hovers a mouse pointer on top of it. Some of the effects even have sliders to give more options than just turning the effect on or off.



Picture 6. A screenshot of the graphics options in a game called Warframe (Digital Extremes 2018).

Making a good post-processing can sometimes feel very ungrateful, because if the post-processing is even slightly off, the response from the players can be very negative. Still the post-processing has to be used, because without it the graphics look unfinished. So, what is good post-processing then? It is a combination of many effects to create subtle

graphical enhancements that fit the style and theme of the game and does not intervene with the gameplay. It also should be highly customizable by the players to alter the style and performance to their liking. For the nausea causing effects, the game developers should recognize these and make the effects to be able to be turned off. So to make a good post-processing for a game that is appreciated by the majority of the players, the developers need to take account many small aspects and have a great artistic vision.

## 3 POST-PROCESSING EFFECTS

There are many different post-processing effects for artists to use. The amount varies depending on which game engine is used. For example, Unity and Unreal Engine 4 do not share all the same post-processing effects, but still enough that in most cases it does not matter which engine to use. However, even if both of the engines support, for example, a bloom filter it does not mean that it will look the same in both engines. The age of a game engine also directly reflects on the amount of post-processing effects it has and how refined those effects are.

Now it is the time to investigate these different post-processing effects, what do they do, and what are the benefits and drawbacks when using these effects. But before explaining these individual post-processing effects, one subject matter that needs to be understood, is shaders.

### 3.1 Shaders

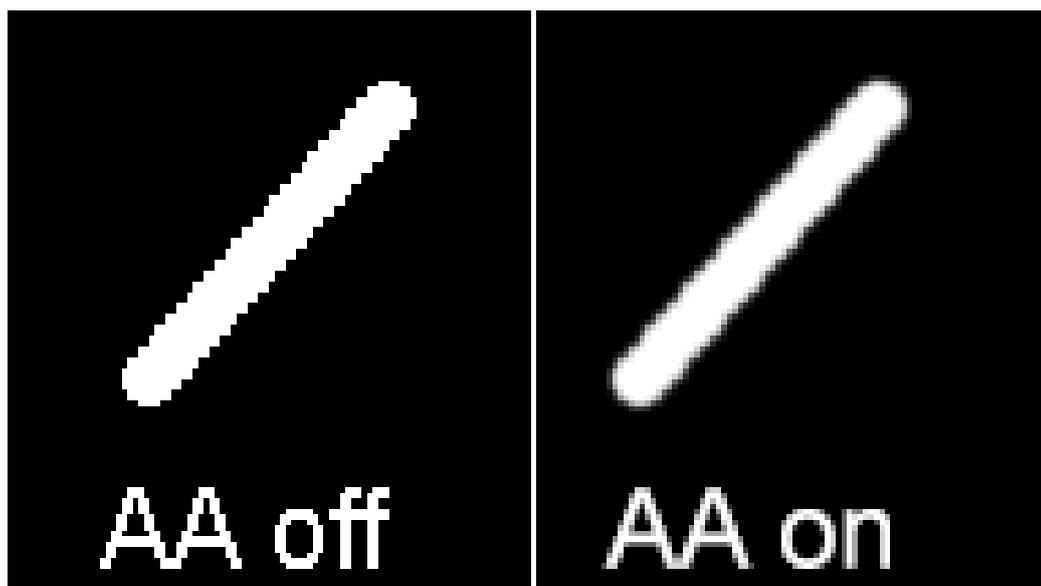
Shaders are used to alter the color, the lightness and the darkness of an object/surface/polygon in 3D graphics. Shaders can simulate how different surfaces react to lights or create reflections. They also hold the data about all of the material properties (ShaderCat 2016). So basically the shaders are responsible how the graphics look in video games and this is why it is important to have a basic knowledge of them related to post-processing. There are multiple types of shaders for different uses: vertex shaders, fragment shaders, surface shaders and geometry shaders, etc. But in this thesis, only the vertex and fragment shaders are examined. This is because, for example, surface shaders and geometry shaders are used to manipulate the geometry of objects, to create textures and other object related effects. The shaders that are used in post-processing are mostly fragment and vertex shaders. (Zucconi 2015.)

Vertex and fragment shaders work close to the way the graphics processing unit (GPU) renders triangles. Individual triangles are passed through a function which decides the final red green blue (RGB) colour for every pixel. They are useful for 2D effects, such as post-processing and special 3D effects which are too complex to be expressed as surface shaders. (Zucconi 2015.)

To apply a global effect on the whole scene, as post-processing, there is a limitation. All the shaders work locally, vertex shaders only know about the current vertex, and fragment shaders only know about the current pixel. The only exception is when working with textures. In this case, any part of the texture can be accessed using texture coordinates. So the idea for post-processing is to first render the whole scene in a texture, and then render this single texture to screen with the post-processing. There are also two main alternatives, rendering the screen for a first time, then copying the screen to a texture. The second alternative is to render directly to a texture through a framebuffer object. Using the second method, which is more efficient, can render on an area larger than the physical screen if necessary. The thesis will not go into further detail about shaders, since they are really complex and they are not the main point of the thesis. (Wikibooks OpenGL 2018.)

### 3.2 Anti-aliasing

Screens render the graphics in pixels and each of these pixels have a rectangular shape. This means that even the most organic looking shapes are rendered in squares. The problem is that whenever a computer tries to render rounded shapes, they need to be converted to pixels. This leads to jagged edges called aliasing. Anti-aliasing was made to counter this problem. The general idea behind Anti-aliasing is to smooth out the jagged edges by blending the colors of the pixels around the jagged edges to create an illusion of smoothness. Picture 7 below helps understanding how the anti-aliasing works on jagged edges.



Picture 7. A demonstration how anti-aliasing affects to jagged edges. Left side is without anti-aliasing and the right is with.

However there are many different styles of anti-aliasing and they vary greatly in performance, quality and efficiency. The thesis is not focusing on the multisampling anti-aliasing (MSAA) and supersampling anti-aliasing (SSAA), because they are not post-processing based, but simply put they render the game on a higher resolution than the display resolution and then downscale it to the native resolution. SSAA is the most performance heavy anti-aliasing, but also looks the best. (Wilde 2014.)

### **Fast Approximate Anti-Aliasing (FXAA)**

Fast Approximate Anti-Aliasing is the most performant type of anti-aliasing on the game industry. When using this anti-aliasing there is not really any significant impact on the performance. FXAA is a pixel shader program that completely ignores polygons and line edges to simply analyze the pixels on the screen and then it smooths out the edges of each of those pixel. The downside to this is that the anti-aliased image is more blurred than in other anti-aliasing types. In some cases this might be good, but in situations where the players need to see small objects on distance, the FXAA can blur those details too much. (Atwood 2011.)

### **Morphological Anti-aliasing (MLAA)**

MLAA was first designed for the CPU and back then it could not run in real time, so it really wasn't for gaming, but now the GPU developed counterpart is very fast and effective choice of anti-aliasing. MLAA is post-processing based anti-aliasing effect, it identifies patterns which have large variations in colors to blend pixels close to these patterns. MLAA is also very light on the performance side, but not as fast as the FXAA. However, there are some downsides to MLAA, it has difficulty to handle pixel-sized features and it can also slightly distort textures by removing detail from them. So in general, MLAA is not as blurry anti-aliasing as FXAA, but it can still cause some problems when it tries to work with pixel-sized features. (Jimenez 2011.)

### **Enhanced Subpixel Morphological Anti-aliasing (SMAA)**

A post-processing based anti-aliasing which is an improvement over the MLAA and combines strategies from MSAA and SSAA, without the heavy performance impact. The MSAA and SSAA techniques are used to improve the subpixel features of SMAA. It also has a better edge detection than the older MLAA, due to the improved local contrast detection. The cost of using SMAA is very low, making it ideal for low-end systems. (Jimenez 2012.)

### **Temporal Anti-Aliasing (TXAA)**

TXAA is a new style of anti-aliasing, which is usually compared to the film industry due to the efficiency of this anti-aliasing. The result produced by TXAA is much smoother than any other anti-aliasing available today. One downside to this is that when comparing it to the previously explained anti-aliasing methods, it is much more performance heavy. Still, it is more efficient than trying to achieve the same results from SSAA or MSAA. What needs to be kept in mind is that TXAA will not run on the older hardware, only on NVIDIA's Kepler GPUs or higher. (Nvidia.)

## Conclusion

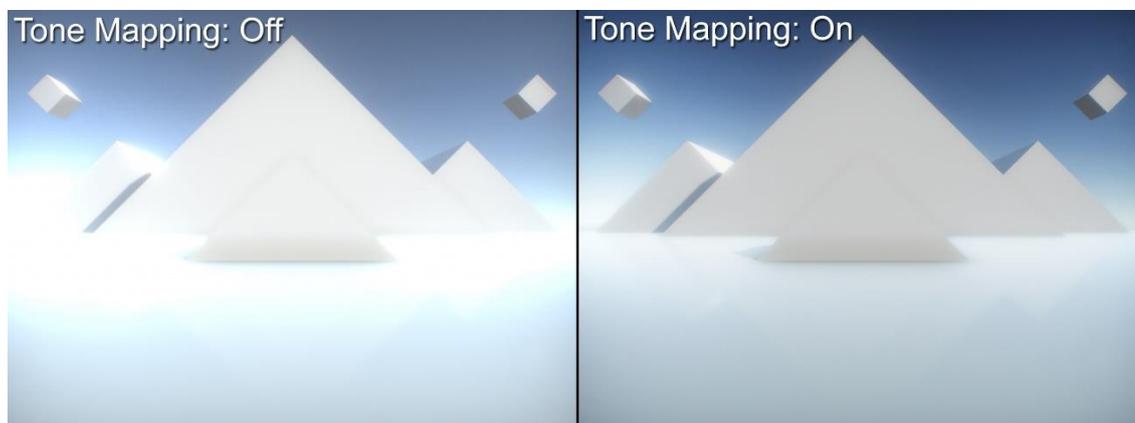
As explained, there are lots of different choices for selecting the desired anti-aliasing, depending if the player wants more of a performance or a quality choice. FXAA is for the lower-end systems, while MLAA and SMAA for average systems and finally TXAA is the high-end solution for aliasing. The older techniques such as SSAA and MSAA are not used so often anymore due to the heavy performance impact and that is why the more robust post-processing based anti-aliasings are taking over.

### 3.3 Tone mapping

Tone mapping is the process of mapping a high dynamic range (HDR) rendered image back to the more limited low dynamic range (LDR) color spectrum. This is implemented because showing a HDR image on a LDR monitor makes the brightness and the contrast of a image look incorrect. A HDR image usually looks very greyed out when it is showed on a LDR screen without tone mapping, because the LDR screens cannot show the darker and brighter HDR colors out of their spectrum. (Unity documentation 2018.)

There are multiple ways to do tone mapping, using premade operators or creating a custom one. When creating a custom one, it also gives the freedom to adjust the lighting the way the game developer wants to. Using one of these ready operators might be an easy choice, but it does not always match what the game tries to look like. However, many of the premade made operators are highly praised and in large use in the game and film industry. These include operators such as: Linear, Reinhard, Haarm-Peter Duiker's Curve, Jim Hejl and Richard Burgess-Dawson, ACES. The operators are algorithms that are used to achieve the tone mapped result in a different ways. The operators do this by adjusting the contrast, gamma and exposure values differently, these values are responsible how bright something is in different light levels. (Hable 2010.) The intention of these operators is that they are made by the professionals and they have been used in the both industries for a long time. So whenever someone watches a film there is a large chance that it is using one of the tone mapping operators, and in an age where video games want to look as cinematic as possible it is almost certain that they are also using those operators.

Picture 8 demonstrates how custom tone mapping brings the overexposed HDR back to LDR when using a normal LDR monitor. The tone mapped version brings down the very bright areas so they will not glow in the sun like in the non-tone mapped version.



Picture 8. Effects of HDR rendering on a LDR monitor, with and without tone mapping.

### 3.4 Color Grading

Color grading is usually implemented later in the project, usually in the polishing phase. However it does not mean that it is less important than the parts that come before it. Color grading is the best post-processing tool to affect the mood and the theme of a game. So what is color grading? Color grading is a process that manipulates the color channels of a game in different ways. For example, if a game is set in a cold environment, the color grading can give it a blue tint to visualize the coldness of that place. What color grading can also do, is to apply colors to a specific brightness level, this means that it can very effectively affect the mood of the game, if it should be scary, calm, gloomy, etc. The color grading is more effective when a game itself does not have that much saturated colors or high contrasts. This gives an artist more freedom to choose how a scene will look like exactly. (Maslanka 2017.)

Color grading shouldn't be confused with color correction, as the name implies color correction is used to fix imperfections in an image. These can include problems such as, too dark shadows, over bright light sources, or some color that does not look how it should. Color grading is then used to change the artistic side of an image, this usually means that the changes might not be realistic and they are more extreme than in the color correction phase. (Maslanka 2017.)

Color grading is one of the most artistic phases in the post-processing, because there is so much the artist can do. All the other post-processing effects are not nearly as time consuming as color grading. In the Picture 9 below is a demonstration in Unity how the color grading affects the scene. The main difference is that the color temperature is lowered to give it a bluish tone and the tint is also adjusted to give it a sunset kind of feeling. There are also lots of smaller adjustments happening, such as channel mixing and using the trackballs to color grade on different brightness ranges within the image.



Picture 9. A demonstration how the color grading works in Unity. The left side is without color grading and the right is with. On the very right side is the actual color grading tool.

### 3.5 Depth of Field

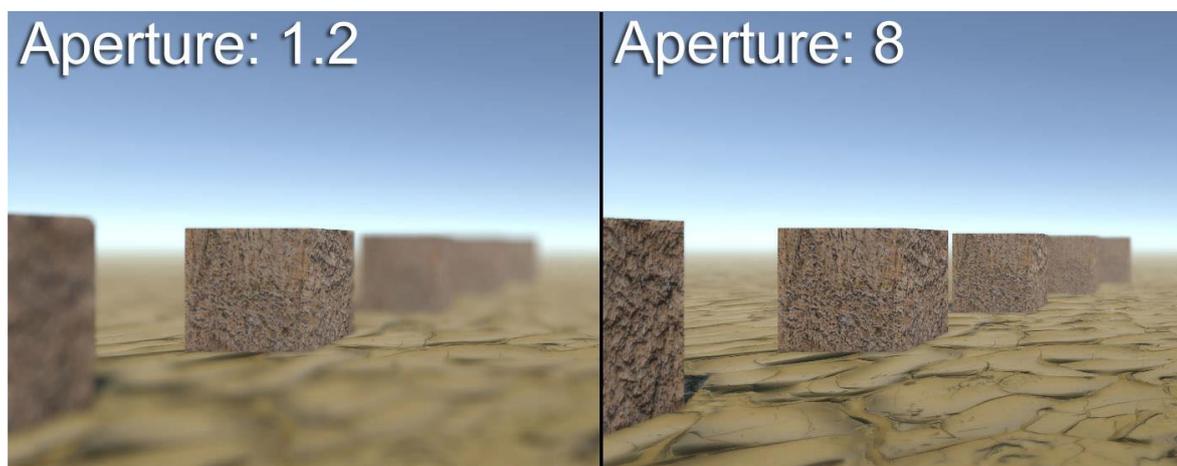
Many video gamers might be familiar with depth of field, which is the effect that blurs all of the game's graphics and gets in the way of the gameplay, or at least that is what they think. Depth of field is a great tool to affect where the player's focus should be, and it can also hide visuals with its effect so the game developers do not have to put tons of hours to build highly detailed backgrounds. Hiding the less detailed background graphics is also good for the optimization of a game. Game cinematics or cutscenes are also a great place to use depth of field for creating a cinematic look.

How does the depth of field work then? Real cameras can only focus to a single point and an area around this point. Anything inside this area appears sharp while everything outside of this area is blurred out. This area is called depth of field. To adjust this area a setting called aperture needs to be adjusted. Aperture controls how large the area is around the focus point. For example, if two characters are having a conversation in a game cutscene it is usually most effective to have the focus on them and use a lower

aperture to blur out everything around them. In turn when showing a larger landscape, the aperture should be much higher to capture the whole view. (Hawkins 2018.)

However depth of field is something that does not need to be used, because games do not share the same limitations as real cameras. This means that the whole view can be in focus if the developers choose to do so. A good alternative is to use a contextual depth of field, so the depth of field is not always enabled and for example, only in use when a cutscene is playing or when the player inspects an item in their hands. One usual place to see depth of field is when an user interface is activated, this usually happens by blurring the whole background and then only the user interface is clearly seen.

The Picture 10 below demonstrates how changing the aperture affects the depth of field effect. The lower aperture of 1.2 on the left side makes the focus area much smaller and only the second box is in focus while the rest are blurred out. The aperture is 8 on the right side so the focus area is much larger and almost all of the boxes are in focus, leaving only the furthest boxes outside of the focus area. Neither of the examples is wrong and it is up to the artist to choose which one suits their vision of the scene.



Picture 10. A demonstration that shows how changing the aperture affects the depth of field. The left side is with an aperture of 1.2 while the right uses a value of 8.

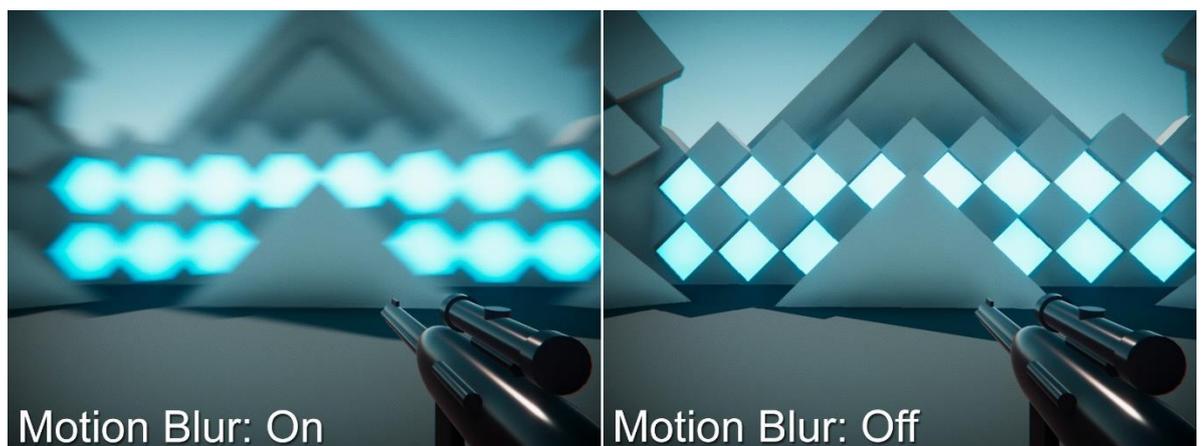
### 3.6 Motion Blur

Motion blur is the effect when something is moving rapidly on the screen and leaving streaks behind it. It can also be the fast camera movement and in which case it blurs the whole screen. In real life motion blur happens if an object is moving faster than the camera's exposure time, but in games it does not happen and needs to be added as a

post-processing effect (Unity Documentation 2018). In case of motion blur there should be always an ability to toggle motion blur off, because it is one of the effects that has the highest chance to make someone feel motion sick, which can make them stop playing the game (Wahab 2017). So, why is motion blur used if it can have such a severe impact?

Motion blur can be used for many positive purposes, for example, console games are usually running on lower frames per second (FPS) than on PC, due to weaker hardware they have. This means that the highest FPS that console games are capable of being is 60 and in many cases it is just 30. Lower the FPS goes, the easier it is to see the individual frames, but when motion blur is applied to the image, it smooths out the camera movement and makes lower frame rates seem higher than they actually are. On PC when playing games on an unlocked frame rate, in some cases the FPS might even get close to a 300. This means that there is no reason to use motion blur for smoothing out the image, but it can still be used to achieve something artistically pleasing. This can be something such as making a player feel as if they would be in a really fast moving car or a game's world is moving so fast that it causes motion blur. There is not a better way to visualize speed than motion blur and that is why it is used in so many racing games.

The Picture 11 demonstrates how a camera motion blur affects the surrounding game environment in Unity. The player's view is rapidly rotated to the right and this creates a horizontal motion blur effect for everything else than the weapon which the player is holding. The weapon is not blurred because it stays still relative to the camera, while everything else is moving from the camera's point of view.



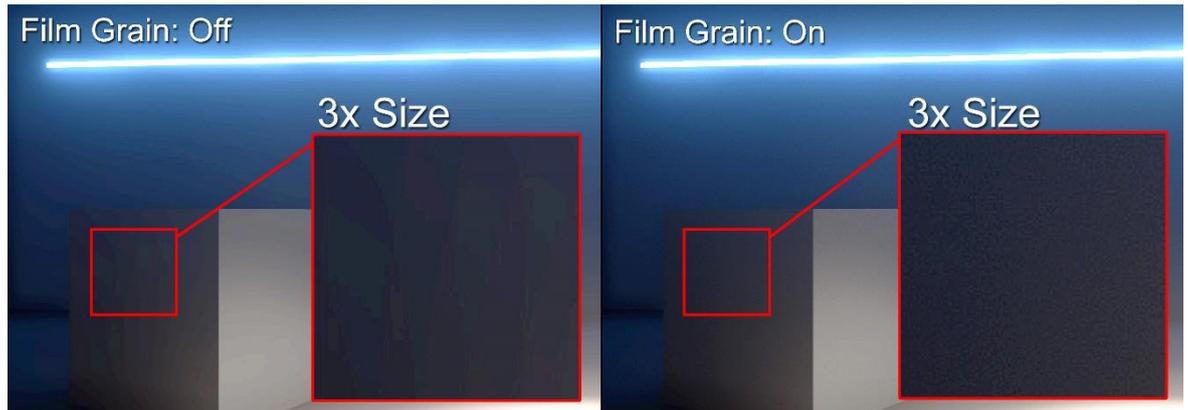
Picture 11. A demonstration of motion blur affecting rapid camera movement in Unity game engine.

### 3.7 Film Grain

As motion blur, film grain is mostly used to achieve a cinematic look for a game and it is even more present in a games that try to mimic the feel of the older cinema. When using cameras with real film, the film grain is caused by “an optical texture of processed photographics film due to the presence of small particles of a metallic silver, or dye clouds, developed from silver halide that have received enough photons” (Wikipedia 2016). When talking about video games this is not the case, because they are digitally made. So more accurate definition for film grain in video games would be digital noise. The difference between film grain and digital noise is that where film grain is something physical on the film, the digital noise is applied in the post-processing as a moving noise pattern where the individual grains are measured in pixels. Modern digital films also use digital noise to make them feel as they were shot on film.

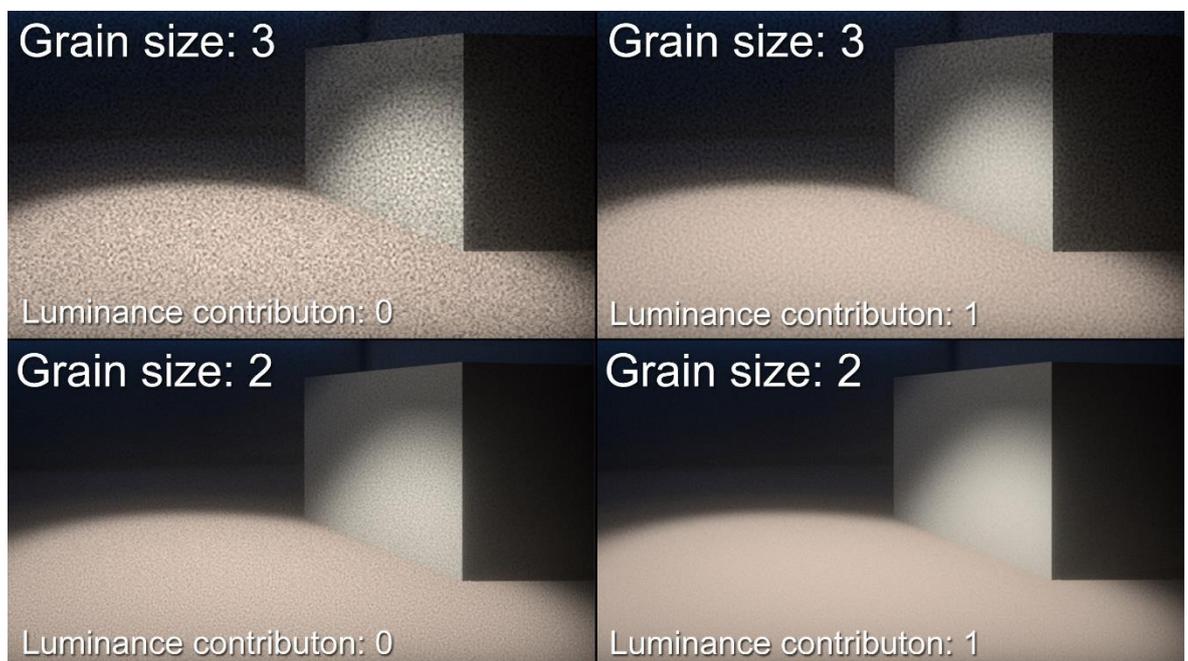
Digital noise usually comes in two types, colored and desaturated. The size of the noise pattern can be adjusted to suit the needs of the theme. It is pretty common to see large digital noise patterns be used in horror games to make the players feel unease. Other adjustable settings usually include features such as the amount based on the luminance, which means that digital noise can be controlled to be shown differently in the dark and bright areas. This is good because if a game environment is very dim, the digital noise does not have to be so intensive or otherwise it might create a very tiresome experience for a player to look at.

Digital noise can also be used to hide unwanted color banding effect that is caused by gradients, such as a light source that has a soft falloff radius. Color banding causes clear lines to appear around these light sources instead of a seamless gradient (Photographylife 2018). Below is a demonstration of this effect and how digital noise fixes it. The left side of the Picture 12 shows how the color banding lines are very noticeable on the box and on the wall behind it. While the effect is reduced next to nothing when looking the right side of the Picture 12 with digital noise enabled.



Picture 12. Digital noise reducing the color banding effect in Unity.

The Picture 13 is a demonstration how different settings adjust the visibility of the digital noise. Starting from the top left corner of the image, the grain size is set to 3 so the effect can be easily seen to demonstrate the differences. The luminance contribution is set to 0 and this can be seen from the large amount of noise in the light, while on the right side with the same grain size the bright area has much less noise applied to it. The two pictures on the bottom are taken with a smaller grain size and lower overall intensity of the digital noise. As can be seen from the bottom right picture, it has the most natural noise compared to the rest of the pictures.



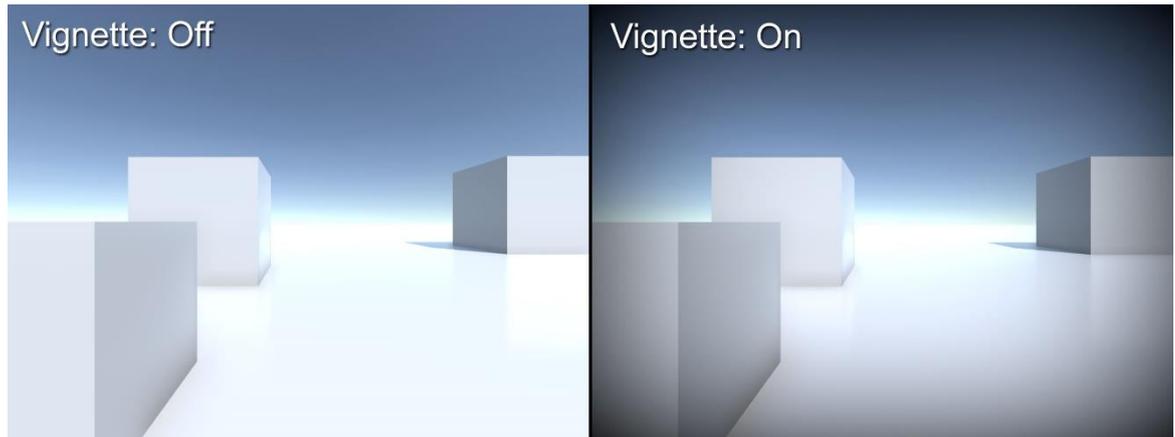
Picture 13. A comparison between different digital noise settings in Unity.

### 3.8 Vignette

Vignetting is an effect that darkens the corners of an image while leaving the center intact. Vignetting is also an effect that happens naturally in real life cameras, but in video games it is available through post-processing. So what causes vignetting in real life? The reason for vignetting is that when light hits the camera's sensor, the light needs to travel longer when trying to reach the edge of the sensor than when it tries to reach the middle of the sensor. This naturally dims the light on the edges of the sensor and creates the optical vignette effect. Video games have to rely on artificial vignetting, but this does not matter because it looks the same as the vignette caused by the real cameras. Artificial vignetting is also good because the artist can set the effect, just as they want it. (RED Cameras.)

When the vignette effect is purposefully added it can help the viewer to focus more to the center of the screen and not the edges or it can be used for artistic reasons (Mansurov 2018). In video games it can also be a powerful tool. For example, in Unity all of the post-processing effects can be modified in real-time. This means that when a person is playing a video game, all of the effects can be changed gradually from the player's inputs or from anything that is happening in the game world. A good example of this would be, when a player enters a darker area such as an interior of a building or a cave, then the vignetting effect would increase its intensity to make the player feel claustrophobic. Another good use of the vignette like effects in video games, are the status effects when the player is for example, taking damage, or otherwise in a bad shape. The vignette effect can then color the edges of the screen in the most appropriate color for that situation. This is a small cinematic touch and sometimes it removes the need to show the status information for the player as a text, or in an icon on the user interface.

The Picture 14 below compares how much vignetting can affect the look of a simple scene. In this example the vignette effect is very gradual and it takes a while before it fades, but the middle of the screen is still clearly visible. Normally when adding vignetting, the amount should be pretty subtle, but the large amount of vignetting in the Picture 14 is there just to visualize the effect.



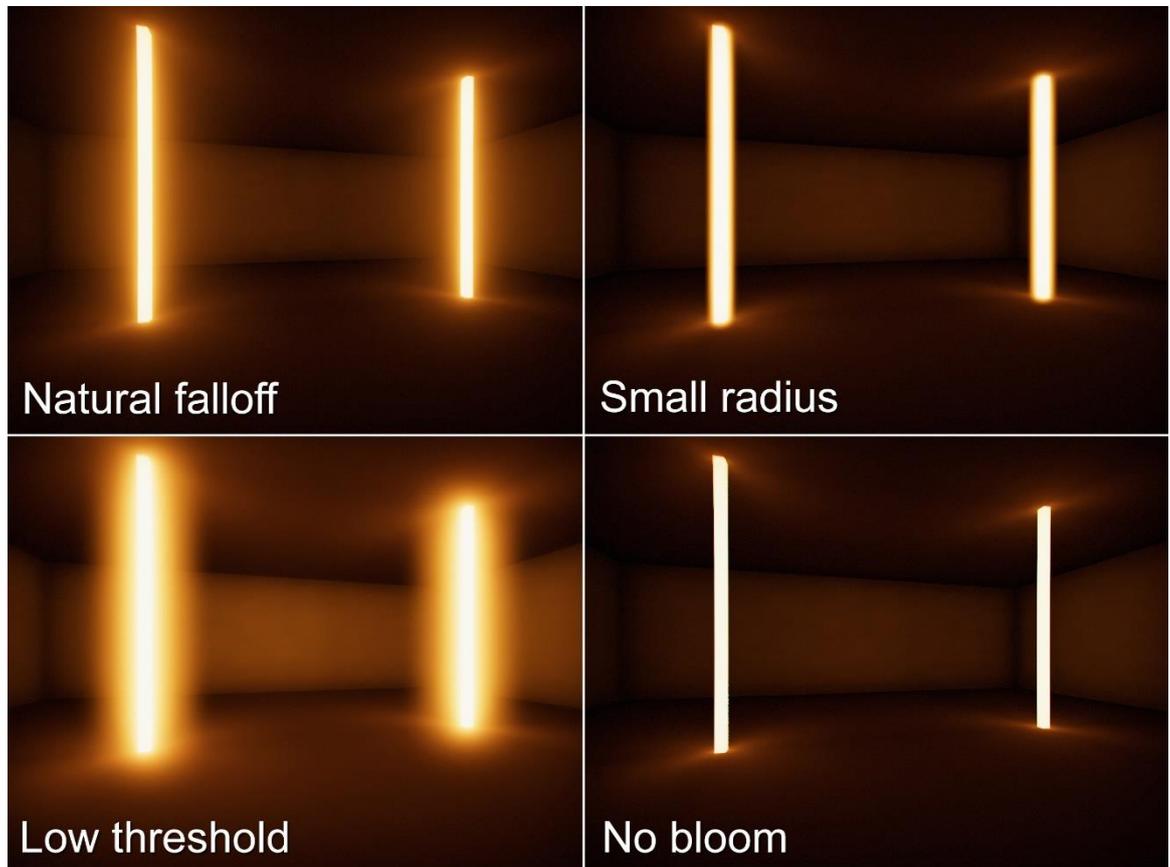
Picture 14. A demonstration how vignette effect looks in a bright test environment in Unity.

### 3.9 Bloom

As previously explained in the Chapter 2.2, bloom can be a very hard effect to do right. When applying too much of it, it makes games look very hazy (Gallant 2008). On the other hand, not using bloom makes a game's lighting look unnatural. So it is all about finding the right balance between the two. In modern video game engines, using bloom means more than just adjusting the intensity. For example, setting a right bloom threshold is crucial for different lighting conditions. The threshold should be always adjusted so that the brightest light source is visible. Threshold is the level from where the light sources check if they are brighter than it, and if they are then the bloom will start affecting them (Unity documentation 2018). The range of the glow around the bright areas can also be adjusted. The amount of range is up to the artist, but generally it is a better idea to use wider range and low intensity compared to a small range and high intensity. This way the light glows into the environment more naturally and achieves the purpose of bloom.

In the Picture 15 bloom settings are compared in Unity. Starting from the top-left corner, which is the most natural bloom effect in the comparison. It has low intensity and large radius which makes it blend into the environment naturally. The problem with the bloom effect at the top-right corner is that the radius is so small, it does not look as it would be glowing. In the bottom-left corner the effect suffers from too small threshold which causes the whole scene to glow even though there are only the two light sources that should be

glowing. The bottom-right corner image is there to demonstrate how large impact bloom has for the overall visuals of a game.



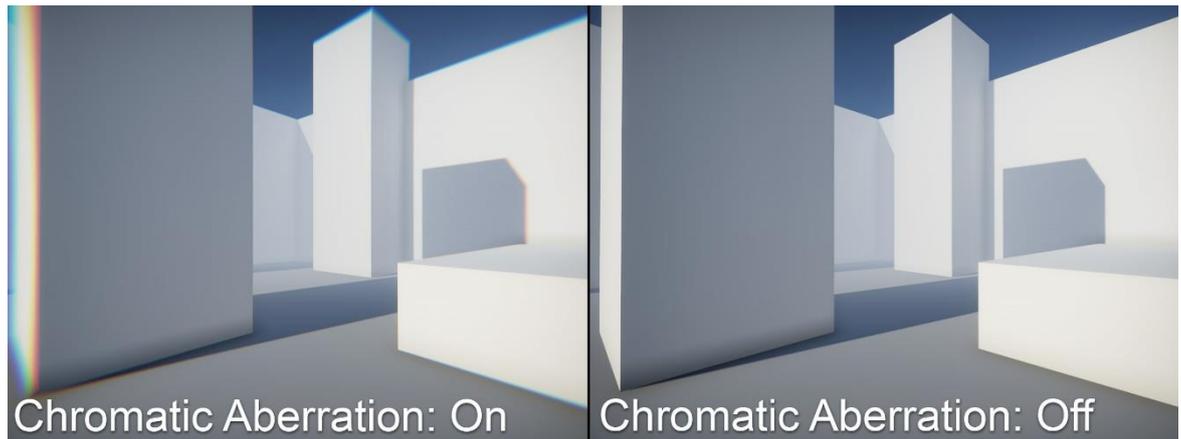
Picture 15. A comparison between different bloom settings in Unity.

### 3.10 Chromatic Aberration

In real world chromatic aberration occurs when a camera lens bends the colors of light at the focal point. This causes a mismatch which prevents the colors combining as they should. The effect can be seen as fringes of colors around objects and it is more noticeable on the edges of the image. Many times chromatic aberration is considered as a problem and that is why photographers try to minimize it by using better lenses or fixing it in post-production, meaning that it is fixed by a software after taking the picture. (Pluralsight 2015.)

However chromatic aberration has always been in films and pictures, so by fixing this imperfection the image just does not look the same. Keeping the small imperfections in the image makes it look more real and interesting (Pluralsight 2015). Chromatic

aberration, similarly to vignette is a good tool to display impact and intoxication effects for the player. For example, in Unity the effect's intensity can be changed in real-time so while the player is damaged or otherwise hurt, the effect can be gradually increased and then toned down after the situation. The Picture 16 demonstrates how chromatic aberration affects the scene by adding small fringes of colors on the edges of the objects. The chromatic aberration effect is exaggerated in the Picture 16 to demonstrate the difference.



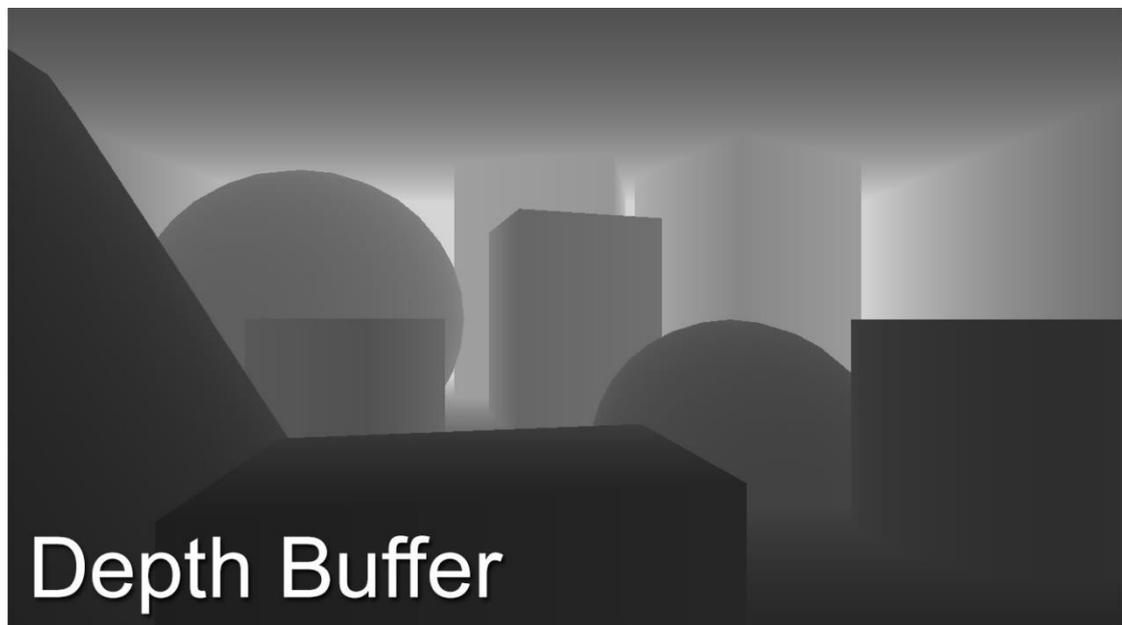
Picture 16. A demonstration of chromatic aberration effect in Unity.

### 3.11 Ambient Occlusion

Ambient Occlusion is used in rendering softwares and in game engines, it is a technique used to improve soft shadows in small details, such as holes and creases. The effect is achieved differently depending on the situation, in game engines ambient occlusion needs to be implemented in the post-processing while rendering softwares can calculate the ambient occlusion based on the geometry (Unity documentation 2018). Because the geometry based ambient occlusion is not related to post-processing that method will not be explained in detail in this thesis. A quick rundown how it works, each surface in the level geometry checks if there is any other surfaces in front of them, and if there is then that surface will become darker. This is much more accurate than the post-processing approaches, but too heavy on the performance to be used in real-time (Pluralsight 2014). Before going into detail about the different ambient occlusion types, one technique needs to be understood called depth buffering, also known as z-buffering.

## Depth buffering

Depth buffering is a graphics programming technique which is used to determine whether an object, or part of that object is visible in the scene. When a scene is rendered, all pixels in the scene have an X and Y coordinate and also a Z coordinate, the Z coordinate is used to determine the distance from the camera. The depth buffer is used to store each pixels Z-value in a two dimensional array. When a new object needs to be rendered in the same pixel location that already has an object, the depth buffer checks if the new pixel is closer to the camera and overrides the previous value. (Computer Hope 2017.) Picture 17 is a visualization of a depth buffer, the darker pixels are drawn on top of brighter pixels.



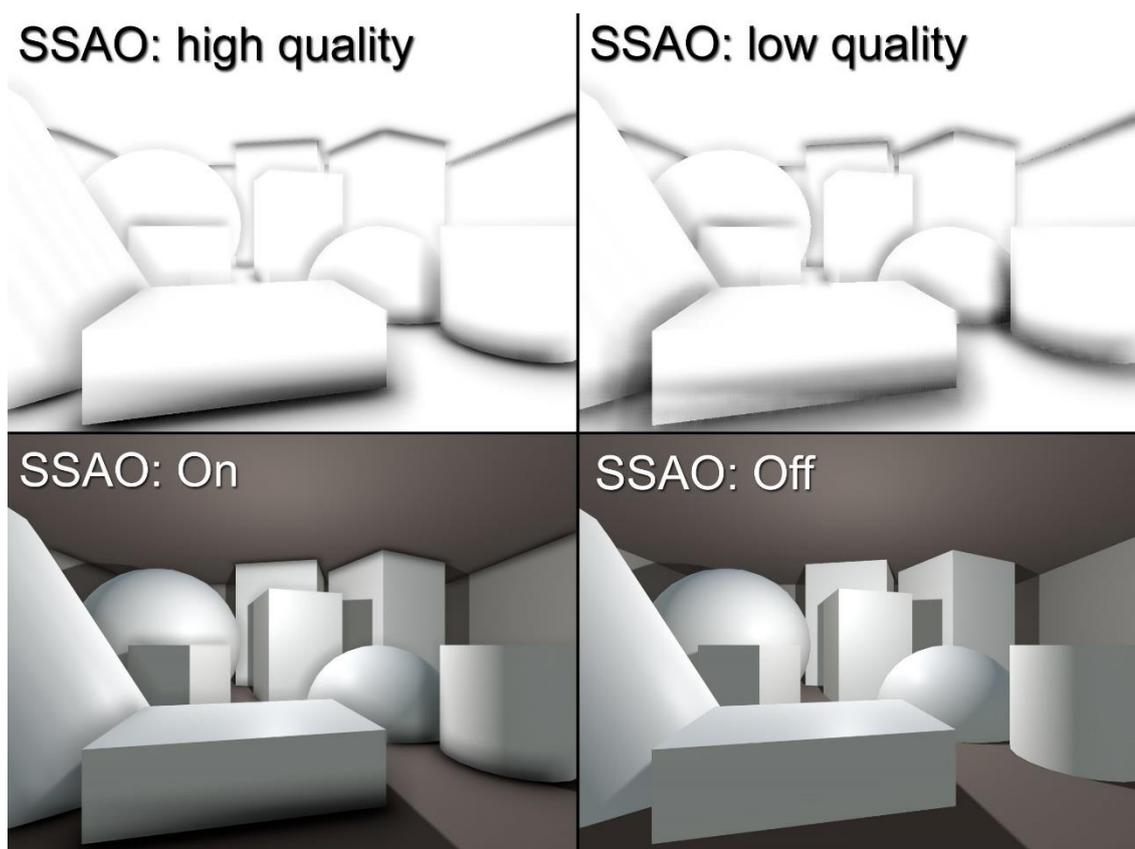
Picture 17. Visualization of a depth buffer in Unity.

## Screen-Space Ambient Occlusion (SSAO)

The idea behind SSAO is that instead of calculating the ambient occlusion based on the geometry, the depth buffer can be used to determine the amount of ambient occlusion something has. SSAO takes each pixel on the screen and checks if the surrounding pixels have a different depth value and based on that difference it creates the occlusion effect between them. For real-time rendering this method is still too slow and to achieve a solution that would run on modern PCs, a different approach is needed.

In real-time rendering SSAO has to use kernels instead of single pixels, kernels are groups of pixels. These kernels are then repeated only for some of the pixels which reduces the performance costs. Due to using kernels the result is much more noisy than calculating each pixel individually and that is why the noise needs to be blurred out before finally rendering it. (Learn OpenGL 2015.)

SSAO is the most performant type of ambient occlusion and it was also the first ambient occlusion to be using the depth buffer as its base. This also makes it the first post-processing based ambient occlusion to be used in video games. Disadvantages of using SSAO is its graininess compared to the other types of ambient occlusions. In the Picture 18 is a comparison how SSAO affects the ambient lighting of a scene. The black and white pictures on the top row have only the ambient occlusion enabled for visualization purposes. The top-left version has more samples of SSAO and also it is using full resolution for the effect, while the top-right has less samples and half of the resolution. As can be seen, the lower quality version is much more noisier and inaccurate compared to the high quality. On the bottom row is a comparison how high quality SSAO affects the scene.



Picture 18. A comparison of different SSAO qualities. The two views on top have only the ambient occlusion effect displayed for visualization purposes.

### Horizon-Based Ambient Occlusion (HBAO)

HBAO is an upgraded variant of the SSAO type of ambient occlusion, developed by NVIDIA. HBAO offers a better quality ambient occlusion than SSAO, it increases the accuracy, quality and visibility of the effect. For performance reasons the HBAO is typically rendered at half-resolution of the game's native resolution. The lower resolution is known to cause flickering which is hard to hide in some situations. Due to these problems an improved version of HBAO has been developed by NVIDIA called HBAO+. It doubles the detail level of the ambient occlusion effect, it performs three times faster, and uses the latest technologies to produce this. The main benefit of HBAO+ is that it renders at full resolution instead of the HBAO's half resolution (Nvidia). AMD has its own version of HBAO called high-definition ambient occlusion (HDAO). HBAO and HDAO both work very similarly, but both work better on their own type of GPUs. That is why HDAO is not covered in this thesis.

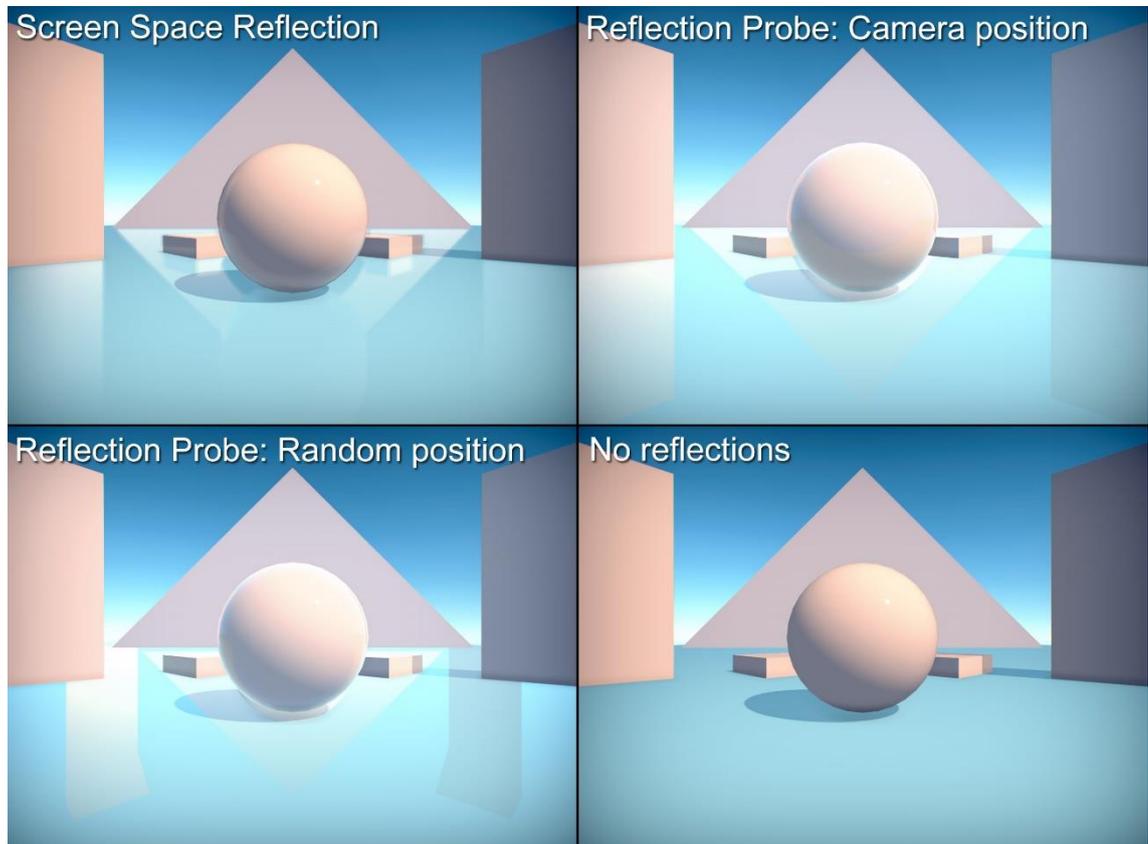
### 3.12 Screen Space Reflection

Reflections in video games have always been a challenge to create. They are very performance heavy and many times just look repulsive. For many years reflections have been achieved by reflection probes, these probes are simply put 360 degree cameras that are stored at desired positions in the game world. Reflection probes capture the view of their surroundings and that view can be used to give reflections to the objects that are in the area of the reflection probe. The main problem with reflection probes is that they need to be manually placed in the positions where the reflections want to be seen. Also, the reflections from the reflection probes are only accurate when they are seen from the exact position of the reflection probe (Unity documentation 2018). On larger surfaces this is a real problem because the reflection probe cannot cover all angles a surface can be looked from. Screen space reflection is made to counter this problem.

Screen space reflection is a dynamic real-time solution to reflections. The major benefit of using screen space reflection is that they will always show the reflections in the correct places unlike reflection probes. Screen space reflection also reflects dynamic objects, this means that objects that are moving in the game receive reflections and also create them. Because the screen space reflection operates in screen space this means that it will only reflect what is seen on the screen, this is good for the performance but also as a downside it cannot calculate anything outside of the screen or objects behind other objects. A good example of this is when a player is on top of a pool of water and looks straight down and no reflections can be seen. This is because anything above of the pool of water is not currently visible on the screen. Alongside with these limitations screen space reflections are also pretty performance heavy and should be only used on newer PCs and console platforms. (Shaw 2017.)

In Picture 19 is a comparison between screen space reflection and reflection probes in Unity. Starting from the top-left is the screen space reflection which is accurately showing all the reflections on the shiny floor. Next on the top-right corner is one reflection probe on the exact same position as the camera. This gives a pretty good result and all reflections are on the right positions, but it is missing the reflections from the sphere in the middle and from the lower platforms. However, the real limitations of reflection probes can be seen from the bottom-left corner, which shows what happens when the reflection probe is not on the same position as the player's camera, resulting into uncorrect

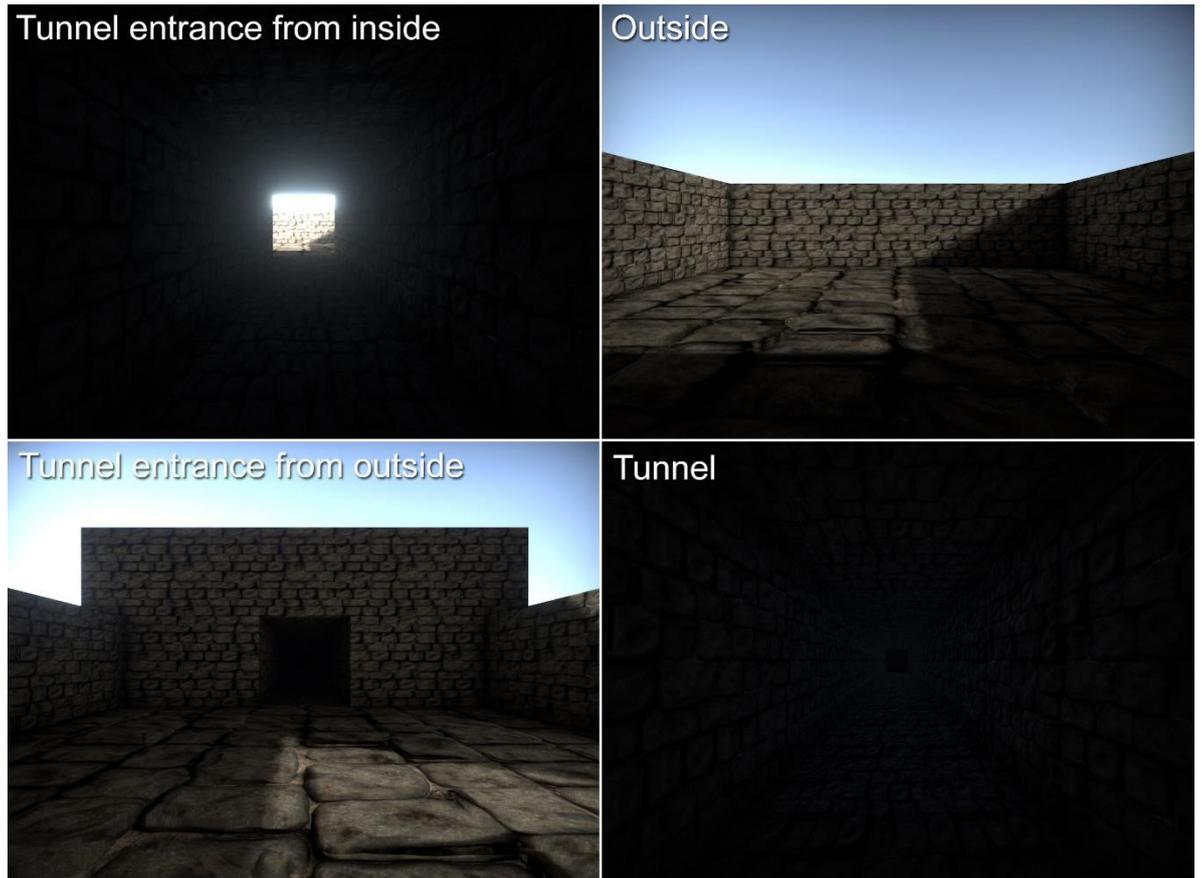
placement of the reflections. The bottom-right is a demonstration when all reflections are turned off.



Picture 19. A comparison between screen space reflections and reflection probes in Unity.

### 3.13 Eye Adaptation

“In ocular physiology, adaptation is the ability of the eye to adjust to various levels of darkness and light. The human eye can function from very dark to very bright levels of light. However, in any given moment of time, the eye can only sense a contrast ratio of roughly one millionth of the total range. What enables the wider reach is that the eye adapts its definition of what is black.” (Unity documentation 2018). Eye adaptation in video games tries to mimic this real world effect, by adjusting the brightness levels over time on transitioning between differently lit areas. The best example of this is when a player is in a dark tunnel and and in the other end of the tunnel is a way out. The way out looks like a glowing light source when viewed from the dark, but when the player reaches the exit the glowing light will gradually transform to a natural Sun light. This also means that the tunnel will look pitch black when seen from outside. Below in the Picture 20 is a demonstration of the example.



Picture 20. Eye adaptation effect at different brightness levels in Unity.

## 4 POST-PROCESSING IN DIFFERENT GAME ENGINES

Now that all of the major post-processing effects have been explained it is time to investigate how the post-processing differs depending on the game engine. The game engines used to compare post-processing are Unity and Unreal Engine 4. Both engines have almost the same post-processing effects, which makes them comparable. The aspects that are compared, are quality and performance. To make the comparison fair both engines are using the same scene as a base.

Another comparable 3D game engine that can deliver high quality graphics which is not used in the next comparison is the Cryengine, developed by Crytek. Cryengine was one of the first engines starting to push the graphics to the next level with a release of Far Cry, a game developed by Crytek in 2004, but the engine back then wasn't open to public. Since then the game engine has been updated and is now available to everyone the same way Unity and Unreal Engine 4 are, but the community around Cryengine is much smaller and that is why it is not used in the next comparison (CRYENGINE 2018). Other 3D game engines that match the power of Unity, Unreal Engine 4 and Cryengine are game companies in-house engines which means that they are not available to public.

### 4.1 Unity

Unity is a cross-platform 3D game engine released on June 8, 2005. Since then it has received constantly major updates and to this day is one of the leading game engines. Unity supports over 25 different platforms including mobile, console and desktop. Unity also comes with a friendly user interface for artists to use, so implementing features such as post-processing is very easy and does not need help of a programmer. Unity is a very efficient tool for prototyping, because of features such as the asset store which is full of community made assets, from which some are free and others paid. The assets can vary from singular 3D models to full systems and solutions for game development. Unity comes with a lot of other features that are not examined in this thesis because they are not related to post-processing or visuals in general. The post-processing used in the comparison is the official post-processing stack V2 developed by Unity Technologies. (Unity 2018.)

## 4.2 Unreal Engine 4

Unreal Engine 4 is also a cross-platform 3D game engine, it was released on March 19, 2014. Unreal also has a very similar update cycle compared to Unity, this means multiple updates each year which fix problems with the engine and add new features. Unreal is widely used in the game industry and even the larger game studios are using it instead of their own game engines. The engine supports over 15 different platforms including mobile, console and desktop. Unreal engine is reasonably easy to start using, but not as easy as Unity. Although adding the post-processing in Unreal engine is as easy as in Unity. The feature is always included in the projects and it only needs to be configured. Unreal's version of the Unity's asset store is called marketplace. It behaves the same way as the Unity's version, but it is more strict what can be shown there. For example, almost all of the free assets are from companies who want to support their software inside Unreal Engine, and the paid assets need to be on a certain level. This way Unreal's marketplace is not so bloated with inferior assets. Unreal also comes with a lot of features and in many cases even more than Unity, but the thesis will not examine those details. (Unreal Engine 2018.)

## 4.3 Benchmarking

The setup for both engines before the post-processing was almost identical, both were using the same 3D models and textures as the game world. The lighting is also implemented the same way. Only the Sun is used as a light source and there is one reflection probe at the same position as the camera. This is the basic setup that both engines received before the post-processing. For the post-processing both engines are using the same effects, but they look different because of the differences in the engines. Effects that are not used in the comparison are, motion blur and eye adaptation. This is because in a comparison where the player camera is not moving, neither effect would have any impact. Rest of the effects that were explained in the Chapter three are all used.

In both engines two post-processing profiles were made, a high-quality and a low-quality. The main difference between the profiles is that the high-quality is using depth of field and screen space reflection. This is because both effects can have a large impact for the performance. Other smaller differences include TXAA and better looking SMAA in the

the high-quality profile, while the low-quality is using FXAA and SMAA with smaller sample count.

To do the benchmarking a software called Fraps by Beepa was used. Fraps is a benchmarking, screen capture and screen recording tool for Windows. It was released on August 25, 1999 but its latest stable build was released on February 26, 2013. For benchmarking, Fraps can calculate average FPS, minimum FPS and the maximum FPS for the user defined time. The result will be logged into a text file where it can be examined.

In the next table are the results of benchmarking Unity and Unreal Engine. To capture a reliable result, the benchmarking was done for 30 seconds to capture an average FPS. The results were surprisingly similar considering the differences in the game engines. The FPS difference in the high-quality profiles is only 8,5 while the low-quality difference is 18,3 and without post-processing the difference is 21,6. What is the most noticeable difference is that the high-quality and without post-processing comparisons, Unity has the higher FPS, while the low-quality profile in Unreal Engine has more FPS. This means that Unreal Engine's most taxing post-processing effects are heavier for the performance than Unity's and when they are turned off the FPS gain is more significant. Because the FPS is still very high in all of the cases, choosing the game engine based on post-processing performance is not that relevant.

Table 1. Performance comparison results of comparing post-processing in Unity and Unreal Engine 4.

| Engine | PP Quality   | Avg (FPS) | Min (FPS) | Max (FPS) |
|--------|--------------|-----------|-----------|-----------|
| Unity  | High-quality | 184.9     | 181       | 194       |
|        | Low-quality  | 287.3     | 285       | 292       |
|        | Off          | 535.8     | 525       | 541       |
| Unreal | High-quality | 176.4     | 173       | 180       |
|        | Low-quality  | 305.6     | 293       | 309       |
|        | Off          | 514.2     | 497       | 520       |

So after benchmarking the performance, the conclusion is that the game engine does not matter that much, but how about the differences in visual quality and the adjustability of the effects? The quality of the effects in both engines is very good and gives very

polished looking results, but the largest difference in this benchmarking is that Unreal Engine is lacking when it comes to the adjustability of the effects. Unity gives much more broader options for each effect, because of this the post-processing might look somewhat generic in Unreal Engine while Unity can be customized more to suit the needs of a game.

The Picture 21 demonstrates the benchmarking results visually. Because the lighting works differently in both engines the results are a bit different looking, but the post-processing is trying to be as similar as possible in both engines. The quality in each image refers to the quality of the post-processing profile and on the bottom row, the post-processing is completely disabled. The goal of the post-processing was to create a more toxic looking environment by heavily using the color grading effect to make a greenish tint. The film grain is also increasing the feeling of a radiated wasteland. The purpose of the other post-processing effects is to increase the graphical fidelity more than affecting the theme.



Picture 21. A comparison of post-processing settings between Unity and Unreal Engine.

## 5 FOCUSPLAN CASE STUDY

The practical part of the thesis was to create a reusable post-processing solution for a company called Focusplan. Focusplan is a Finnish engineering office founded in 1988, it specializes in creating solutions and services for many different industrial sectors. These services are mechanical design, plant and process engineering, electrical and automation engineering, surveying services, analytics and simulations, etc. (Focusplan 2018.)

The goal of the thesis was to improve the quality in one of the Focusplan's CAVE showcases. These CAVE showcases are gamelike presentations of built 3D structures. They allow the player to move freely in the game world to inspect every detail. Focusplan uses the Unity game engine to create these showcases and this made it possible to integrate post-processing into these projects. Any exact information about the showcase is under non-disclosure agreement (NDA) and because of this the details about the showcase will not be opened further. Instead the case study will focus on how and why the post-processing was implemented, what were the challenges and the results.

### Creating the post-processing

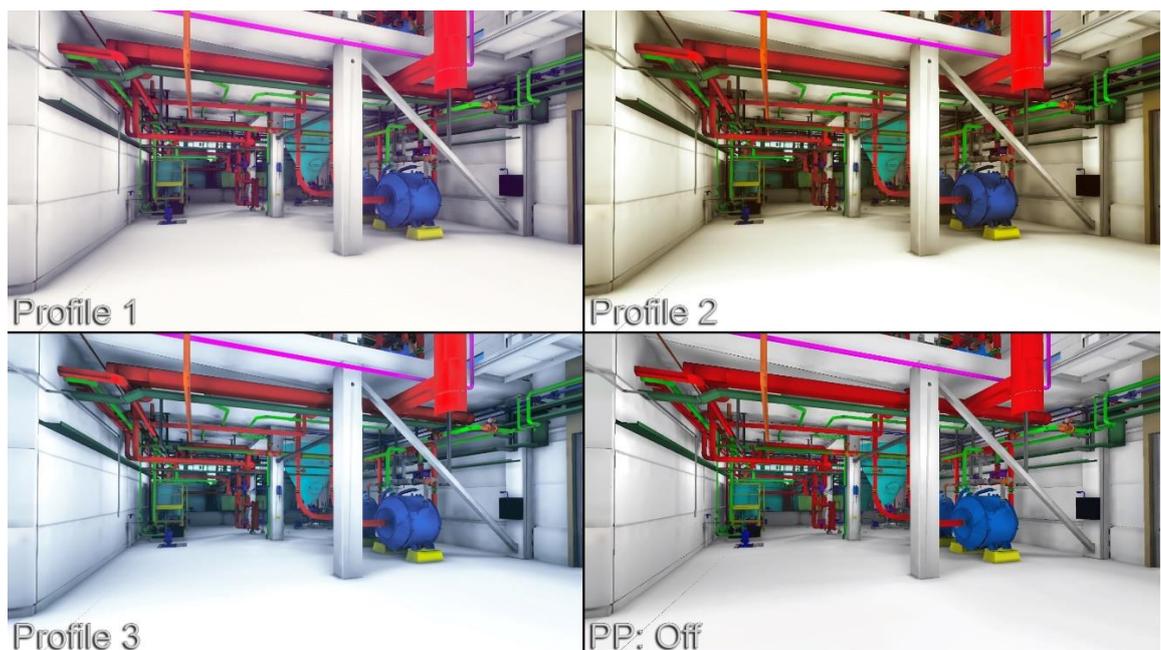
The starting point was that the Focusplan's CAVE showcases didn't have any post-processing applied to them, which made them look really bland visually and the goal of the thesis was to solve this problem. Because of the way CAVE projectors display graphics, there were multiple features that could not be achieved with the post-processing. For starters Unity needs an eight way camera setup, this is because the CAVE system uses four projectors and a stereoscopic 3D effect. This means that every projector needs to have two cameras in Unity to produce the 3D effect. Because of this complex camera setup, Unity limits the possibility of using some of the post-processing effects. For example, using vignette would cause the effect to be rendered individually for each camera instead of rendering it once for the whole CAVE view. This would cause darkened edges for each of the camera views and this would not be acceptable because the final CAVE view should be one seamless widescreen. This same problem applies to effects such as, depth of field and chromatic aberration. Because of this, any of these effects could not be used to create the post-processing.

Another problem was the complexity of the 3D models used in the showcases. Unity is not made to support industrial 3D models with every little detail included. All 3D models that are put into Unity should be heavily optimized for game usage. This means removing all small details from the 3D models. Because the optimization possibility is off the table, the heavy performance cost is very noticeable. For the post-processing this means that effects such as, screen space reflection and TXAA should not be used because of the large performance impact they have. All of these restrictions affect how the final post-processing will look and the artistic freedom is much smaller.

The effects that are used in the Focusplan's showcase post-processing are anti-aliasing, ambient occlusion, bloom, color grading, film grain and motion blur. So half of the effects explained in the Chapter three are not used. These effects were chosen because they are light on the performance side and they are supported by the CAVE system. The other effects would cause problems, either for the performance or the effects would not be compatible with the CAVE system's multiple projectors.

Usually post-processing cannot be used for optimizing purposes, but in this case the FXAA type of post-processing based anti-aliasing replaced the MSAA which would have been heavier for the performance. The ambient occlusion is used improve the soft shadows in the environment. In an industrial environment with lots of small shapes in the geometry, the ambient occlusion helps to ground everything in place. After the ambient occlusion a bloom was used to create a very subtle glow around brighter parts in the game world. This makes the environment seem softer and more natural. Another very subtle effect was the film grain, the main use was to make sure that there are no color banding issues and also to create even better blend for all visuals. Motion blur's purpose was to smooth out the parts when the frame rate goes too low caused by the high detail geometry of the 3D models. Other than that it was not used to achieve anything artistic. Finally the color grading was added and this was the part that had most impact on the visuals. Due to color grading being purely an artistic process, it could split opinions on what is good looking. This is why three post-processing profiles were made, each having different looking color grading. One profile being very neutral and two profiles which affect the colors a little bit more. Still, all of the profiles are quite neutral because in an industrial showcase the color grading cannot be too extreme or otherwise the focus would be in the fancy looking theme, and not in the technical information presented by the showcase.

The Picture 22 compares the three different post-processing profiles and they are explained to give an idea how the post-processing will look in the showcase. Keeping in mind that the environment itself is not finalized and will receive better looking materials later by Focusplan. What can be noticed right away is that each profile has more detailed soft shadows compared to the bottom-right version where the post-processing is disabled. When it comes to the color grading of each profile, the profile one is the most neutral and has only a small magenta tone added to its darker areas. The profile two has a more noticeable difference and goal there was to create a dirtier looking environment than in the other profiles. Because the dirtiness might be something that is not desirable when Focusplan is presenting the showcase to a customer, a third profile was made. The third profile has a small amount of blue tint applied to it and this is usually something that represents new or futuristic look. That is why it could be a better choice when showing the project. All of the color grading profiles are also tone mapped, as this feature is combined in Unity with the color grading.



Picture 22. A comparison of the different post-processing profiles used in the Focusplan case study.

## Conclusion

The end result of the case study is a small pack of optimized post-processing profiles which can be easily drag and dropped into different Unity projects. This means that

Focusplan can use them in all of their showcases and the implementation time is very quick. The different profiles were made so they can be used depending on the type of the showcase or just to use the one which is best in their opinion. The complete post-processing profiles can also be customized later, if Focusplan so decides.

Implementing the post-processing into Focusplan's projects is still an ongoing process. When the post-processing was tested in the Focusplan's own CAVE system, it showed that the effects are working, but the picture shown on the CAVE projectors was overlapping. The problem was outside the topic of the thesis and because of that it is going to be fixed later by Focusplan. The problem also prevented collecting any reliable data from the post-processing performance.

## 6 CONCLUSION

The objective of the thesis was to give a better idea for game developers on what is post-processing and what are the benefits of using it. An additional objective was to identify the problems of bad post-processing and why video gamers do not like it. Also sometimes gamers seem to think that post-processing only includes a singular effect such as bloom or motion blur, but as explained it really is a combination of many effects to create a final polishing layer for a game. By allowing gamers to adjust the individual post-processing effects to their liking, it makes it possible to tone down just some of the effects instead of disabling post-processing completely. This is the better option, because disabling all post-processing effects makes a game look duller.

The thesis also opened most of the post-processing effects on how they exactly affect the visuals and discussed if they have a real world counterpart. The intention was not to go too deep into the theory of every effect but the focus was more on the artistic side. So that the game developers should have a better understanding of how each effect impacts the graphics. After all, post-processing is more about the artistic process and not so much about the technical process.

Testing post-processing in different game engines gave surprising results because the differences in post-processing performance were very small between Unity and Unreal Engine. This means that when choosing a game engine to use, post-processing is almost identically optimized in both engines used in the comparison. Also, the end results looked pretty similar with a small differences in lighting.

The Focusplan case study had unique challenges because of the CAVE system it was built for. To overcome these challenges, the normal post-processing approach had to be changed. It had to be very optimized and also all of the effects could not be used because of the multiple projectors of a CAVE system. The end result of the thesis was three optimized post-processing profiles for Focusplan to choose from.

To improve the thesis a singular Unity scene could have been used, instead of using a unique scene for each different effect in Chapter 3. This could have made it easier for the reader to distinguish each effect better. However, creating a unique scene for each effect made it possible to craft the scene so that each effect would be as visible as possible. The comparison between different game engines could have been improved

by comparing more game engines. While the compared Unity and Unreal Engine are the main choice for independent game developers, comparing more engines could have brought valuable information for game developers. Also, the case study could have been more informative and given more detailed results, as it stands now it only focuses in the making of the post-processing. This was because the post-processing did not work immediately in the Focusplan's CAVE system and that is why benchmarking the performance could not be possible.

## REFERENCES

- Atwood, J. 2011. What is FXAA, And Why Has It Made Anti-Aliasing As We Know It Obsolete? Referenced 5.5.2018 <https://www.kotaku.com.au/2011/12/what-is-fxaa/>
- Computer Hope. 2017. Z-buffering. Referenced 5.5.2018 <https://www.computerhope.com/jargon/z/zbuffering.htm>
- CRYENGINE. 2018. Why choose Cryengine?. Referenced 5.5.2018 <https://www.cryengine.com/>
- Dries, T. 2016. The 7 Crucial Steps to Creating a 3D Game Environment. Referenced 5.5.2018 <http://www.therookies.co/blog/training/7-crucial-steps-creating-3d-game-environment/>
- Focusplan. 2018. Focusplan on pitkän linjan insinööritoimisto Turusta. Referenced 5.5.2018 <http://focusplan.fi/>
- Gallant, M. 2008. Bloom Disasters. Referenced 5.5.2018 <http://gangles.ca/2008/07/18/bloom-disasters/>
- Hable, J. 2010. Filmic Tonemapping Operators. Referenced 5.5.2018 <http://filmicworlds.com/blog/filmic-tonemapping-operators/>
- Hawkins, M. 2018. Depth of field explained. Referenced 5.5.2018 <https://www.techradar.com/how-to/photography-video-capture/cameras/what-is-depth-of-field-how-aperture-focal-length-and-focus-control-sharpness-1320959>
- Jimenez, J. 2011. Practical Morphological Anti-Aliasing. Referenced 5.5.2018 <http://www.iryoku.com/mlaa/>
- Jimenez, J. 2012. SMAA: Enhanced Subpixel Morphological Antialiasing. Referenced 5.5.2018 <http://www.iryoku.com/smaa/>
- Lane, R. 2017. How to: Understand PC game graphics settings. Referenced 5.5.2018 <https://www.pcauthority.com.au/feature/how-to-understand-pc-game-graphics-settings-464265>
- Leadbetter, R. 2017. Digital Foundry: Hands-on with Switch's 'impossible' Doom port. Referenced 5.5.2018 <https://www.eurogamer.net/articles/digitalfoundry-2017-hands-on-with-doom-on-switch>
- Learn OpenGL. 2015. Advanced lighting SSAO. Referenced 5.5.2018 <https://learnopengl.com/Advanced-Lighting/SSAO>
- Mansurov, N. 2018. What is Vignetting?. Referenced 5.5.2018 <https://photographylife.com/what-is-vignetting>
- Maslanka, M. 2017. Explanation of The Color Grading Process. Referenced 5.5.2018 <https://www.motionsource.com/blog/explanation-of-the-color-grading-process>
- Math Open Reference. 2011. Vertex. Referenced 23.5.2018 <https://www.mathopenref.com/vertex.html>
- Morgan, T. 2016. The Division has graphics settings we're not used to seeing on console. Referenced 5.5.2018 <https://www.eurogamer.net/articles/2016-01-28-improve-xbox-one-image-quality-in-the-division-beta>
- Nvidia. HBAO+ Technology. Referenced 5.5.2018 <https://www.geforce.com/hardware/technology/hbao-plus/technology>

- Nvidia. TXAA Technology. Referenced 5.5.2018  
<https://www.geforce.com/hardware/technology/txaa/technology>
- Photographylife. 2018. What is Color Banding and How to Fix It. Referenced 5.5.2018  
<https://photographylife.com/what-is-color-banding-and-how-to-fix-it>
- Pluralsight. 2014. Understanding Ambient Occlusion. Referenced 5.5.2018  
<https://www.pluralsight.com/blog/film-games/understanding-ambient-occlusion>
- Pluralsight. 2015. Understanding chromatic aberration and why lens effects are important. Referenced 5.5.2018  
<https://www.pluralsight.com/blog/creative-professional/understanding-chromatic-aberration-lens-effects-important>
- RED Cameras. Understanding Lens Vignetting. Referenced 5.5.2018  
<http://www.red.com/learn/red-101/lens-vignetting>
- Roberts, P. 2017. What is the purpose of an "Image Buffer" in computer programming? Referenced 23.5.2018  
<https://www.quora.com/What-is-the-purpose-of-an-%E2%80%99Image-Buffer%E2%80%99-in-computer-programming>
- Rouse, M. 2005. RGB (red, green, and blue). Referenced 23.5.2018  
<https://whatis.techtarget.com/definition/RGB-red-green-and-blue>
- Rouse, M. 2010. fps (frames per second). Referenced 23.5.2018  
<https://whatis.techtarget.com/definition/fps-frames-per-second>
- Rouse, M. 2010. Texture mapping. Referenced 23.5.2018  
<https://whatis.techtarget.com/definition/texture-mapping>
- Rouse, M. 2015. Pixel. Referenced 23.5.2018  
<https://whatis.techtarget.com/definition/pixel>
- Senior, T. 2015. Six terrible graphical effects that need to stop. Referenced 5.5.2018  
<https://www.pcgamer.com/six-terrible-graphical-effects-that-need-to-stop/>
- ShaderCat. 2016. What is a Shader?. Referenced 5.5.2018  
<http://www.shadercat.com/what-is-a-shader/>
- Shankar, P. 2018. Graphics Matter. Referenced 5.5.2018  
<https://www.gamespace.com/all-articles/opinions/graphics-matter/>
- Shaw, M. 2017. Screen Space Reflections. Referenced 5.5.2018  
<https://gizmosandgames.com/2017/01/17/screen-space-reflections/>
- Slick, J. 2018. What is 3D Modeling?. Referenced 23.5.2018  
<https://www.lifewire.com/what-is-3d-modeling-2164>
- Technopedia 2018. Gameplay. Referenced 23.5.2018  
<https://www.techopedia.com/definition/1911/gameplay>
- Technopedia 2018. Graphics Processing Unity (GPU). Referenced 23.5.2018  
<https://www.techopedia.com/definition/24862/graphics-processing-unit-gpu>
- TechTerms. 2017. HDR. Referenced 23.5.2018  
<https://techterms.com/definition/hdr>
- Unity documentation. 2017. Manual: Post-processing overview. Referenced 5.5.2018  
<https://docs.unity3d.com/560/Documentation/Manual/PostProcessingOverview.html>
- Unity documentation. 2018. Manual: Ambient Occlusion. Referenced 5.5.2018  
<https://docs.unity3d.com/Manual/PostProcessing-AmbientOcclusion.html>

- Unity documentation. 2018. Manual: Bloom. Referenced 5.5.2018  
<https://docs.unity3d.com/Manual/PostProcessing-Bloom.html>
- Unity documentation. 2018. Manual: Eye Adaptation. Referenced 5.5.2018  
<https://docs.unity3d.com/Manual/PostProcessing-EyeAdaptation.html>
- Unity documentation. 2018. Manual: High Dynamic Range Rendering. Referenced 5.5.2018  
<https://docs.unity3d.com/Manual/HDR.html>
- Unity documentation. 2018. Manual: Reflection Probe. Referenced 5.5.2018  
<https://docs.unity3d.com/Manual/class-ReflectionProbe.html>
- Unity documentation. 2018. Manual: Motion Blur. Referenced 5.5.2018  
<https://docs.unity3d.com/Manual/PostProcessing-MotionBlur.html>
- Unity. 2018. The world's leading content-creation engine. Referenced 5.5.2018  
<https://unity3d.com/unity>
- Unreal Engine. 2018. Unreal Engine Feature. Referenced 5.5.2018  
<https://www.unrealengine.com/en-US/features>
- Usher, W. 2015. 75% Of Gamers Say That Graphics Do Matter When Purchasing A Game. Referenced 5.5.2018  
<https://www.cinemablend.com/games/75-Gamers-Say-Graphics-Do-Matter-Purchasing-Game-64659.html>
- Wahab, F. 2017. How To Manage Motion Sickness Caused By Games. Referenced 5.5.2018  
<https://www.addictivetips.com/windows-tips/manage-motion-sickness-caused-by-games/>
- Wikibooks OpenGL. 2018. OpenGL Programming/Post-Processing. Referenced 5.5.2018  
[https://en.wikibooks.org/wiki/OpenGL\\_Programming/Post-Processing](https://en.wikibooks.org/wiki/OpenGL_Programming/Post-Processing)
- Wikipedia. 2016. Film grain. Referenced 5.5.2018  
[https://en.wikipedia.org/wiki/Film\\_grain](https://en.wikipedia.org/wiki/Film_grain)
- Wikipedia. 2017. Pre-rendering. Referenced 23.5.2018  
[https://en.wikipedia.org/wiki/Pre-rendering#cite\\_ref-1](https://en.wikipedia.org/wiki/Pre-rendering#cite_ref-1)
- Wilde, T. 2014. PC graphics options explained. Referenced 5.5.2018  
<https://www.pcgamer.com/pc-graphics-options-explained/2/>
- Zucconi, A. 2015. A gentle introduction to shaders in Unity3D. Referenced 5.5.2018  
<http://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/>