Thesis no: MECS-2015-05

A TEMPORAL STABLE DISTANCE TO EDGE ANTI-ALIASING TECHNIQUE FOR GCN ARCHITECTURE

Jonas Alexander Göransson

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Engineering: Game and Software Engineering. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author: Jonas Alexander Göransson
E-mail: jonas.goransson@dice.se

University advisors:

Dr. Veronica Sundstedt
Dept. Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Aliasing artifacts are a present problem in both the game industry and the movie industry. With the GCN (Graphics Core Next) architecture used on both new generation of consoles; Xbox One and Playstation 4, a unified Anti-Aliasing solution can be constructed with high performance, temporal stable edges and satisfying visual fidelity.
Objective. This thesis aims to implement several prototypes which will be utilizing GCN architecture to solve aliasing artifacts such as temporal stability.
Method. By doing performance measurements, a survey and an experiment on the constructed prototypes and current state of the art solutions this thesis will create both a benchmark between given state of the art solutions for the industry and at the same time evaluate the new solutions given in this thesis.
Result. With having potential of being the fastest Anti-Aliasing solution in the field it does not only bring high performance, but also very temporal stable edges and satisfying visual quality.
Conclusion. If not used as a standalone solution, the prototype can be decoupled from GCN specific features and be a very suitable complement for Multi Sample Anti-Aliasing which can not handle alpha clipped edges.

Keywords: Anti-Aliasing; Temporal Stability; GCN; Xbox One; Playstation 4; High Performance; Frostbite; Distance to Edge; Deferred Rendering.

Preface

This thesis was written by Jonas Alexander Göransson, a master student at Blekinge Institute of Technology in collaboration with EA DICE. The majority of the thesis was produced at EA DICE head office in Stockholm, Sweden, during a period of 20 weeks of full-time work that started in October 2014 and ended in February 2015.

The assignment was suggested by Frostbite Research and Development and involved finding a fast and high quality Edge Anti-Aliasing solution for the next generation of games, focusing experiments and measurements on EA DICE next coming title STAR WARS: Battlefront.

List of Figures

1.1 Rasterized triangle representation 2
1.2 Sub-dividing the problem 2
1.3 An aliased line followed by an Anti-Aliased representation. 3

2.1 Distance to edge blending directions 6
2.2 The rendering pipeline stages 7

3.1 Perpendicular distance 17
3.2 Signed distance field 18
3.3 Micro triangles and under-sampling 20
3.4 Laplacian filter showing the rapid change in the depth of the scene . . 21

5.1 A preview of the animateable result of temporal stablity. 30
5.2 Prototype 1 versus Prototype 2 31
5.3 Gradients and Laplacian 31
5.4 Figure showing the difference between derivative implementations. . . 32

A.1 No AA 42
A.2 FXAA 43
A.3 SMAA2tx 43
A.4 Prototype 1 44
A.5 Laplacian filter for edge evaluation 44

List of Tables

5.1 SCENE A 720p Xbox One 26
5.2 SCENE A 720p Playstation 4 26
5.3 SCENE B 720p Xbox One 27
5.4 SCENE B 720p Playstation 4 27
5.5 SCENE A 1080p Xbox One 27
5.6 SCENE A 1080p Playstation 4 28
5.7 SCENE B 1080p Xbox One 28
5.8 SCENE B 1080p Playstation 4 28
5.9 Survey on static image, 24 participants. 29
5.10 Survey on temporal stability image, 24 participants. 30

Contents

Abstract i

Preface ii

1 Introduction 1
- 1.1 Context and problem description . . . 1
- 1.2 Aim, objectives and research question . . . 3
- 1.3 Key contributions . . . 4
- 1.4 Research delimitation . . . 4
- 1.5 Structure . . . 5

2 Background 6
- 2.1 The concept of Distance to Edge . . . 6
- 2.2 The rendering pipeline . . . 7
- 2.3 FROSTBITE and STAR WARS: Battlefront . . . 9
- 2.4 GCN . . . 9
- 2.5 Related work . . . 10

3 Implementation 13
- 3.1 Motivation . . . 13
- 3.2 Previous implementations . . . 14
- 3.3 Prototype 1 . . . 15
- 3.4 Prototype 2 . . . 20
- 3.5 Custom partial derivatives . . . 22

4 Evaluation Method 23
- 4.1 Selection of method . . . 23
- 4.2 Setup and expected outcome . . . 23
- 4.3 Experiment . . . 24

5 Results 25
- 5.1 Performance measurements . . . 25
- 5.2 Visual result . . . 29
- 5.3 Temporal stability . . . 29
- 5.4 Prototype 1 versus Prototype 2 . . . 30
- 5.5 Gradients and Laplacian filter . . . 31
- 5.6 Derivative variations . . . 31

6 Discussion **33**
6.1 Temporal stability 33
6.2 Prototype 1 34
6.3 Prototype 2 34
6.4 Measurements 35
6.5 Visual result 36
6.6 Threats to validity 37

7 Conclusion and future work **38**
7.1 Conclusion 38
7.2 Future work 38

Acknowledgments **40**

Appendices **41**

A Bonus content **42**

References **45**

Chapter 1

Introduction

The following chapter will introduce the reader to the subject, however the reader should have in mind the complex background of the problem will need some clarifications from different fields of input, such as background of previous work and how hardware solves some sort of tasks etc. The content of this part of the thesis will start with a problem description, aim and goal of the research, contributions and delimitations.

1.1 Context and problem description

Quality in rendering is one of the core elements when it comes to the game industry and also the movie industry, however when it comes to real-time systems the goal can be more complex to reach. Going from a high frequency to a lower frequency mathematical representation can cause artifacts, and this phenomenon is usually called Aliasing. One everyday example of this is when one can see the wheels on a vehicle, e.g a car-wheel rotate backwards when indeed the vehicle travels in the opposite direction. This is because the motion perceived is reconstructed and interpreted in a lower frequency than the real motion, causing aliasing. Some aliasing effects are natural how we perceive them and may even be what we want to end up with in our digital reconstruction, e.g. the wheels rotating backwards in a racing game, or what ever direction the Nyquist threshold will make us interpret it [10].

However there are other aliasing artifacts that occur due to that the digital reconstruction is in a lower frequency than what we can perceive. For example the amount of pixels on a screen versus the physical bounds of what a human eye can see, as shown in Figure 1.1. There are none in the authors knowledge scientifically proven fixed value on the boundary what numbers of pixels that can be perceived by the viewer. However the strong connection between the receptors in the human eye and the resolution we see the world in is a challenging subject. One must remember the brain does a lot of work when reconstructing the signal, and the density of the receptors in the eye do vary in the field of view [17]. It is indeed a complex problem which this thesis will not focus on. However the number of pixels are not the relevant variable, but instead the

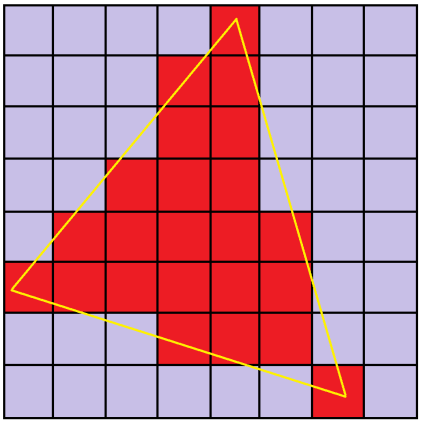


Figure 1.1: The yellow outline represents the mathematical triangle however after being rasterized the red pixels represent an aliased version of the triangle on the screen.

density of the pixels and the distance from them are.

One can make the assumption that this problem will be solved by screen technology in the next couple of years, by just dividing the problem until the points are smaller than what we can perceive, see Figure 1.2. Even if the trend of increasing the resolution tends to sustain, this will not necessary mean that televisions will become smaller in size. Even without this assumption, with the new 4k screens with almost four times the resolution of regular 1080 screens the aliasing problem is still present and the fact that new generation of consoles will be locked in hardware until the next generation arrives will not be able to solve this problem by rendering in higher resolutions with their hardware limitations.

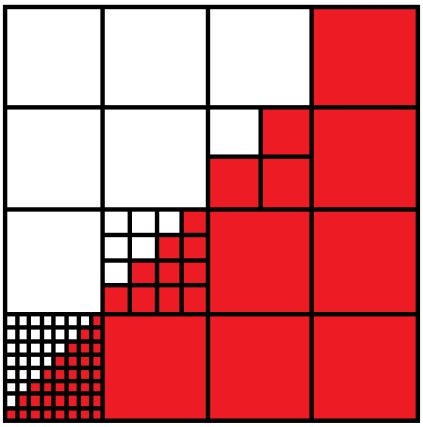


Figure 1.2: By sub-dividing the problem and increasing the density of the pixels will eventually remove the aliasing problem.

Aliasing can take many forms and be the cause of many different problems in computer science, this thesis however will focus on the rendering artifacts as pixel-space edges i.e jaggies, and how to solve those artifacts i.e Anti-Aliasing as seen in Figure 1.3. Those will be the artifacts refereed to when the terms Aliasing and Anti-Aliasing are used through the thesis from here and on.

Figure 1.3: An aliased line followed by an Anti-Aliased representation.

However the focus of the thesis is more precise than that and more explanation of the problem is required before understanding the real scope. Aliasing occurs as mentioned earlier when rendering at a lower resolution than the eye can perceive. This will be explained in section 2.2 about rasterization. Temporal stability will also be covered in the same section.

1.2 Aim, objectives and research question

The aim of this study is to evaluate if one can implement and integrate an Anti-Aliasing solution which does not depend on sub pixel data like common super sampling techniques and can handle temporal instability which most post-processing solutions can not. All these in high performance in the Frostbite engine and evaluated on the static hardware environments; Microsoft's Xbox One and Sony's PlayStation 4. The aim of the thesis is to find a new state of the art solution for solving Anti-Aliasing on the new generation of consoles powered by the GCN architecture and a worthy contender to Nvidia's Fast Approximation Anti-Aliasing (FXAA) solution [7], in both performance and quality.

Objectives

1. Implement a prototype of a Distance to Edge concept, highly adapted to GCN architecture and resolve artifacts as post-processing pass.

2. Evaluate quality of the prototype versus other state of the art Anti-Aliasing solutions, in both Anti-Aliased edges and temporal stable edges.

3. Evaluate the performance impact on the systems and pinpoint the bottlenecks, if any.

Research question
How large are the differences between the given technique and proven state of the art Anti-Aliasing solutions when it comes to performance, visual quality and temporal stability?

1.3 Key contributions

The implementation can, without any sub-pixel information and by storing extra geometrical data in a pre-render pass, prevent temporal instability and solve aliasing artifacts during a post-render pass. Thus being a real alternative to the state of the art post-processing Anti-Aliasing solutions, in sense of performance and quality in high end game engines and systems.

1.4 Research delimitation

This thesis will only look at a solution for GCN powered hardware, which is both used in Xbox One and PlayStation 4, however no measurements will be done on other systems utilizing the GCN architecture. The main reason for this is the time scope of the thesis and huge amount of different system setups utilizing GCN. Instead by studying the static setup of the hardware on the new generation of consoles this thesis will deliver results to a broad chunk of hardware users and developers and create results comparable for all other GCN systems. This thesis is written under a non-disclosure agreement (NDA) between the author and EA DICE, which will lead to:

1. Some of the quality evaluations screenshots will not be presented in this thesis.
2. Some of the measurements will not be presented.
3. No source code or content from the game or engine will be presented.

The points stated above will not affect the validity of the thesis but will create restrictions on how results will be presented, for example some data can not be presented in raw form due to its relationship to other systems. This is to prevent data to be reversed computed/engineered which can tell how systems are working/constructed, this information is protected under the NDA and is not allowed for the public, however everything needed for the validity of the thesis will be presented.

1.5 Structure

Chapter 2 explains key components and concepts which are needed to understand the scope of this thesis. The chapter also covers related work and what have previously been done in the field. Chaper 3 is about the implementation and describes all the steps in previous solutions and the new proposed prototypes. Chapter 4 covers the scientific method used in this study, expected outcome and the experiments layout. The result of the implementation and the result of the experiment will be presented in Chapter 5. Discussion regarding the results, the prototype, experiment and potential threats to validity can be found in Chapter 6. Finally Chapter 7 covers the conclusions of the study and future work.

Chapter 2

Background

In this chapter key background research is explained and all information the reader should be aware of to understand the following chapters in this thesis. It starts with the concept of the idea, continued with the core building blocks in the rendering pipeline. Some information will be clarified about the company, the engine and the game, followed by a brief description on GCN and what it is. It is finally concluded with related work in the field.

2.1 The concept of Distance to Edge

In this section the concept and idea behind the prototype will be presented just so that the reader understands the core concept of this Anti-Aliasing technique. This is because the reader will be presented a long list of background items which the author have utilized the features and knowledge about. The idea is to store the distance from the nearest edge in all pixels, then when the scene is rendered an extra post-pass blurs neighbors with each other if an edge is shared by the pixel pair, which can be seen in Figure 2.1. The idea is pretty simple which means it can take, and have taken, many forms which can be read about in the related work 2.5. The key thing to note is that the method will not only be a post-pass but also a pre-pass storing geometric data, however no sub-pixel information is used, i.e the rendered resolution will be the at least the same or smaller if up scaling to the screen resolution is required.

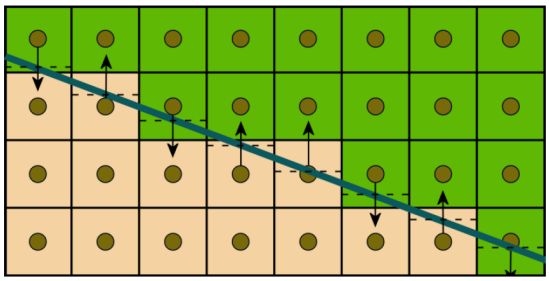


Figure 2.1: Showing the blend directions and distance to edge for each pixel. Image taken from Emil Perssons presentation [11].

2.2 The rendering pipeline

This section explains briefly the rendering pipeline and its stages. This is relevant for several reasons. First of all how the traditionally rendering pipeline looks and behaves, so one can understand where and why implementations have been done as they have. Second when discussing the GCN architecture, there are some new features presented that users of the traditional pipeline might not know and this thesis will take advantage of. Now the rendering pipeline will be briefly explained, with Figure 2.2 as an reference. However the core thing that differs in different API's e.g. Direct3D versus OpenGL is the naming convention. This thesis will just cover the relevant stages for the proposed prototypes and the previous implementations.

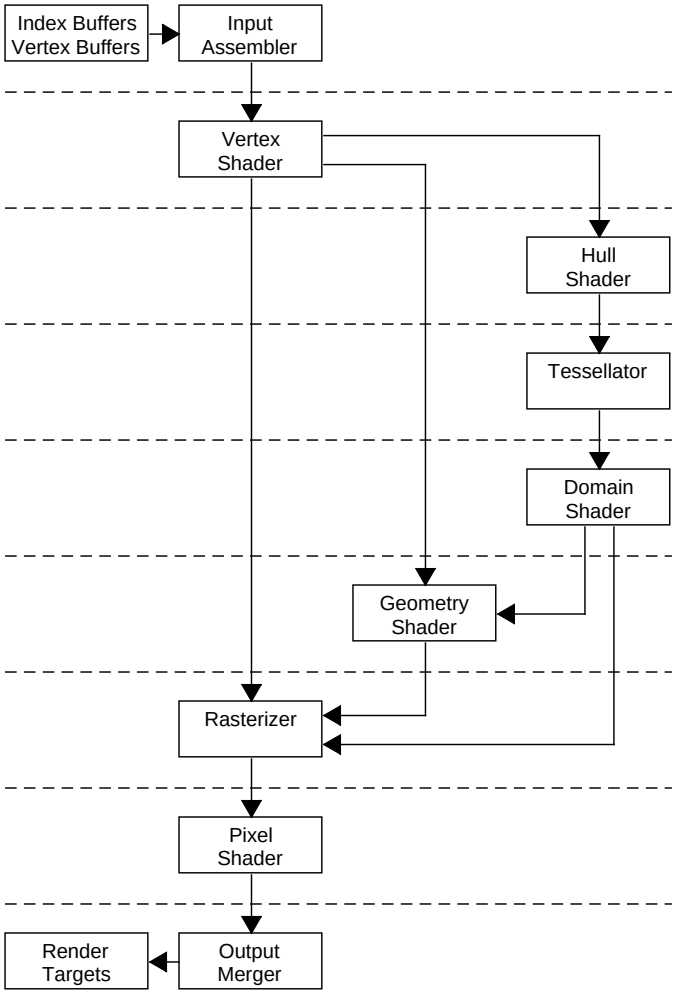


Figure 2.2: The rendering pipeline stages. Adapted from: https://msdn.microsoft.com/en-us/library/windows/desktop/ff476882%28v=vs.85%29.aspx Available 2015-04-19

Vertex Shader Stage
Here are all the vertices handled, the typical use for this shader stage is transformation, using world- view- and projection-matrices to transform a vertex position from its local defined space to world space to view space and finally to clip space.

Geometry Shader Stage
This stage is optional in the rendering pipeline, however this stage is executed after the vertex shader stage. Here the user has acess to the whole primitive, in this case the triangle with its three vertices. This stage is used by Emil Persson [12], where he calculates the distance to edge for every vertex in the triangle. More information can be found in the related work 2.5. However the key thing to note here about the geometry shader is its purpose to clip or add additional geometry or change the even structure of the geometry, this uses extra bandwidth, evaluation and computation from the hardware.

Rasterizer Stage
This stage is a non-user stage which means the developer can not program what happens here. However this stage is very important to understand what exactly happens in the hardware. The description from MSDN summerises it like this:

"The rasterization stage converts vector information (composed of shapes or primitives) into a raster image (composed of pixels) for the purpose of displaying real-time 3D graphics. During rasterization, each primitive is converted into pixels, while interpolating per-vertex values across each primitive. Rasterization includes clipping vertices to the view frustum, performing a divide by z to provide perspective, mapping primitives to a 2D viewport, and determining how to invoke the pixel shader. While using a pixel shader is optional, the rasterizer stage always performs clipping, a perspective divide to transform the points into homogeneous space, and maps the vertices to the viewport. Vertices (x,y,z,w), coming into the rasterizer stage are assumed to be in homogeneous clip-space. In this coordinate space the X axis points right, Y points up and Z points away from camera."[1]

The representation of the cartesian coordinate system may differ from different API's. However what happens during rasterization usually does not, this will be explained in the GCN section 2.4. The problem when it comes to aliasing is that a pixel rasterized is considered as an binary operation, either it is inside the triangle or it is outside. This is what is refered to as temporal unstability during movement. The observer will see this binary behavior as waves traveling on the edges. Another important thing to note during rasterization is that even if the programmer can do very little to interact with this stage, the native implementation of rasterization will also utilize the massive par-

[1] `https://msdn.microsoft.com/en-us/library/windows/desktop/bb205125%28v=vs.85%29.aspx` Available 2015-04-06.

allel structure of the GPU. This means that the same geometry rasterized will be in a fixed region, a minimun of 2x2 pixels are guaranteed to get rasterized at the same time and invoke pixel evaluations. Obviously the GPU is capable of executing more pixels in parallel which it also does, however the knowledge of 2x2 execution is the thing to note at this point.

Pixel Shader Stage
In the pixel shader stage; or a more suitable name fragment shader, we have either fragments of pixels or whole pixels, but for the simplicity we refer to everything as pixels in this description. Here we only get a single pixel which does not know much about its origin or how many pixels have been evaluated before it, but the following properties can describe a pixel:

1. It has not been rejected based on its depth value, so it might end up in the result to present to the screen.
2. It has all its values interpolated between the parent vertices by the rasterizer, and therefore has lost the orientation of the geometry it belongs to.
3. The rasterizer has transformed the position from clip space to screen space.

Now with the basics of the rendering pipeline the following steps in the thesis will be easier to understand when we talk about the implementation and GCN specific operations and structure.

2.3 FROSTBITE and STAR WARS: Battlefront

This section is about what is relevant to know about the Frostbite engine and what is relevant about the game itself. Some details can not be exposed in this thesis about several things protected by the NDA, however none of those details will affect the validity of the results. The implementation is for deferred rendering where one store all pixel data in a pre-render pass and postpone the shading to a post pass, this is what refers to when storing values in the gbuffer. This technique saves a lot of computations on depth rejected/occluded pixels and opens up to a easy way to shade pixels with huge amount of illumination. When complex integration is refered to in this thesis the author refers to the big shader graph system and how shaders are built in Frostbite, a more in depth presentation on the subject can be read by Johan Anderssons presentation on the shader graph implementation [2].

2.4 GCN

Graphics Core Next (GCN) by AMD, is the GPU architecture used and studied in this thesis. As mentioned earlier the same architecture is used in both Xbox One and Play

Station 4 which is the new generation of gaming consoles that will stay with us for the coming years. They are relevant for studies just because of the architectural difference, the performance and what new concepts they opens up for. The reader is strongly advised to read the crash course of GCN architecture given by Layla Mah [8], however a brief description of what is relevant to know about GCN follows. These will be enough for the reader to understand this thesis. Wavefront is the term for a group of 64 threads on GCN. They are strongly connected to the use of vGPR (vector general purpose registers), briefly explained as the number of registers used will impact the numbers of wavefronts that can be executed in parallel. However one thing to note here is the fundamental change of the hardware, one can not use the hardware to its full potential just utilizing the standard rendering pipeline. There are parts in the hardware only used by compute shaders and general purpose execution, and the same for the reverse case. This means to utilize the GCN at its fullest is a combination of compute shaders and the standard pipeline, however sequential execution of job dispatches are not enough. Asynchronous calls open up for new algorithms to be studied and invented. There are several things to know about the different units on the GCN such as standard ALU, scalar calculations and branching, however this thesis will not go into detail on those features but they will affect how code should be written. The, by far, most important thing about the GCN architecture for this thesis is how it handles interpolation. All interpolation is done by the pixel shader itself instead of the rasterizer doing it for every invocation. All this will be explained when the implementation is presented in Chapter 3, with code example and a more in depth description.

2.5 Related work

Anti-Aliasing has been a topic for many years, and by now the reader should be aware of what causes aliasing. The following section will now explain some other state of the art solutions followed by what has already been done and the background of the concept for this thesis solution.

A common technique mostly used in the movie industry for all visual effects is Super Sampling (SSAA) which solves both aliasing artifacts and temporal unstable edges [3]. This technique is also used in the game industry to some extent, however it is very computationally heavy so it is commonly used for high end setups. How SSAA works is straight forward. It renders to a larger target than the intended resolution. By pretending that the target is some defined power of two larger than the end target, one can take an average on those fragments which are inside the main render target pixel and end up with a very high quality representation. With the definition on how rasterized rendering works this is the closest to the visual truth as possible based on the target resolution.

Instead of Super Sampling a very common technique in the gaming industry is Multi Sampling (MSAA) which uses as much memory footprint as SSAA but not as much

computational footprint [15]. It works almost the same way as Super Sampling however it only does the super sampling part on the actual edges. On most GPUs there are native support for this and the rasterizer figures out the coverage for each pixel deciding what to do. This saves a lot of computations but the implementation can be complex to integrate with a deferred renderer when the native support is for forward rendering and the deferred will loose its sense of geometry when it writes all data to the different gbuffers. Though the solution is as temporally stable as SSAA the solution is not perfect. Edges which are not created by geometry such as alpha clipped edges will no longer get Anti-Aliased. It is impossible for the rasterizer to know if a pixel will be clipped and therefore create an edge.

Both SSAA and MSAA are very common and well know techniques however the industry tends towards using post-processing techniques for their minimal memory footprint and their relative low performance impact. There is a huge amount of different techniques however this thesis will briefly describe the most common used post-processing in games which are also the ones measured against during the evaluation of the new technique.

Fast Approximation Anti-Aliasing (FXAA) was developed by Nvidia [7] to be exactly as the name suggests. It was inspired by several other techniques and simplified to a very fast solution. In essence it just takes a color texture as input, i.e the rendered frame in which it estimate the difference in luminance. In that way it finds out if the edge is horizontal or vertical, after several other small estimations it re-samples the input with a offset it has calculated and applies a lowpass filter for blurring the edges. This technique is very simple and has high performance. It also solves alpha edges which MSAA does not, however it is not temporally stable. This is because it works on a post-pass image and as mentioned earlier the rasterization is a binary operation on the pixel if its on or off, so one can still see the pixels turn on and off in motion i.e still temporal unstable. One other significant drawback of the technique is that it can find edges were there are none, resulting in a blurred image, this is because of the fundamentals on how the technique works with color difference and luminance.

Another popular technique which yields very high quality is Enhanced Subpixel Morphological Anti-Aliasing (SMAA) [16] which is new enhanced version of Morphological Anti-Aliasing(MLAA) [14]. It can easily be combined with MSAA or temporal reprojection. The technique is similar with FXAA however it can use luminance, normals, depth and even geometry id to determine edges. It also has pre-stored different blending patterns for different characteristics of edges. In its simplest form it does not handle temporal stability well, it has the same disadvantage as FXAA except it is better to find true edges. As mentioned earlier it also works with MSAA or temporal reprojection in which one add a simple jitter to the camera to get different sample points between frames. While later add and weight the frames together will handle subpixel artifacts and make it more temporal stable. The versions used in the experiment were

the standard SMAA and SMAA T2x with two times reprojection.

The concept of distance to edge will now be discussed and the background on what has been done in this particular area. When it comes to storing distance to edge on a per pixel basis in a gbuffer, very little is documented and almost nothing academically published nor evaluated. It is mentioned in the conclusion of several courses held at Siggraph 2011 [6]. Both Distance to Edge Anti-Aliasing (DEAA) [9] and Geometry Buffer Anti-Aliasing (GBAA) [11] are presented in the courses. However it is only simple presentations of the techniques and are more ideas and concepts on how such a technique could work. As mentioned above the courses were more of a presentation of concept and a forum for discussion instead of an evaluation. Which is one motivation for this thesis to evaluate the concept and see if it is a plausible technique with the new technology and hardware such as GCN. There are several differences on how they approach the problem, Emil Persson uses a geometry shader to calculate the distance to an edge which he describes on his blog [12]. DEAA sends barycentric coordinate from the vertex shader down the pipeline. How their implementations work in more detail and what problems their techniques have will be explained later in this thesis. Michal Drobot talks about the GCN architecture during Digital Dragon 2014 [4], and mentions a new approach which takes advantage of the interpolation on GCN to approach DEAA and GBAA which this thesis utilizes. He also mentions that he has implemented the technique in two projects, names which are not announced and therefore the technique not confirmed if it exists in a product out on the market or not.

Chapter 3

Implementation

In this chapter the two prototypes will be described, however the chapter will start with explaining the motivations and requirements for the prototypes 3.1, followed by a description of the old solutions to the concept 3.2. After that the prototypes will be presented in chronological order.

3.1 Motivation

In this section we will cover the background on how the new solutions arose based on the old implementations and interpretations of distance to edge, such as GBAA and DEAA. When it comes to finding a solution one must understand the context in which the implementation actually is a satisfying solution and the scope on what criteria must be fulfilled to even be a plausible alternative to the state of the art solutions. To meet those criteria we constuct a list of information about the problem we know.

1. The domain we are looking for a solution in is a real time application e.g Star Wars Battlefront, which means it will need a high performance solution to still be considered a high frames per second realtime solution.

2. We are looking for a temporally stable solution, which means that in as many cases and situations as possible the solution needs to be temporally stable.

3. Because it is a part of a large system, i.e a game or an engine the implementation and integration needs to be as simple as possible, this problem arises from that the solution is no longer a post-processing pass with very little relationship to the other system. In large systems such as Frostbite small changes may affect different components, and because of the outcome can be unknown the integration should affect as few sub-systems as possible. As for here when it is bound to the gbuffer pass it can, and will, have relationships with the content and the data bandwidth.

These items will be the base of what and why things needed to be done differently from the old implementations. Thus be a plausible solution on a real time application e.g. Star Wars Battlefront.

3.2 Previous implementations

First DEAA will be covered on how it works followed by how GBAA works. The thesis will pinpoint out how and why the implementations are not plausible for a AAA title and the drawbacks on those solutions.

DEAA takes a content driven approach where it puts a lot of responsibility on how the triangle mesh is constructed. The technique is to let the rasterizer interpolate and create barycentric coordinates which it can do if each of the vertices in a given triangle has a unique component in a three dimensional vector set to one and the other two to zero. Then the result of a given pixel inside the triangle will produce a barycentric coordinate. What needed to be done with the barycentric coordinates will be covered later when discussing prototype 1 and partial derivatives in section 3.3, however we must discuss how one can make this approach work with the content. There are indeed restrictions and rules a graphical artist needs to follow when building a triangle mesh, such as vertices shared between triangles. However when it comes to high performance, adding data to process will not save performance when bandwidth is a large restiction in defeered rendering systems. So it is not common to have a mesh that has several vertices representing the same point in space, usually the case is only one vertex holding all information, unless one wants sharp edges on the mesh with normals pointing in different direction e.g a box. As mentioned the common case is to share vertices and use indexation or make the mesh a triangle strip. With that explained the solution becomes not so trivial, because now we have relationships between how the vertices are ordered and how they must be connected to neighbouring triangles. With some triangulations, it is impossible to create this pattern consistently without subdividing and adding extra vertices to reach the condition.

So even if it is baked in the mesh or one dedicates the values in a vertex shader based on index and order as done in DEAA one must put responsibility to the artists for an extra step in their pipeline of creating assets. Adding data may or may not be the largest problem. A hard problem arises when a Xbox uses e.g DEAA and a highend PC uses e.g SSAA as a different setting. Extra setups of data may not be preferable, this is probably a trivial task to solve. However one must remember AAA games are big systems and include huge amounts of people working in different areas, so if one can decouple some dependencies between the domains it can be a huge win in production. This is one motivation for why a new solution is proposed. This is something that is perferred to be calculated during runtime and not put the responsibility on the content.

Geometry Buffer Anti Alising (GBAA) conducted by Emil Persson is the main source of inspiration for this thesis. His idea was to decouple the dependencies of content by solving the major part of the problem in a geometry shader. By doing it at the shader stage he could use all three vertices which creates a triangle and calculate their position in screenspace. After that he calculated the distance in pixels from each vertex

to the facing edge. Like DEAA he let the rasterizer do the job of interpolating the distances to the pixel shader. In the pixel shader he could just check which one of the three components had the lowest interpolated value and if it were less than one pixel, then he had an edge in that very pixel and also the exact distance from the sample point.

The post pass will be explained in the following section covering the thesis prototypes, this is because of the similarities and the idea of the concept does not change as much between the interpretations. However the major thing about why GBAA is not a preferred solution is because of performance. As mentioned earlier the Geometry Shader has a very powerful field of use as it can add and clip geometry. This comes with a large price on performance depending on how the use case is defined. As the geometry shader can change the topology the GPU must evaluate the topology before and after the geometry dispatch. While the geometry may have changed in the data layout all values must be copied over to a new internal memory so all the data is spacial aligned. By just adding data and changing the structure as done in GBAA the price of the implementation is high on performance and bandwidth.

Now both the prototypes constructed during this thesis will be covered starting with the first one which behaves most like DEAA and GBAA. We will go more indepth on every part and explain some of the parts postponed in the description of DEAA and GBAA such as partial derivatives and alpha edges.

3.3 Prototype 1

Now why GCN is so important for this thesis will be explained. The reader should now be familiar with is the rendering pipeline stages and, most importantly, with the rasterizer and the previous implementation approaches. To understand the importance of GCN we will start with a simple code snippet of a dummy pixel shader which changes how we should look at the traditional pipeline.

Start off with a simple input to the pixel shader stage and the shader program:

```
struct pixelData
{
    float4 position : SV_POSITION;
    float4 color : COLOR0;
}

float4 psMain( pixelData input )
{
    float4 output = (float4)0;
```

```
    output = input.color;
    return output;
}
```

This is a very simple pixel shader program which only returns the color of a pixel given from its vertices. However the interesting part which cannot be seen by just the user's implementation of this program is what the compiler and driver are doing in assembly code. If instead looking at the assembly code from the GCN driver for this specific program some specific instructions appear in the very start of the program, which follows:

```
v_interp_p1_f32 v2, v0, attr0.x
v_interp_p2_f32 v2, v1, attr0.x
v_interp_p1_f32 v3, v0, attr0.y
v_interp_p2_f32 v3, v1, attr0.y
v_interp_p1_f32 v4, v0, attr0.z
v_interp_p2_f32 v4, v1, attr0.z
v_interp_p1_f32 v0, v0, attr0.w
v_interp_p2_f32 v0, v1, attr0.w
```

The assembly code above is actually the interpolation happening in the pixel shader itself. Interpolation on GCN does not happen in the rasterizer anymore. More about this can be read in AMD's Southern Islands Instruction Set Architecture [1]. What this means is that the hardware-generated barycentric coordinates and the actual data from all three vertices are stored in the Vector General Purpose Registers (vGPR) and local data shared (LDS) memory. This behavior is actually intentional. If the nointerpolation prefix is used instead on the interpolants or non-interpolatable data types such as integers, the following assembly will be generated:

```
v_interp_mov_f32 v0, p0, attr0.x
v_interp_mov_f32 v1, p0, attr0.y
v_interp_mov_f32 v2, p0, attr0.z
v_interp_mov_f32 v3, p0, attr0.w
```

This loads the specific data into registers and yield the same data as the last case. This was presented by Michel Drobot at Digital Dragon Presentation's late summer 2014 [4] where he discussed various low level GCN optimizations and ideas as well as applications such as GBAA. Emil Persson also discussed this and various optimizations on the Game Developer Conference 2014 [13]. The intention of this behavior is to let the

programmer do custom interpolation on various custom formats. However this thesis will use this information as free computations which in the old implementations were very costly.

There are several ways to calculate the distance to the edge, first we will cover perpendicular distance and then calculate the distance with partial derivatives. Perpendicular distance is a simple one dimensional projection on the normal to an edge, as can be seen in Figure 3.1. The problem is easy to solve because it is in two dimensions, i.e. screen space. One can find the normal to an edge by just swizzling the components and set one of them to zero for a given vector between the corners of the triangle. From there a simple projection on the normalized normal will give the distance in pixels from origo. The major direction on the normal tells which blend direction will be used if that given edge is the closest one.

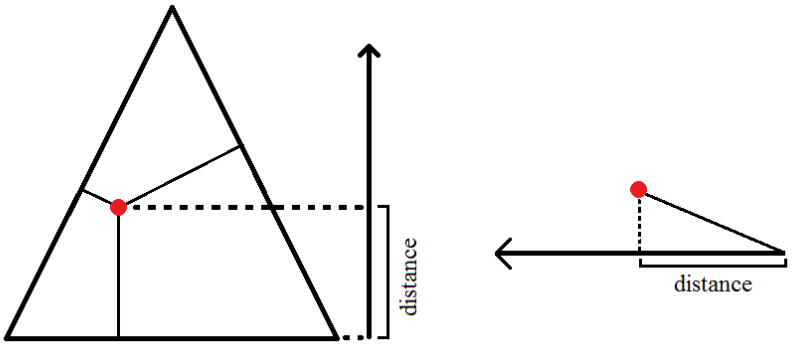


Figure 3.1: Calculating the distance to each edge in a triangle from any point within the triangle, by finding a perpendicular normal and project the point onto it.

When it comes to alpha edges, i.e. clipped pixels due to an alpha channel threshold, such as foliage or decals, to name a few, the information about the edge looks different. No longer can we use the triangle data to calculate the distance because of the fact that an alpha edge can emerge anywhere within the triangle itself. To handle this problem we have to know the following information:

1. The threshold where alpha clipping occurs.
2. Which alpha value any given pixel has.
3. The rate of change in the alpha value between pixels.

All these values are easy to get a hold of on a modern GPU, either they are user defined values or calculated on the GPU. The threshold is just a user setting on the alpha channel on the diffuse texture. This thesis uses the decimal value of 0.5 as the clipping

condition, this is because it will be represented as half of the maximum value in its different binary representations. A problem when it comes to alpha clipping on a diffuse texture is that bilinear interpolation on the texture sampling in fact is not linear, which will produce non-linear edges. To compensate Signed Distance Fields (SDF) are used in the same manner as Chris Green from Valve talks about in his paper on vectorized decal graphics [5]. To explain it briefly we store the distance from each clip threshold in the texture to every other texel, i.e data point, in the texture. In other words the distance from each alpha edge is stored in the texture, with 0.5 as our edge, $0.5 < 1.0$ for texels inside the object and $0.0 < 0.5$ for texels outside the object. With this extra dimension of information a simple circle can look like the red 3-dimensional visualization in Figure 3.2. The purple plane in the figure cuts our circle in the edge value 0.5.

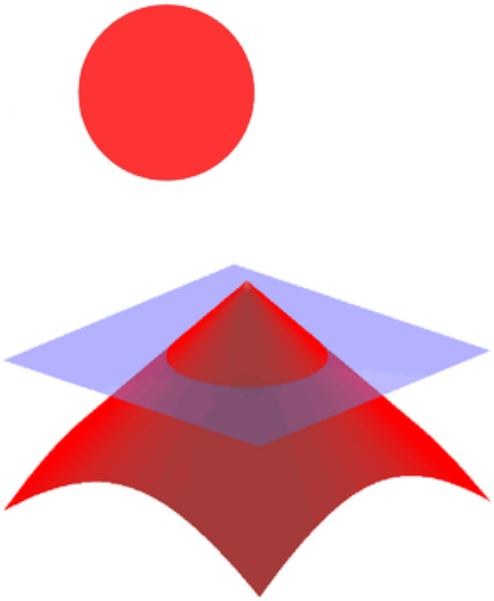


Figure 3.2: The resulting rasterized circle represented as a 3 dimensional signed distance field.

The rate of change per texel is needed to know how far from the edge a given sample point is, however there are indeed an infinite amount of derivatives in a given point on the 3 dimensional graph given in Figure 3.2. In Cartesian coordinate systems using the partial derivatives and locking the axis for a given derivative will give the results needed for this thesis. We are just interested in the change of rate in the two axis in screen space. The following formula will give the distance vector to the edge on the x-axis and the y-axis in screen space.

$$\overline{distance} = (\frac{-\alpha}{(\frac{\partial \alpha}{\partial x})}, \frac{-\alpha}{(\frac{\partial \alpha}{\partial y})})$$

When it comes to calculating the derivatives in a pixel shader there are some native support for this using the same functions the hardware is using while choosing mips in Anisotropic filtering. Usually called something like DDX and DDY for the partial derivatives in each dimension. However it is very important to know how, and why, this functionality works. As mentioned earlier pixel dispatches are done in quads of 2x2 pixels, which means that the hardware during execution has information about a pixel and its neighbor and can easily calculate the rate of change for any given value within the flow control. This has its restrictions the reader should be aware about, this will only give the rate of change in a pixel quad. This means all pixels within that specific quad will have the same rate of change and now any relationship with the neighboring quads. In other words the resolution of the rate of change is approximately the half of the rendering resolution. When and why this is a problem will be discussed later in the thesis in section 3.5.

Now when the distances have been calculated for both the alpha edges and the triangle edges the result has to be stored in any of the gbuffers in the deferred render. It is possible to bind extra rendertargets with different data size as long as the dimensions do not change. In this prototype an extra 8 bit rendertarget was used for the distance to edge post-pass. As the distance is in screen space any distance over 1.0 is irrelevant, one might think any distance over 0.5 is irrelevant because of the fact that the sample point is in the middle of the pixel. However storing the distance to the next neighbors sample point is essential. This is because the neighboring pixel may not have been rasterized by that very geometry creating the edge. Therefore the information must be stored in a neighboring pixel. This will be clear when we discuss the post-pass. 8 bit data is more than enough to store the distance, 2 bits are used for the direction / axis the blend will occur in, the rest of the bits are used for the gradient, also known as the distance. A total of 5bits used yield a very good result but it is not within the scope of this thesis. In this thesis all bits were used because an unused bit costs the same amount of bandwidth as a used one in a byte.

The post pass is very simple, this is because all of the complex work has already been done when rendering the geometry during the pre-pass and storing the distance to edge information in a gbuffer. In this pass we only look at the stored value and apply it to our sampling point and resamples from the result image, i.e the image to end up on the backbuffer. However as mentioned earlier there are pixels which may have an edge inside them from another source of geometry. This is detected by them having the upper half of the distance, i.e 0.5-1.0 in distance. This means they have to check the immediate neighborhood if there is an edge splitting the pixel in question. This branch can easily be flatten out to avoid divergence and extra execution time.

This concludes the first implementation, and any results or discussions about it will

be covered later in this thesis, Chapter 5 and 6. However another implementation was conducted which will be covered now.

3.4 Prototype 2

This implementation evolved from Prototype 1 which addresses a few problems that will be clarified below. The post-pass is the same as in prototype 1, but an extra middle-pass is added. First we will present the changes in the pre-pass where the distance to edge is calculated and stored. In the pre-pass we now store two edges instead of one, the closest one as before and the next closest one. There are cases when two edges are inside a pixel and the closest one might not be the one to use. More about why this is the case will be covered later below. However it should be clear that in this case we might need the other edge, and we need to add an additional middle-pass to evaluate which edge to use.

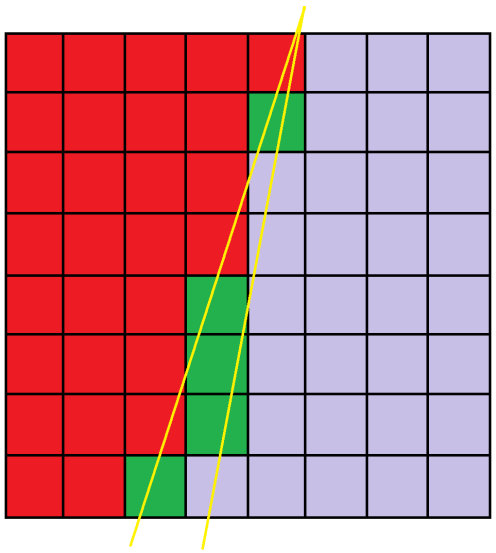


Figure 3.3: The yellow outline represents a green triangle neighbouring the big red one, in this example one can see undersampling and several edges in the same pixel.

The middle-pass is implemented on a compute shader, the motivation for this will be discussed in the discussion part of the thesis, as there is nothing that limits the implementation to a compute shader. It can be done on a pixel shader however as mentioned earlier in this thesis GCN opens up for asynchronous calls. When it comes to compute shaders on GCN they can execute in full parallel utilizing resources not accessible by the normal pipeline, which implies that one cannot utilize the hardware to the fullest without having compute shaders mixed in with the regular pipeline. After

the pre-pass has stored both the distances in an extended 8 bit gbuffer, in the same format as previous prototype, a compute shader executes to decide which edges to use. In cases where two edge are inside the triangle choosing the right one might not be trivial. A common case can be seen in Figure 3.3, where the closest edge is the one facing to another triangle in e.g. the same mesh, when it should have chosen the real silhouette edge instead. To solve this problem we use a similar approach as SMAA, by using the depth data. Using the depth buffer and applying the following simple Laplacian filter on it:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

This will give us the the rapid change in the depth buffer, an example of this can be seen in Figure 3.4. With this extra data it can be calculated in what direction an edge actually is facing. Also with this data and a certain user threshold depending on the nature of the data we can filter away all edges between triangles which are not the edges we are interested in. This conclude the implementations and will be discussed and evaluated in the next coming parts of the thesis.

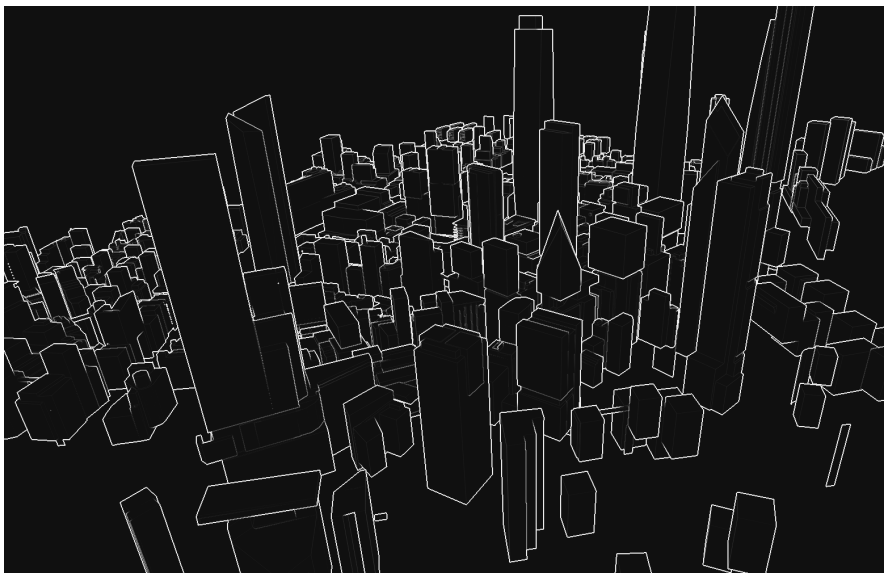


Figure 3.4: This figure shows a city with sky-scrapers and visualizes the rapid change in the depth buffer, i.e Laplacian filter.

3.5 Custom partial derivatives

A custom implementation of partial derivatives was implemented with a three point sample pattern, as mentioned earlier in thesis there are limitations in partial derivatives on how the hardware implements the calculation. The problem will be very visible when two opposite edges are in the same quad. This will be covered in the results in Chapter 5, however due to the time scope and the complex integration any performance analysis were not conducted to the custom derivatives implementation.

Chapter 4

Evaluation Method

In this chapter a short description about the scientific method will be clarified and the expected outcome from the experiments. Also any relevant information about the system setup and the complexity of the experiments.

4.1 Selection of method

In order to answer the research question asked earlier performance measurements, a survey and experiments were done. Two prototypes were implemented based on the concept of distance to edge by storing the distance data in a pre-pass and resolving it in a post-pass. Because of the broad research already done in the field on other state of the art Anti-Aliasing solutions a quantitative method with focus on gathering performance measurements, temporal stability tests and visual differences will evaluate the solutions against each other on the same hardware. By choosing a unified hardware the result gathered from this thesis will be easy to apply to other agents within the field. This thesis will do a experiment on the visual result in context of quality and temporal stablilty. The experiment is to test the null hypothesis that there are no significant difference between the given techniques in terms of quality and temporal stability.

4.2 Setup and expected outcome

The experiments will be done on an Xbox One and Playstation 4. There are several differences when it comes to the hardware on Xbox One and Playstation 4, however the most relevant for the nature of this thesis is the GPU.

Xbox One: 2133MHz DDR3, 12 CU 853MHz
Playstation 4: 5500MHz GDDR5, 18 CU 800MHz

Even with the Xbox One's extra clock speed, the Playstation 4 is expected to perform faster, beacuse of the nature of the implementation. It is not so calculation heavy as it is bandwidth dependent, mostly because of the deferred rendering.

In terms of performance between the given techniques and the given prototype, it is expected to perform as fast as FXAA, but be much more temporal stable and giving a better visual result than the other techniques.

4.3 Experiment

The experiments will be done on actual game content of Star Wars: Battlefront, however no visual result will be shown in this thesis from those test cases because of the NDA. The experiment will run on different scenes with different complexity, different resolutions and with different techniques. The techniuqes tested are, the given prototype from this thesis, FXAA, SMAAt2x and no Anti Aliasing turned on. The same scene and time will be used to create as reliable and deterministic a result as possible. The raw measurements will be tested against the new Prototype by calculating a relative ratio between the given prototype and the other techniques. This ratio will show the increase or decrease of performance for any given technique relative the one given in this thesis.

Measurements on the pre-pass will be done but the raw data will not be presented in this thesis because the data consists of core functionality and results from the game and engine itself. The author of this thesis will discuss and evaluate the behavior of these measurements. This is because no data can be released which can be reversed calculated giving confidential information about the system protected by the non-disclosure agreement between the author and the company.

A Chi-square calculation will test the null hypothesis and the data will be collected by the survey with a Four Alternatives Forced Choice (4AFC) preference experiment. This is to show if there is a significant trend pointing in the favor of the given technique presented in this thesis. The participants will be introduced to the subject of the thesis and what may be different between given pictures. They will see all four choices at the same time but no participant will get the same combination of the pictures. Two different set of pictures will each participant see, one to test the visual quality and one to test the temporal stability. The Chi-square can be calculated by the following equation:

$$\chi^2 = \sum_i \sum_j \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

24 participants were contributing to the experiment, all participants were adults (with age 18+) and had normal vision or corrected to normal vision. No additional data about the participants were gathered during the survey, all participants remained anonymous.

Chapter 5

Results

In this chapter the results will be presented in form of measurements, hyperlinks to clips in motion and images. The techniques evaluated against each other are Prototype 1, Prototype 2, FXAA, SMAA and SMAA2tx. First will be several tables showing performance measurements from different setups, followed by results from the survey on visual quality and temporal stability and Appendix A with visual results. Finally ending with differences in both prototypes and also laplacian filtering and custom derivatives.

5.1 Performance measurements

All the following data points are measured in milliseconds (ms) and are based on 600 frames, which includes average, minimum (min), maximum (max) and standard deviation (StD) on the execution time. P1 Ratio (Prototype 1) and P2 Ratio (Prototype 2) refers to the difference between given techniques. The bold notation indicates the fastest average bewtween the given techniques during one setup. There are two different real scenes from Star Wars: Battlefront, refered to as Scene A and Scene B. Every scene was measured with two different settings on the resolution. One setting on full HD 1920x1080 pixels, refered to as 1080p, and the other one with 1280x720 pixels resolution, refered to as 720p. The measurements were done on both Xbox One and Playstation 4, in a total of eight measurements. The different Anti-Aliasing techniques evaluated and measured are the following, Prototype 1 and Prototype 2, FXAA in three different settings refered to as low, medium and high. These different settings on FXAA is basically the blend range on an edge, for more information the reader should read the FXAA documentation [7]. The standard implementation of SMAA and the temporal double buffering variant SMAAt2x will also be measured against each other.

The following tables with measurements will be presented in the same order:

- TABLE 1 SCENE A 720p Xbox One
- TABLE 2 SCENE A 720p Playstation 4
- TABLE 3 SCENE B 720p Xbox One

- TABLE 4 SCENE B 720p Playstation 4
- TABLE 5 SCENE A 1080p Xbox One
- TABLE 6 SCENE A 1080p Playstation 4
- TABLE 7 SCENE B 1080p Xbox One
- TABLE 8 SCENE B 1080p Playstation 4

Table 5.1: SCENE A 720p Xbox One

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.18**	0.17	0.23	0.01	1	2.33
Prototype 2	0.42	0.39	0.51	0.02	0.43	1
FXAA low	0.22	0.21	0.27	0.01	0.82	1.91
FXAA medium	0.29	0.28	0.34	0.01	0.62	1.45
FXAA high	0.31	0.30	0.33	0.01	0.58	1.35
SMAA	0.66	0.62	0.81	0.04	0.27	0.64
SMAAt2x	1.01	0.91	1.30	0.09	0.18	0.41

Table 5.2: SCENE A 720p Playstation 4

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.12**	0.11	0.13	0.00	1	2.08
Prototype 2	0.25	0.24	0.26	0.00	0.48	1
FXAA low	0.16	0.16	0.16	0.00	0.75	1.56
FXAA medium	0.20	0.20	0.21	0.00	0.6	1.25
FXAA high	0.22	0.22	0.22	0.00	0.55	1.14
SMAA	0.42	0.39	0.45	0.01	0.29	0.6
SMAAt2x	0.60	0.56	0.66	0.02	0.2	0.42

Table 5.3: SCENE B 720p Xbox One

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.20**	0.17	0.31	0.02	1	2.3
Prototype 2	0.46	0.40	0.61	0.04	0.43	1
FXAA low	0.21	0.18	0.29	0.02	0.95	2.12
FXAA medium	0.27	0.25	0.32	0.02	0.74	1.7
FXAA high	0.29	0.27	0.35	0.01	0.69	1.59
SMAA	0.58	0.49	0.80	0.09	0.34	0.79
SMAAt2x	1.03	0.83	1.54	0.15	0.19	0.45

Table 5.4: SCENE B 720p Playstation 4

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.11**	0.11	0.13	0.00	1	2.27
Prototype 2	0.25	0.24	0.26	0.00	0.44	1
FXAA low	0.13	0.13	0.13	0.00	0.85	1.92
FXAA medium	0.17	0.17	0.18	0.00	0.65	1.47
FXAA high	0.18	0.18	0.19	0.00	0.61	1.39
SMAA	0.33	0.30	0.37	0.01	0.33	0.76
SMAAt2x	0.51	0.47	0.57	0.02	0.22	0.49

Table 5.5: SCENE A 1080p Xbox One

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.41**	0.38	0.57	0.04	1	2.32
Prototype 2	0.95	0.88	1.66	0.08	0.43	1
FXAA low	0.48	0.46	1.20	0.03	0.85	1.98
FXAA medium	0.64	0.62	0.74	0.02	0.64	1.48
FXAA high	0.69	0.68	0.79	0.02	0.59	1.38
SMAA	1.82	1.67	2.77	0.18	0.23	0.52
SMAAt2x	2.59	2.43	3.20	0.19	0.15	0.37

Table 5.6: SCENE A 1080p Playstation 4

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.19**	0.17	0.22	0.01	1	2.84
Prototype 2	0.54	0.52	0.58	0.01	0.35	1
FXAA low	0.34	0.34	0.34	0.00	0.56	1.59
FXAA medium	0.43	0.42	0.44	0.00	0.44	1.26
FXAA high	0.47	0.46	0.48	0.00	0.4	1.15
SMAA	0.85	0.78	0.94	0.04	0.22	0.63
SMAAt2x	1.26	1.12	1.70	0.10	0.15	0.43

Table 5.7: SCENE B 1080p Xbox One

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.42**	0.38	0.58	0.04	1	2.29
Prototype 2	0.96	0.90	1.27	0.07	0.44	1
FXAA low	0.44	0.41	1.15	0.05	0.95	2.18
FXAA medium	0.57	0.55	0.72	0.03	0.74	1.68
FXAA high	0.61	0.58	0.73	0.03	0.69	1.57
SMAA	1.59	1.42	2.50	0.18	0.26	0.6
SMAAt2x	2.35	2.12	3.49	0.23	0.18	0.41

Table 5.8: SCENE B 1080p Playstation 4

Technique	Average	Min	Max	StD	P1 Ratio	P2 Ratio
Prototype 1	**0.24**	0.23	0.26	0.01	1	2.29
Prototype 2	0.55	0.53	0.57	0.01	0.44	1
FXAA low	0.27	0.27	0.28	0.00	0.89	2.03
FXAA medium	0.36	0.36	0.37	0.00	0.67	1.53
FXAA high	0.39	0.38	0.40	0.00	0.62	1.41
SMAA	0.65	0.60	0.75	0.03	0.37	0.85
SMAAt2x	1.03	0.69	1.13	0.04	0.23	0.53

5.2 Visual result

The images presented are not associated with any game content produced for the game nor any other game.

In Appendix A of this thesis, an actual asset from the game Star Wars: Battlefront is included. The reader should note, however, that anything and everything related to the asset is subject to change until the release of the game. This asset does in no way represent the final result, but is included as an addition to the thesis result.

In Table 5.9 the results of 24 participants looking at cropped versions of the pictures in Appendix A are presented. They are told to pick the one picture they consider has the highest amount of quality when it comes to the edges. P1 is refered to Prototype 1 and Chi-sq as the Chi-square, finally the p-value is the significance level.

Table 5.9: Survey on static image, 24 participants.

	AA off	FXAA	SMAAt2x	P1	Chi-sq	p-value
Observed	2	6	9	7	4.33	0.23
Expected	6	6	6	6	0	1

5.3 Temporal stability

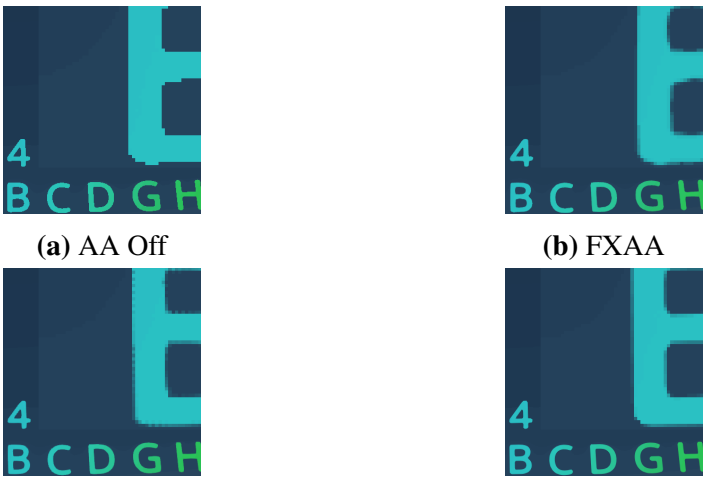
Temporal stability is hard to present in a simple document when it is about a picture of alpha edges in motion, the reader will get links to open in a web-browser to see the following presented images as animated .gif files. The following techniques will be presented, one picture with no Anti-Aliasing, one with Prototype 1, one with FXAA medium setting and lastly SMAAt2x. Figure 5.1 gives a preview of the result. The following links will give the reader the animations: AA off[1], FXAA [2], SMAAt2x[3] and Prototype 1[4].

[1]AA off `http://i.imgur.com/Z7hpPvU.gif`
[2]FXAA `http://i.imgur.com/8BccAkh.gif`
[3]SMAAt2x `http://i.imgur.com/ZQsqdtO.gif`
[4]Prototype 1 `http://i.imgur.com/kB5jsaF.gif`

Table 5.10: Survey on temporal stability image, 24 participants.

	AA off	FXAA	SMAAt2x	P1	Chi-sq	p-value
Observed	1	3	6	14	16.33	0.001
Expected	6	6	6	6	0	1

In Table 5.10 the results of the same 24 participants as in 5.9 looking at the animated version of Figure 5.1. They are asked to choose only one of the pictures which they feel got the most temporal stable edges i.e. smooth edges during movement.



(a) AA Off **(b)** FXAA

(c) SMAAt2x **(d)** Prototype 1

Figure 5.1: A preview of the animateable result of temporal stablity.

5.4 Prototype 1 versus Prototype 2

When it comes to visual differences on both prototypes conducted during this thesis using two edges instead of one will fix the problem with storing the wrong distance. By storing the closest to the neighbor geometry instead on the one to the actual edge. This fix can be seen in Figure 5.2 however one new artifact gets visible on the fixed picture. One can see the gradient suddenly restarts and this is because of under-sampling and will be explained in the discussion section of this thesis.

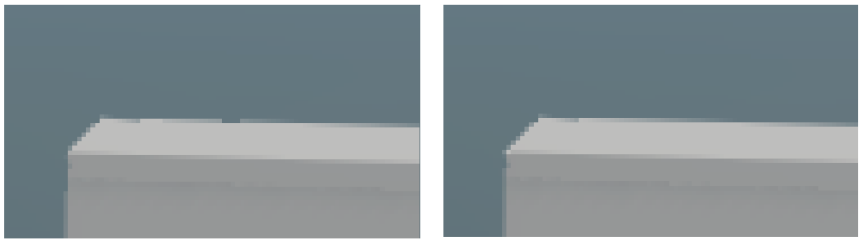


Figure 5.2: Prototype 1 to the left and Prototype 2 to the right, figure showing when there are 2 edges in the same pixel.

5.5 Gradients and Laplacian filter

The following image shows the gradients, i.e. the length to the edge visualized as a gradient however the mageneta color and the green color as seen in Figure 5.3, represent if the edge is inside the pixel or if it belongs to a neighbor which needs a immediate neighborhood lookup in the post-pass.

Prototype 2 did use a Laplacian filter on the depth buffer which can be seen in Figure 5.3. This show the intensity of the epsilon needed for filtering out certain unwanted edges. With this additional data, Prototype 2 can solve the the missed pixels in Figure 5.2.

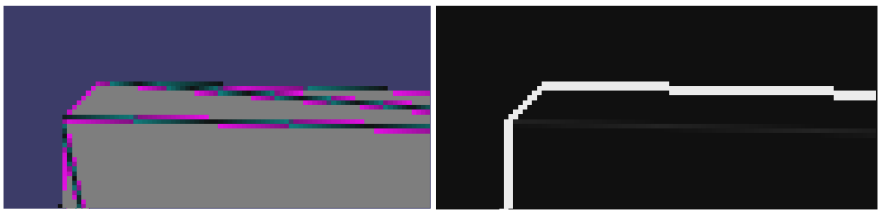
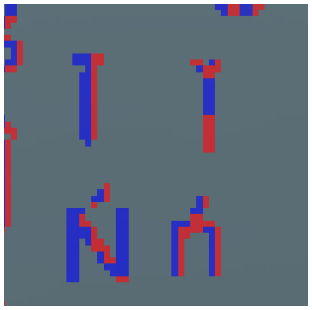


Figure 5.3: Image to the left shows the distance to the edge in each pixel, the colors represent if it is inside the very pixel or the distance belongs to the neighbour. The image to the right shows a Laplacian filter.

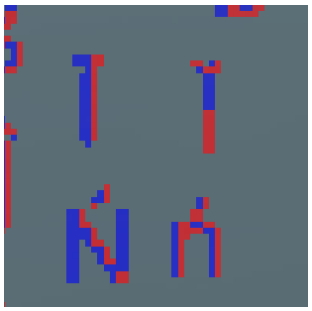
5.6 Derivative variations

The hardware has two different implementations for solving derivatives in a pixel quad, one coarse and one fine version. The difference between them are in the coarse one all pixels gets the same value. In the fine one the columns or row gets different values depending on which axis is locked. Figure 5.4 shows the custom three sample points implementation giving all pixels unique derivatives and making the direction artifact

corrected. As can be seen in Figure 5.4, by visualizing the negative and positive direction to the edge one can see the underlying quads in the coarse version and the rows and columns in the fine version and finally the result of the custom version.



(a) Coarse derivatives



(b) Fine derivatives

(c) Custom 3 point sample implementation

Figure 5.4: Figure showing the difference between derivative implementations.

Chapter 6

Discussion

This chapter covers all parts of the experiment, the prototype and any other and will give some discussion and deeper analysis of the content. It will also cover threats to validity of this thesis.

6.1 Temporal stability

We will start by discussing the core essence of this thesis, temporal stability. Almost all old state of the art implementations do not handle temporal stability well, for the exception of super sampling with an large cost. Multi sampling do also handle temporal stability well but is restricted to only triangle geometry which leads to aliased alpha edges. This means that MSAA often needs an extra technique on top of when it encounter some scenes and situations. One must remember MSAA is very expensive as well and with the common setting four times multi sampling the pixels are still standing out in the image.

The result from the survey in Table 5.10 shows that the p-value is below 5% (0.05) which indicates there is a significant difference in the result, this rejects the null hypothesis about there is no significant difference between the given techniques. The data in Table 5.10 indicates that our solution yields very good results an may be considered as a temporal stable solution.

FXAA is not temporally stable at all, this is because it only works on an already temporally unstable data set. It does not has any extra information to take into account about the orign of the edge, it only approximate where edges tend to appear that very frame. SMAAt2x however is a little bit more stable temporally. This is because it uses two frames with a little jitter to account and get sub pixel information.

As seen in Figure 5.1, our implementations are temporally stable with very high precision. If there is space in the gbuffer one can easily achieve results and gradients at a higher degree of state of the art solutions and even quality as high as the movie industry. Because the edges use the same data as interpolation of a texture, they will be

almost as stable as an linear interpolated texture depending on the numbers of bits used.

6.2 Prototype 1

The first prototype yields very satisfying results and has a very simple structure. Its simple implementation and integration is an easy starting point when it comes for testing it in various systems. However as explained it has its weaknesses, as it does blur edges shared between triangles. This can be an undesired behavior, one simple solution to solve this problem is if the sum neighbouring pixels distances to an edge, and see if they add up to one. Then there is a very high chance that the specific edge is between two neighboring triangles which is not an edge we do want to blur. However the large problem arises as seen in Figure 5.2, where the algorithm chooses an undesired edge, the one pointing inwards e.g. this case has two edges in same pixel. This will lead to with the simple fix presented above will remove the very existence of the edge in that pixel.

6.3 Prototype 2

The second prototype accounts for these two problems with two edges inside a pixel and choosing the right one to blur. However this implementation relies on using support of parallel execution on GCN hardware with compute shaders running at the same time as other pipeline code. In this thesis the execution is only sequential and this will be discussed in future work.

The strong side of this implementation versus other post-pass solutions is that after the geometry has been stored in a gbuffer, all computations are free to be executed when the user prefers to do so. There are no relationships with anything else except in this case the depth buffer which means the code can be pushed parallel to a not that heavy part of the frame.

One disadvantage however is that compute shaders do not have access to the texture cache on the GPU as pixel shaders does, this because of how the GPU handles resources during execution. Higher performance on a pixel shader instead of a compute shader for a sequential execution is expected. This is expected because of the nature of the problem with more reads than instructions. This was not included in the scope of this thesis, however the measurements suggest it. Prototype 2 is only a proof of concept that it can be done in parallel with few relationships to other resources if it were to utilze the GCN asynchronous features.

6.4 Measurements

When it comes to performance the different techniques are scattered in a large spectrum, with the relationship of execution time versus visual quality. However this might not be true anymore, as our implementation is very fast and yields very good results for many situations, such as alpha edges, triangle edges and temporal stability. The numbers in the tables in Chapter 5, yields as expected with the Play Station 4 noticeble faster than the Xbox One in all scenarios. However one thing to note is that there is almost no difference between the scenes and with the same setup, even with the statement that the scenes differed much in complexity. This might be because the code is flattened and a very small potion of the code diverges, however GCN has got several features to handle branches very well.

However looking at the performance ratio in Table 5.1-5.8 this indicates that the implementation of Prototype 1 is the fastest one, and Prototype 2 do take some extra time to solve two edges. But the important thing to note is that Prototype 2 consists of two passes, one which can be parallel with the regular pipeline. More measurements on this is needed to do any conclusions of Prototype 2.

One very important matter to discuss is the performance on the pre-pass. Due to the confidential nature of the thesis no numbers will be presented because the gbuffer pass includes so much information, but the numbers themselves are not that important for this part. Simply presenting how the pre-pass behaves with different settings will bring value to the results. When only using simple calculations with derivatives no extra execution time were measured except the time for the extra bandwidth needed for this implementation. However when using perpendicular edge calculations some calculations could not be latency hidden. This may lead to problems when much geometry is passing the pipeline such as tessellation etc. But due to the complex nature of the engine and indeed this prototype is not alone in the pipeline, which means much work can be done here. Depending on the bounds of the pipeline, different results will be acquired, e.g computation bound, texture bound etc. This will not be covered in this thesis but the reader should be aware that different systems will look may behave entirely different. Especially in this particular part of the system when it comes to deferred rendering with different needs. However the author suggests that if one prioritize this feature and makes proper space for it in the main gbuffers and with the other parts that executes in the pre-pass this implementation can be totally “free” in that perspective in an engine as Frostbite.

6.5 Visual result

For the simplicity we will discuss the visual result in two parts, one for triangle edges and the other for alpha edges. First we will look at the survey then we will start with triangle edges where there are some significant issues that needs to be explained and evaluated by looking at Figure 5.2.

In Table 5.9 the null hypothesis cannot be rejected, because of the results do not show any significant difference. This may indicate that the techniques are not differential when it comes to visual quality, or it may indicate that more data is needed to draw any conclusions. However to the author's knowledge some users do prefer to turn Anti-Aliasing of during gameplay for the reason of a sharp image, so it's not impossible that the participants who choose No AA in 5.9 did not misunderstand the question. The abstract concept of visual quality is hard to interpret even if our solution follows an analytical mathematical model of reality. However the important thing to note about the result of no significant difference is that our solution may indicate that it did not produce any worse quality.

With triangle edges when the conditions are right one can argue that we will not need higher resolution monitors anymore because of the good results, however this is not the case for all situations. When discussing rendering a triangle is a topic one might think of a simple big rasterized triangle on the screen, and in this situation the presented prototype will show its greatest potential. However there is a huge problem with under-sampling, which Figure 3.3 shows us. There is a reason why there is no Prototype 3 with three edges in one pixel and it is because it would not solve this problem. A simple explanation about the origin of the problem can be in the term micro triangles, blurring edges and making the image look as precise as possible can be achieved by having a relationship between pixels and having a regularity with the edge. If we have small triangles, e.g. over-tessellation or triangles far away, all information will be inside one pixel alone making the problem hard to solve. Even impossible to solve if there is geometry between the pixels which does not get rasterized, which it probably is in these situations. That is what we refer to as under-sampling.

However if we talk about under-sampling and look at Figure 3.3, this situation can be any triangle with an extreme angle or the tip of the triangle not getting rasterized. This will lead to under-sampling and can be seen in Figure 5.2, on the top edge where the gradient restarts in the Prototype 2 version. This is because the triangle on top does not get its tip rasterized, but the bottom triangle's edge will be instead. In this particular case it might not be noticeable. However there are cases when the irregularities will be seen, and when it comes to images in motion and temporal stability this artifact can be counter productive making it very distinct. This is a sub-pixel artifact in the technique itself, which will lead to irregularities on the edge. This cannot be solved without additional sample points or coverage points. If some complexity is added in the pre-pass

near neighborhood situations can be detected and accounted for, but because of the nature of the problem it might not be in the near neighborhood. This is a very large problem for the solution and it suffers from the same thing it is meant to correct.

When it comes to alpha edges it looks different. This is because the problem is based on a different approach which can not be applied to triangle data. Here no data is missing because everything is stored in a texture and will always be reachable as an extra dimension of information. Alpha edges will not suffer from under-sampling, but there is the other problem with 2x2 pixel quads returning the same derivatives for multiple pixels. This might not sound like a problem but this will lead to edges being in half the intended resolution and for non-moving static images it mostly are not a problem. However the problem emerges as seen in figure 5.4, when a pair of pixels points in different directions but the derivatives are shared in the same quad, this will lead to one side being Anti-Aliased correctly and the other side blending in the wrong direction and still being aliased. Even worse when the picture is in motion when the directions will alter from side to side based on which pixel in the quad that will determine the direction. The custom derivatives that were implemented solves these problems, however due to the complexity of the engine and integration it could not be included in the performance measurements. One can do the assumption that it is very costly with two extra samples, and it will be something that needs extra evaluations in future work.

6.6 Threats to validity

Due to the unified hardware of the new generation of consoles, Xbox One and Playstation 4, all results conducted during this thesis will be the same for other users that will implement this technique. However with the complexity of game engines, games or applications some variations in performance is expected. Because of the integration in a real complex engine as Frostbite and a AAA title as Star Wars: Battlefront this thesis with measurements will be a valid benchmark for the rest of the industry within the domain of Anti-Aliasing. The survey on visual quality and temporal stability increases the validity of the thesis, however a larger survey could bring value to the thesis, because of no significant difference was observed during the visual quality experiment.

Chapter 7

Conclusion and future work

In this last chapter the author will give his conclusion of this thesis and propose several fields of future work in some areas.

7.1 Conclusion

The solution is an high degree analytical model with very temporal stable and high precision images making them look smooth and natural in motion. With also good performance, faster than the fastest state of the art solutions in the industry. With good visual quality on the images the technique can be the choice as an Anti-Aliasing solution for some projects. However in some situations the technique will suffer from under-sampling problems and produce irregularities on the edges, making the solution counter-productive.

Right now it uses features only supported by GCN hardware such as Xbox One and Playstation 4 for solving triangle edges. However this feature can easily be removed from the solution making it implementable on a large variety of hardware. If this is the case prototype 1 can be an excellent complement to Anti-Aliasing solutions such as Multi Sampling which can not solve alpha clipped edges. On alpha clipped edges such as foliage it yields very good results with features like procedurally generated animations and will give the scene a very smooth motion as seen in Figure 5.1.

7.2 Future work

By utilizing the parallel power of GCN hardware when it comes to Compute Shaders and regular pipeline, do some additional performance measurements on Prototype 2. This will bring the given technique to new levels in the industry, if this yeilds good results one can consider putting all code in a Compute Shader and just a look up and re-sample in the pre-pass. Also some evaluations on the performance differences using a compute shader versus a pixel shader with the focus on the texture cache would be of value. Additionally do some tests between a full screen quad and a full screen triangle

as a pre-pass resource on the GPU.

An implementation on a different engine or even game will give value to this thesis and the field, e.g if it is possible to also latency hide computations in other engines in the pre-pass. Additional even test two dimensional rendering engines and not only three dimensional.

By testing the given prototypes it in combination with other state of the art Anti-Aliasing solutions such as FXAA or MLAA to evaluate the visual result, temporal stability and also the performace may give another view on how to solve aliasing artifacts. Either by pass on pass approach or as a hybrid taking some of the concepts from the given solution.

A deeper analysis between the difference of gradients on the state of the art solutions for evaluation on the visual representation of all techniques, and by doing that also analyze different bit values of distances in the given prototypes in this thesis to evaluate the relationship between precision and performance in given prototypes.

Acknowledgments

I would like to acknowledge and show my biggest gratitude to Robert Khil, Lead Rendering Engineer on the Star Wars Battlefront project, he has been a true role model and given me several interesting discussions. The second person I want to thank is Christopher Birger also on the rendering team for all the tips and trick while learning the system and his ultra friendliness making my integration a joy. I would also like to show my gratitude to Jonas Kjellström, Technical Director on Star Wars Battlefront for his enthusiasm and interest in my progress and work.

I would like to thank every person I have interacted with on EA DICE for their friendliness and the joy of putting faces onto my role models.

I want to thank Dr. Veronica Sundstedt and Adam Nilsson for their thoughts and directions in my writing of this thesis.

Last but absolutely not least I want to personally thank the character artists who were my desk neighbors during my thesis. They were awesome and made my time literally a blast.

Appendices

Appendix A

Bonus content

The following pictures do not represent the final result of the game, they are just added to bring extra value to this thesis.

These images are intended to be seen in full HD i.e 1920x1080 resolution.

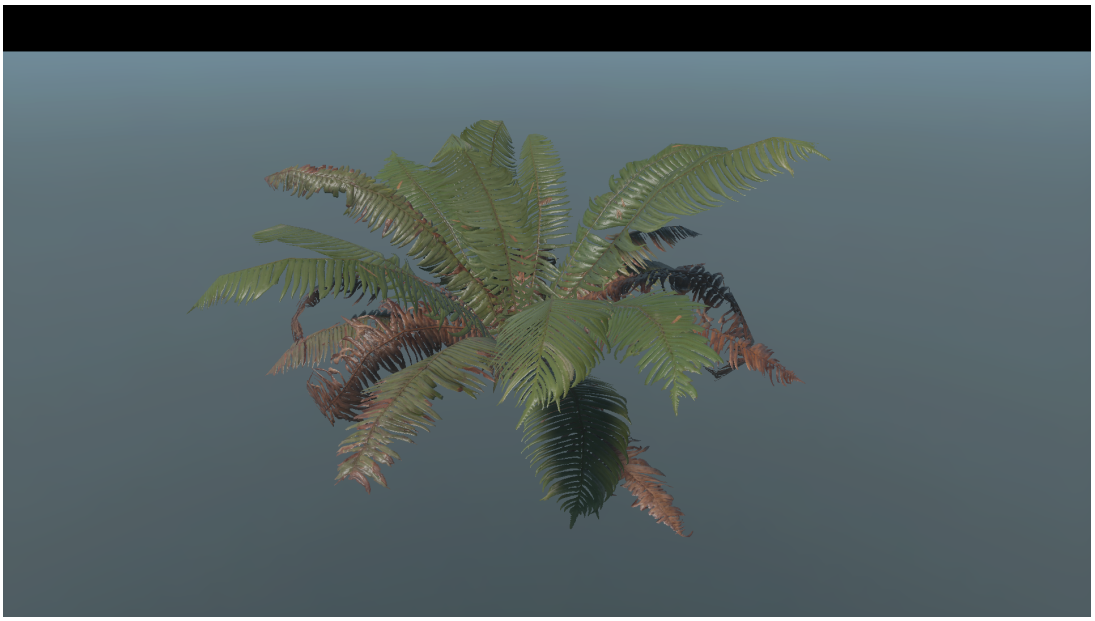


Figure A.1: No AA

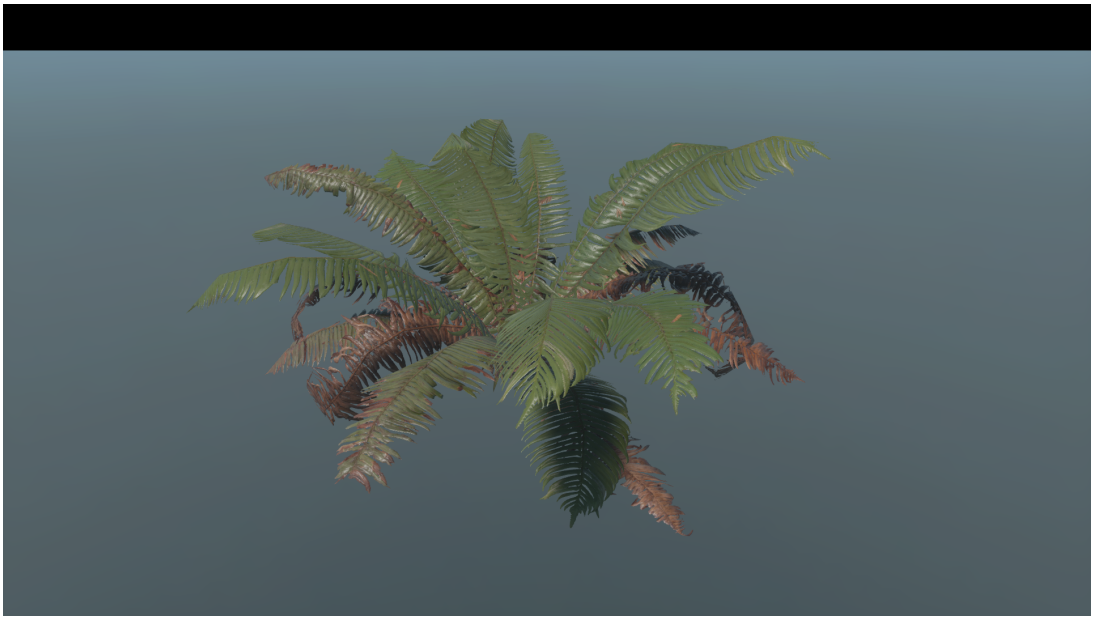


Figure A.2: FXAA

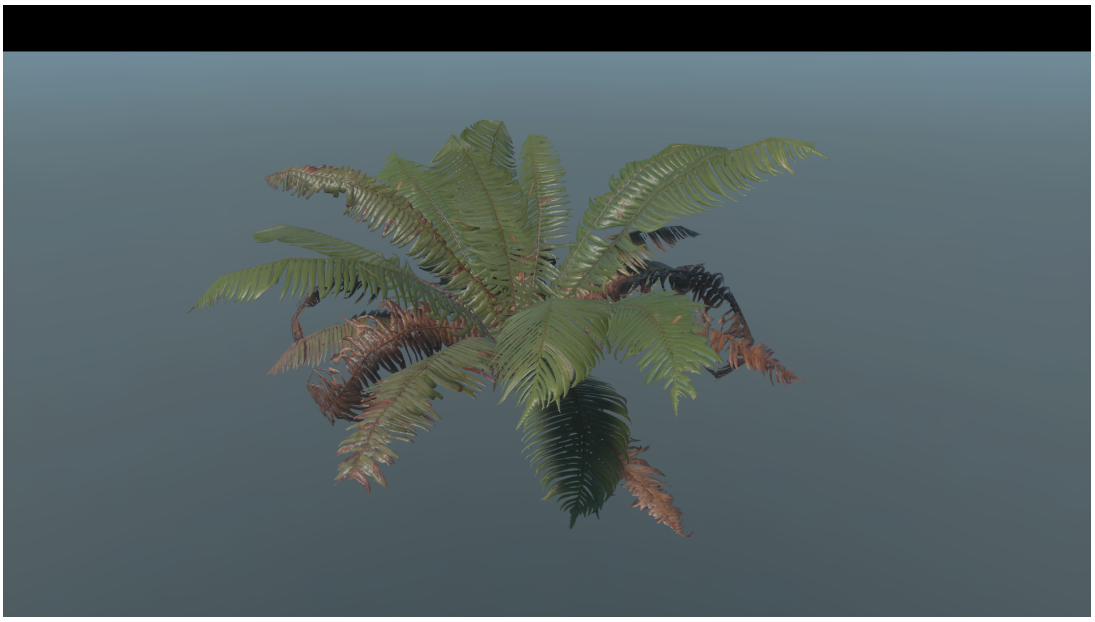


Figure A.3: SMAA2tx

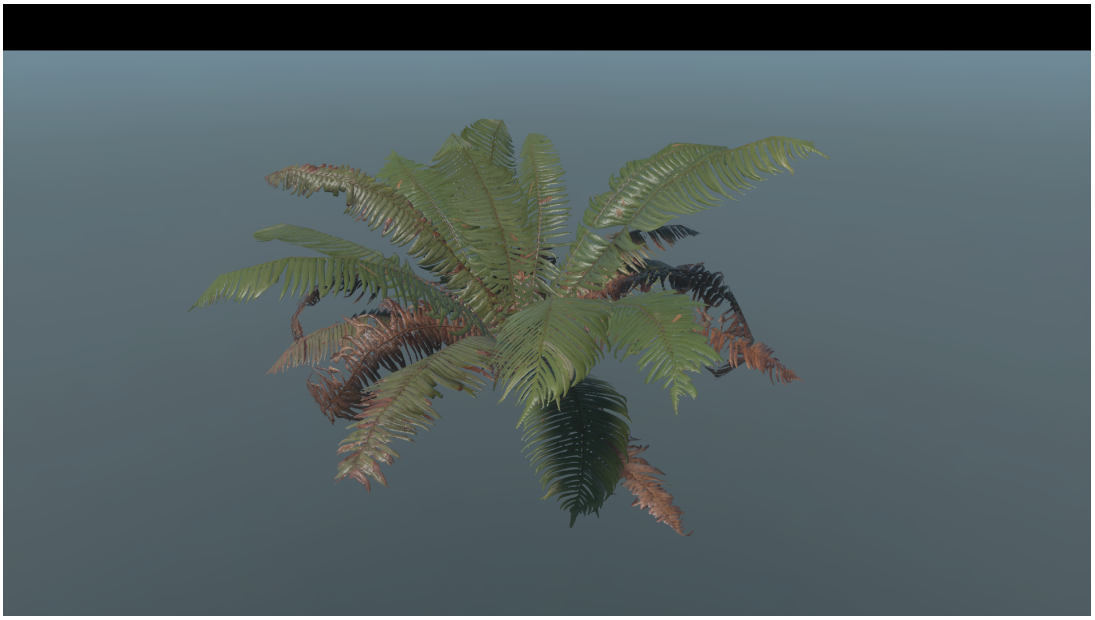


Figure A.4: Prototype 1

Figure A.5: Laplacian filter for edge evaluation

References

[1] AMD. Southern islands series instruction set architecture. http://developer.amd.com/wordpress/media/2012/12/AMD_Southern_Islands_Instruction_Set_Architecture.pdf, 2012. [Online; accessed 6-April-2015].

[2] Johan Andersson. Frostbite rendering architecture and real-time procedural shading texturing techniques. *Game Developer Conference*, 2007. DICE.

[3] Kristof Beets and Dave Barron. Super-sampling anti-aliasing analyzed. *3dfx*, 2000. Beyond3D.

[4] Michal Drobot. Low level optimizations for gcn. *Digital Dragons*, 2014.

[5] Chris Green. Improved alpha-tested magnification for vector textures and special effects. *Siggraph*, 2007. Valve.

[6] et al. Jorge Jimenez. Filtering approches for real-time anti-aliasing. *Siggraph*, 2011.

[7] Timothy Lottes. Fxaa. *NVIDIA*, 2 2009.

[8] Layla Mah. The amd gcn architecture - a crash course. *AMD Developer summit*, 2013. AMD.

[9] Hugh Malan. Distance-to-edge aa (deaa). *Siggraph*, 2011. Filtering Approches for Real-Time Anti-Aliasing.

[10] Bruno A. Olshausen. Aliasing. *PSC 129 - Sensory Processes*, 2000.

[11] Emil Persson. Geometry buffer anti-aliasing. *Siggraph*, 2011. Filtering Approches for Real-Time Anti-Aliasing.

[12] Emil Persson. Geometry buffer anti-aliasing (gbaa). http://www.humus.name/index.php?page=3D, 2011. [Online; accessed 6-April-2015].

[13] Emil Persson. Low-level shader optimization for next-gen and dx11. *Game Developer Conference*, 2014.

[14] Alexander Reshetov. Morphological antialiasing. *Proceedings of the Conference on High Performance*, pages 109–116, 2009.

[15] Michael Schwarz and Marc Stamminger. Multisampled antialiasing of per-pixel geometry. *EUROGRAPHICS*, 2009. University of Erlangen-Nuremberg.

[16] Jorge Jimenez, Jose I. Echevarria, Tiago Sousa and Diego Gutierrez. Smaa: Enhanced subpixel morphological antialiasing. *Computer Graphics Forum*, 31, 2012.

[17] Tobii Technology. Tobii eye tracking an introduction to eye tracking and tobii eye trackers. 2010.