Improved Sampling for Temporal Anti-Aliasing

(A Sobel Improved Temporal Anti-Aliasing)

Christian Alexander Oliveros Labrador
christianol_01@hotmail.com

May 18, 2018

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Michael Doggett, michael.doggett@cs.lth.se
Examiner: Flavius Gruian, Flavius.Gruian@cs.lth.se

Abstract

Anti-aliasing is a key component of modern 3D computer-generated imagery. For Real-Time image generation in applications such as games, it is important to increase the sampling rate per pixel to improve overall image quality. But increasing sampling can be expensive, especially for current Deferred Rendering architectures. An innovative solution to this issue is the Temporal Anti-Aliasing (TAA) technique which combines samples from previous frames with the current frame's samples to effectively increase the sampling rate. In this thesis, we will explore methods to improve the quality of TAA by using edge detection, of both color and depth, and triangle indexing to ensure only samples belonging to the current frame pixels are blending together. Our objective is to reduce ghosting and other TAA artifacts created with current implementations. Quality improvement will be evaluated by comparing TAA generated images to ground truth images generated by using much higher sample counts that would not be practical in real-time. The improved TAA was tested using MSE, PSNR, and SSIM against the original implementation and other current Anti-Aliasing techniques. The results obtained showed that the improvements applied to TAA enhanced the image quality above existing techniques with PSNR values around 39 and SSIM values above 0.99.

Keywords: TAA, Sobel, Anti-Aliasing, Triangle Indexing, TRAA

Acknowledgements

"The day we stop exploring is the day we commit ourselves to live in a stagnant world, devoid of curiosity, empty of dreams." – Neil deGrasse Tyson

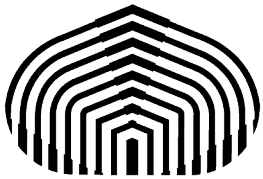
I have so many people to thank for this amazing adventure. Coming from Venezuela, more than 8000 km away; being the first time I am away from home for such a long time; the first time I take classes only on English; and being the first exchange student to perform a Master Thesis in the Computer Science Department at Lund University, this has been a life-changing experience.

I would like to thank all my family, especially, my dad, Wilmer Oliveros; my mom, Raixa Labrador; and my brother, Jean Pierre Oliveros for all their help and support during this trip. Also, I would like to thank all my friends that are all over the world, thank you for helping me be who I am right now and for being with me this whole time.

I would like to thank my Lund University supervisor, Professor Michael Doggett, for giving me this amazing opportunity and for helping me through it. Also, I would like to thank Pierre Moreau for helping me whether I was stuck. Also, I would like to thank my Universidad Simón Bolívar supervisor, Professor Angela Di Serio; and my Computer Science Coordinator, Professor Marlene Goncalves, for helping and supporting me through this incredible journey.

Finally, I would like to thank Lund University and Universidad Simón Bolívar, with special thanks to the International Office LTH; the Department of Computer Science at Lund University; the Computer Science Coordination at Universidad Simón Bolívar; and the Direction of International Relations and Cooperation at Universidad Simón Bolívar, for letting students take advantage of this type of opportunities of academic and self-growth.

LUND UNIVERSITY



UNIVERSIDAD SIMÓN BOLÍVAR

Contents

Chapter 1
Introduction

Modern Computer Graphics are based on rendering scenes that are made of objects represented as models composed of primitive polygons, the triangle being the most common one used. This is to take advantage of their simplicity and all their geometric properties to create optimal algorithms to handle their rendering. Triangles are composed of three vertices, each of them which consists of a position and other parameters associated with them, i.e. the color or normals of the triangle, for interpolation.

When we want to render the objects in a scene, we take the vertices and send them to the Rendering Pipeline. There, they are processed and mapped to the pixels of the screen with their respective color.

We have two main uses for this process: Offline Applications, like movies; and Real-time Applications, like video games. Each of them have their requirements and constraints, but for this project, we will only give attention to Real-time Applications.

The focus of this project is to improve the Temporal Anti-Aliasing algorithm, which is a technique that increases the quality of the images after the process of mapping triangles to pixels by mixing frames previously rendered with current ones.

The main requirement would be to render the highest quality possible representation of the scene, with two main constraints: we must render at least thirty frames per second, with no high frame rate loss; and we must work with a limited amount of memory and bandwidth, because we need to be able to run on an average computer or mobile device. [14, 3]

1.1 Problem Definition

Temporal Anti-Aliasing (TAA) is a relatively new real-time technique that provides good results without incurring heavy memory or processing power costs of other techniques. Edge detection and triangle indexing techniques appear as good candidates to improve the quality of the technique by reducing the ghosting and blurring unwanted effects created by

current implementations of TAA.

The aim of this thesis is to improve the Temporal Anti-Aliasing technique by using edge detection, of both color and depth, and triangle indexing techniques to reduce blurring and ghosting without decreasing the quality of the rendered image or incurring in heavy memory or processing power costs.

1.2 Related Work

As the simplest Anti-Aliasing technique, we have Super Sampling Anti-Aliasing (SSAA), it consists on rendering at a higher resolution and then downsampling it to the required resolution. Another technique is the Multi Sample Anti-Aliasing (MSAA), which calculates the color for the final pixel just once [14]. We can learn what would become the base of Temporal Reprojection Anti-Aliasing (TAA or TRAA) in the papers Accelerating Real-time Shading with Reverse Reprojection Caching by Nehab D., Sander P. V., Lawrence J., Tatarchuk N., Isidoro J. R. [15], in which they describe how pixel shaders could be used to save pixel information from the previous frame and reproject it in the next frame; and Amortized Supersampling by Yang L., Nehab D., Sander P. V., Sitthiamorn P., Lawrence J., Hoppe H. [20] in which they describe how to use the reprojection of old frames in the current one as a method of real time Anti-Aliasing.

Next, we start to see Post Processing techniques like Fast Approximate Anti-Aliasing (FXAA) by Timothy Lottes [10] which uses a form of edge detection to correct aliasing while being compatible with the deferred shading architecture. We also found the Crytek implementation of Temporal Anti-Aliasing (TAA or TXAA) explained by Tiago Sousa on his presentation Anti-Aliasing Methods in CryENGINE 3 [9].

Enhanced Subpixel Morphological Antialiasing (SMAA) by Jorge Jimenez, Jose I. Echeverria, Tiago Sousa and Diego Gutierrez [8] which uses a more complex edge reconstruction technique while being able to work with SSAA, MSAA and a basic form of TAA.

Finally, we have the TRAA implementations of Ke Xu for Uncharted 4 and Lasse Fuglsang for Inside, which implement new advances like the Color Clipping Box and Sharpen Filter. These two last implementations are used as the base of this master thesis. [16, 19]

Chapter 2

Technological Background

In this chapter we will explain which tools were used in this master thesis and why.

2.1 C++ and Bonobo Framework

C++ is a compiled general-purpose programming language with imperative, object-oriented programming and low-level memory management features. It is widely used in Computer Graphics due to its performance, especially on real-time applications, and its wide knowledge base.

The Bonobo Framework is the base of the laboratories of Computer Graphics (EDAF80) and High-Performance Computer Graphics (EDAN35) courses from Lund University. It was developed in C++ and provides a rendering engine that is easy to modify and use.

2.2 OpenGL and GLSL

The Open Graphics Library (OpenGL) is a 2D and 3D computer cross-platform open-source graphics Application Programming Interface (API) that abstracts the programmer from directly interacting with the Graphics Processing Unit (GPU) to achieve hardware accelerated rendering. It provides the programmer with a graphics pipeline to use, which is normally implemented through hardware.

OpenGL Shading Language (GLSL) is a high-level shading language that allows programmers greater control of the graphics pipeline without requiring the use of the OpenGL assembly language or hardware-specific languages.

2.3 MATLAB

MATLAB is a proprietary multi-paradigm numerical computing environment. Commonly used for science, engineering, and economics. It is popular for image processing applications because of its wide library of algorithms for this purpose, including image metrics which are used in this thesis.

Chapter 3

Theoretical Background

In this chapter we will explain all the theoretical information that is the base of this master thesis. From the Computer Graphics Theory to the Image Metrics used.

3.1 Rendering Pipeline

Today's graphics pipeline can be simplified into three steps: Vertex Shader, which moves the geometry associated with the vertices and prepares them for the next step; Rasterizer, which maps the triangles to pixels in the screen, calculates their visibility and interpolates the parameters of the vertices for each pixel covered by the triangle; and the Pixel (or Fragment) Shader which takes the visual pixels from the Rasterizer and colors them.

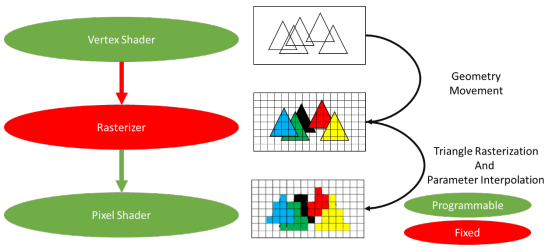


Figure 3.1: Rendering Pipeline, based on EDAF80 5th Lecture. [13]

It is important to note that Vertex and Pixel Shaders are controllable by a programmer using special programs called Shaders. They provide a way for the programmer to control the rendering hardware. In contrast, the Rasterizer is not controlled by the programmer and it is handled entirely by a hardware fixed function. [13]

This Rasterization Process is important for us because it is there where some of the errors corrected by Temporal Anti-Aliasing come from.

3.2 Rasterization Process

During the Rasterization Process, each triangle is tested to establish which pixels are covered by it. While this is being done, each pixel is being tested to find out if another triangle is covering it.

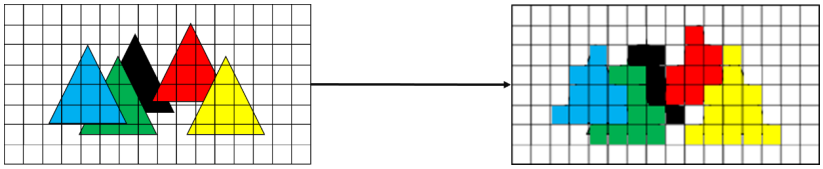


Figure 3.2: Example of the results of the Rasterization Process. *Note*: colors were added to differentiate the triangles, but they would only be added by the Pixel Shader.

Because we are mapping a continuous triangle to a finite number of pixels, we face the problem of pixels partially covered and how to determine what is enough to qualify it as covered. This is solved by calculating if the center of the pixel is covered by the triangle geometry. This process is susceptible to errors due to the precision of the representation used for the vertices.

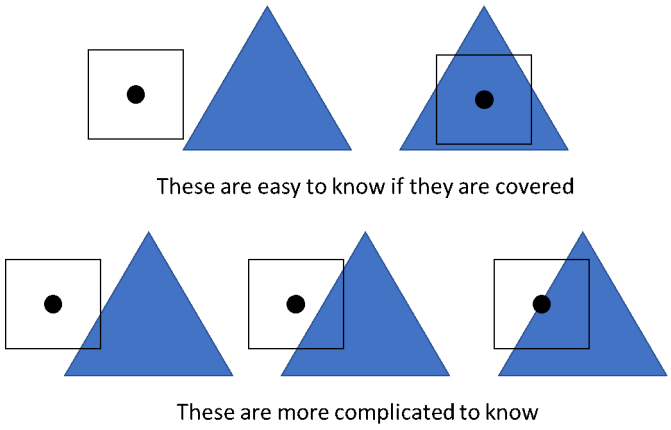


Figure 3.3: Example of the Partial Cover problem, based on EDAN35 second Lecture. [14]

This process shows us that what is rendered to the screen approximates what is being represented in the scene because pixels can only be covered by one triangle at a time. [1, 14]

3.3 Aliasing Problem

When we map a continuous representation to a finite one, it is going to generate errors. As explained by Edward Angel and Dave Shreiner in their book (page 413) [3], we can interpret the rendering process as the sampling of a continuous function $f(x, y)$, which represents the color of the scene at that point, to an $n \times m$ grid of pixels in which we assume that the point f_{ij} is the value of f over a small area; and to reconstruct the f function to display the image to the screen using only what we know from the samples. The mathematical tool used to evaluate the issues of this process is the Fourier Analysis, which states that a function can be decomposed into a set of sinusoids, at possibly an infinite number of frequencies. For two-dimensional image analysis, we can think of the f function as a set of sinusoids at two spatial frequencies.

For this thesis, we will use the First part of the Nyquist sampling theorem as a tool to illustrate why aliasing problems appear and relate to sampling problems.

"Nyquist Sampling Theorem (Part 1): The ideal samples of a continuous function contain all the information in the original function if and only if the continuous function is sampled at a frequency greater than twice the highest frequency in the function.

The Nyquist frequency is defined as one half of the sampling frequency, which is the lowest frequency that cannot be in the data to avoid aliasing."

Taken from Edward Angel and Dave Shreiner book page 415. [3]

As Edward Angel and Dave Shreiner explain, this idealized sampling assumes that we can take an infinite number of samples per sample frequency which we cannot do in practice. The Aliasing problem that computer graphics experience comes from not being able to sample as required by the Nyquist Sampling Theorem, creating ragged edges that appear in the rasterization process (Spatial Aliasing) and jumps between moving objects (Temporal Aliasing), according to Doggett and Wronski [14, 18]. Many solutions have been proposed and used to solve it, e. g. the Super Sampling Anti-Aliasing (SSAA) family of solutions that work on higher frequencies than the required at the cost of more space requirements.

Figure 3.4: Representation versus Aliased Approximation.

3.4 Shadow Mapping and Deferred Shading Architecture

As we know, lights and shadows contribute with spatial information to an image. As humans, we have come to expect that objects react to the lights in a scene, considering their geometry; especially because the shadow that it creates gives us a sense of size.

As explained by Doggett [14], under the Rendering Pipeline based on the Rasterizer, the process of shadow calculation becomes challenging to do. The Rasterizer does not know if objects are covered or not from a light, so we must figure out a method to calculate if an object is in shadows.

The shadow calculations are done through a process called Shadow Mapping, it consists of rendering the scene through each light perspective and then test that against the camera perspective to establish if the object is affected by the light or if it is in shadows.

As we might expect, rendering the scene several times is expensive, so we need a way to reduce the cost as much as we can. The Deferred Shading Architecture provides that solution to only shade, possibly using an operation that could be expensive, only visible pixels to avoid wasting resources shading pixels from geometry that is not visible; it works by first rendering the scene, without shading calculations, to a buffer called the Geometry Buffer. There, information regarding colors, normals, depths, specific object information to interact with lights, etc. is saved for future use.

Then, we perform a technique called Shadow Mapping, which takes advantage of the Deferred Shading Architecture to calculate the way lights affects only visible pixels; we calculate the Shadow Map of every light only performing depth calculations. Afterward, we compute and save the effect of each light using the Shadow Maps.

In the end, we take all the information of the lights, shadows, and the Geometry Buffer to render the light scene.

3.5 Anti-Aliasing

As we have explained, there are two main types of Aliasing, Spatial Aliasing, and Temporal Aliasing; Anti-Aliasing solutions provide improvements against the artifacts created by either of those types at the cost of increased rendering time. For real-time applications this increase of rendering time limits which Anti-Aliasing solutions are feasible to apply.

Another important factor that decides which Anti-Aliasing technique to use is how it behaves with current architectures. For example, old Anti-Aliasing solutions do not work with Deferred Shading.

3.5.1 Super Sampling Anti-Aliasing (SSAA)

This technique consists in rendering the scene at 4 times the size of the screen and then averaging pixels 4x4 to calculate the result [14]. It provides good results but requires more rendering time and heavy memory usage.

3.5.2 Multi Sample Anti-Aliasing (MSAA)

MSAA consist in taking several samples per pixel; on each sample, the depth values are calculated but only one color is calculated for the rasterized triangle. This solution provides good results at the cost of increased memory usage for depth calculations.

The biggest problem this technique has is that it does not work properly with Deferred Shading [14]. This makes it complicated to use with current pipelines, normally requiring other corrections to reduce the artifacts created when applied with Deferred Shading.

3.5.3 Fast Approximate Anti-Aliasing (FXAA)

FXAA is a post-processing anti-aliasing technique that works by detecting edges on the rendered images and then smoothing them. [10]

It is relatively cheap compared to MSAA and provides relatively good results, its smoothing capabilities are limited by the amount of information the edge detection can get on a single pass, and it provides relatively good results for temporal aliasing.

3.5.4 Enhanced Subpixel Morphological Antialiasing (SMAA)

SMAA is a post-processing technique based on Morphological Anti-Aliasing. It works by reconstructing edges and their surroundings to regenerate the subpixel information lost by aliasing. [8]

3.6 Temporal Anti-Aliasing

As explained by Ke Xu and Lasse Fuglsang in their respective presentations [19, 16], the basic principle of Temporal Anti-Aliasing is to mix the current frame being rendered with frames from the past. This is done to increase the number of samples through time rather than only using samples from the same frame.

One such technique is Temporal Reprojection Anti-Aliasing (TRAA), it works by saving the past frames as a History Buffer which it is then reprojected to the present scene and blended to the current frame being rendered. To do this, we take the current frame and look for the color it should have in the History Buffer, this process is called Reprojection.

To implement TRAA we need: Camera Jitter, Velocity Buffer, Frame History Buffer, Clipping Color Box, Sharpen Filter, and Motion Blur.

3.6.1 Camera Jitter

Camera Jitter is applied every frame to preserve information from local regions of surface fragments. If the current frame is static relative to the past ones then the system is losing information that could be used to refine it. [16, 19]

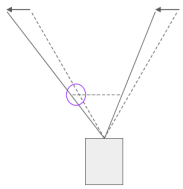


Figure 3.5: Jittering the Projection Process. Image taken from Fuglsand presentation. [16]

The jittering is applied as a translation to the projection matrix using the $HaltonSequence(2,3)$ as the translation deltas. This sequence is used because it generates an irregular pattern for the translations that help preserve more information than a regular pattern and the Halton Sequence provides a cheap pseudorandom pattern generator. [16, 19]

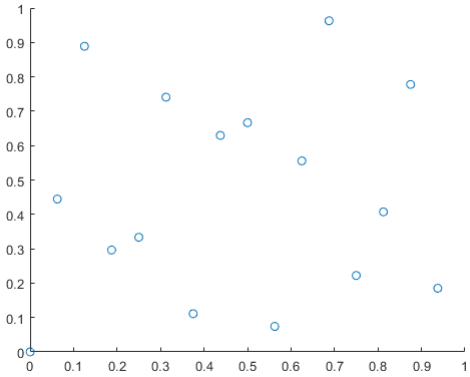


Figure 3.6: Values from the $HaltonSequence(2,3)$ used.

The Figure 3.6 shows the representation of the 16 points used to jitter the projection in the current implementation, as proposed by Fuglsand [16]. It was generated using MATLAB with the command *haltonset(2)* then scrambled using reverse-radix scrambling, *scramble(p, 'RR2')* and, finally, generated the 16 points used with *net(p, 16)*.

3.6.2 Velocity Buffer

The Velocity Buffer algorithm used in this implementation is the one proposed by Chapman [5] which is calculated by subtracting in NDC space the current pixel position by its last frame position. This is possible by saving the MVP matrix of each object in the scene.

Also, as suggested by Xu [19], the jittering is not included as part of the motion.

3.6.3 Frame History Buffer

For each fragment in the current frame, we look for the 3x3 neighborhood and plus $(+)$ pattern neighborhood (See Figure 3.7). We look at both patterns for the minimum and maximum of colors of the current frame, next we average them and use it in the Clipping of the pixels in the History Buffer. [16]

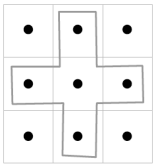


Figure 3.7: Sampling Pattern used. Image taken from Fuglsand presentation. [16]

On the 3×3 neighborhood we look for the velocity of the pixel with the closest depth, this is to get better edges in motion for pixels that are occluded [16]. We use this velocity to reproject the position of the current frame in the History Buffer. [16, 19]

After we have the pixels in the History Buffer, we constrain it (See next subsection) and mix it with the current frame. We linearly mix both using a feedback value that is calculated by the difference of luminance between colors. This feedback is clamped between values closer to one to add some information of the current frame while keeping the pixel's history. This mix stabilizes the image, removing the jittering and smoothing the edges [16, 19]. Because each pixel's history is accumulated in the History Buffer, we get the effect that pixel history from previous frames weights less the more time the pixel's history is not rejected from the History Buffer. [16]

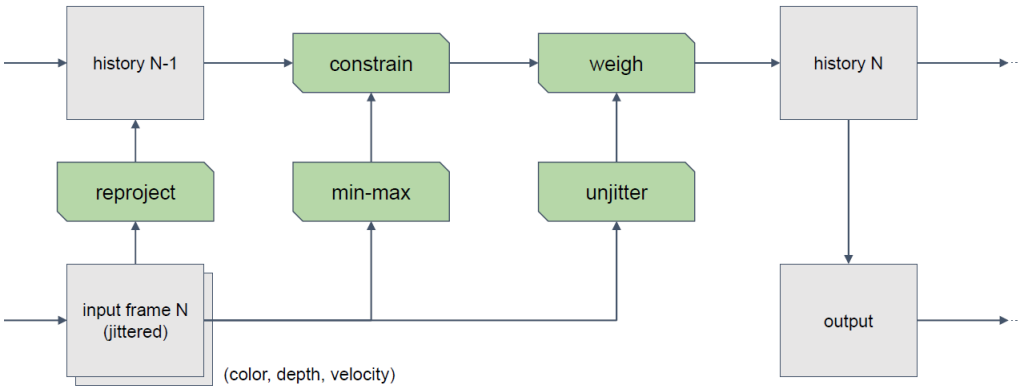


Figure 3.8: Temporal Reprojection Anti-Aliasing process. Image taken from Fuglsand presentation [16]

3.6.4 Clipping Color Box

A Clipping Color Box is used to handle color rejection when the pixel's history in the History Buffer is too distant from current color. This is a box built using the current pixel color as the center and the minimum and maximum color calculated in the last subsection as limits. The history color is taken as a position and projected against the limits of the box if it lies outside, else, it is left untouched. The usage of the Clipping Color Box prevents the color clustering that would happen if Clamp is applied (See Figure 3.9). [16].

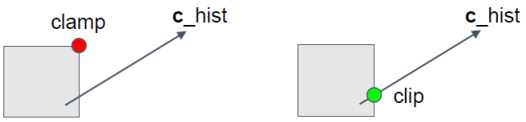


Figure 3.9: Color Clamping versus Color Clipping. Image taken from Fuglsand presentation. [16]

3.6.5 Sharpen Filter

Because the Reprojection process and Color Clipping create blurriness, a Sharpen Filter is required. We used the one proposed by Xu. [19]

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3.1}$$

Equation 3.1 being the Sharpen Filter Convolution Matrix used in Xu presentation. [19]

3.6.6 Motion Blur

Because of the nature of the History Buffer, ghosting is created by fragments from objects that move so fast that they are not rejected as quickly as necessary, under special lighting and background conditions. Fuglsand and Xu [16, 19] proposed to use Motion Blur solutions to hide these artifacts.

The Motion Blur used is the one proposed by Chapman [5]. It tries to behave like a real camera by scaling the velocity of each pixel by the division of the current Frames Per Second (FPS) to the one wanted, thus, simulating the shutter speed. Then it mixes the colors of the pixels that are sampled while following the direction of the velocity buffer vector.

3.6.7 Problems

3.6.7.1 Blurriness

Current implementations of TAA generate a very aggressive blur because of the way they mix the colors of the current frame and the history; the use of areas larger than the pixel increases the errors generated, therefore a Sharpen Filter is required. The filter applied in the implementation is the one used by Xu [19], it solves blurriness reasonably well but it cannot eliminate some artifacts.

3.6.7.2 Ghosting

Some Ghosting is created when objects move, especially under particular lighting and background conditions that make the foreground and background look alike. This is partially corrected with motion blur, nevertheless, some of it remains near objects that move

fast enough to create some Ghosting but slow enough to avoid Motion Blur. Xu proposes the use of Motion Blur and to increase the size of everything using a Stencil technique and manual tagging of objects [19]. Pederson's implementation allows the jitter in the Velocity Buffer calculations to avoid ghosting but it creates some unwanted blurriness [16].

3.7 Accumulation Buffer

The Accumulation Buffer is an anti-aliasing technique that consists, according to Paul Haeberli and Kurt Akeley [7], on rendering the scene several times with camera jittering and then performing a scaled weighted sum of the renderings to generate the current frame.

This is process increases the sampling per pixel and reduces the aliasing effects producing a high-quality image at the cost of rendering everything several times per frame.

3.8 Sobel Operator

It is an efficiently computable 3×3 isotropic gradient operator, as explained by Irwin Sobel. [17]

It works by taking the four-possible simple central gradient estimates in a 3×3 neighborhood and adding them together. The image function is taken as a density/intensity function and the four-possible estimates as orthogonal vectors which are directional derivatives multiplied by a unit vector specifying the derivative's direction. The sum of the four-possible simple central gradient estimates is equivalent to the vector sum of the eight directional derivative vectors.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{3.2}$$

Let 3.2 be the 3×3 neighborhood and $|G|$ the magnitude of the directional derivative estimate of the neighborhood.

The direction of G will be given by the unit vector to the appropriate neighbor. Vector summing causes all the e (center of the 3×3 neighborhood) values to be canceled leaving only the next expression 3.3:

$$\begin{aligned} G &= \frac{c-g}{4} * \begin{bmatrix} 1 & 1 \end{bmatrix} + \frac{a-i}{4} * \begin{bmatrix} -1 & 1 \end{bmatrix} + \frac{b-h}{2} * \begin{bmatrix} 0 & 1 \end{bmatrix} + \frac{f-d}{2} * \begin{bmatrix} 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} \frac{c-g-a+i}{4} + \frac{f-d}{2} & \frac{c-g+a-i}{4} + \frac{b-h}{2} \end{bmatrix} \end{aligned} \tag{3.3}$$

Then we multiply by 4, rather divide by 4, to approximate the value and ensure that we do not lose precision if we perform this with small fixed-point integers. The newly calculated magnitude is sixteen times larger than the original average gradient.

$$G' = 4 * G = \begin{bmatrix} (c-g-a+i) + (f-d) * 2 & (c-g+a-i) * 4 + (b-h) * 2 \end{bmatrix} \tag{3.4}$$

This can be expressed in two weighting matrices. 3.5 for the x component and 3.6 for the y component.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{3.5}$$

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.6}$$

For edge-point detection, what is commonly done is compare the magnitude of G against a numeric threshold to mark points as edges.

3.9 Image Metrics

The process of measuring the quality of an image is a complicated one. As explained by Al-Najjar and Soong [2], we can follow two main methods: subjective or objective. The subjective methods are based on opinions collected from humans and, as one would expect, are considered expensive, difficult to implement and time-consuming to perform. The second kind of methods, the objective ones, are based on mathematical formulas and algorithms to measure the quality of the image without human intervention. For this thesis, we are using objective methods.

Objectives methods can be categorized into three groups, as described by Al-Najjar and Soong [2]:

- **No-Reference:** In which we have no reference image to compare with.
- **Reduced-Reference:** Where we have part of a reference image.
- **Full-Reference:** We have the complete reference image.

For computer graphics, the preferred methods are the Full-Reference ones because reference images can be generated using higher quality but bad performing algorithms to render them. The most common metrics used, and the ones used in this thesis, are: Mean Square Error (MSE); Peak Signal-to-Noise Ratio (PSNR); and the Structural Similarity Index (SSIM).

3.9.1 Mean Square Error (MSE)

Based on the average of the squared error between the pixels of the image and the reference.

$$MSE = \frac{1}{N * M} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (Im(i,j) - Ref(i,j))^2 \tag{3.7}$$

Where N, M are the width and height of the images and Im, Ref are the pixel of the Image and Reference.

3.9.2 Root Mean Square Deviation (RMSD)

It is the standard deviation of MSE. Also called Root Mean Square Error (RMSE).

$$RMSE = \sqrt{MSE} \tag{3.8}$$

3.9.3 Peak Signal-to-Noise Ratio (PSNR)

It is based on the mathematical concept of Signal-To-Noise Ratio (SNR) which measures the signal of the image, which is stored as the colors of the pixels for our purposes, against its error compared to the reference. [2]

$$PSNR = 10 * \log\left(\frac{S^2}{MSE}\right) \tag{3.9}$$

Where S is the maximum value the signal can achieve.In our case it is 255 because we use 8-bit color channels.

3.9.4 Structural Similarity Index (SSIM)

SSIM is a widely used image metric that matches human subjectivity and is highly sensitive to degradations in the spatial structure of image luminance, as explained by Malpica and Bovik. [11]

It requires two images to compare, X and Y, and three similarity functions are performed in a sliding $N \times N$ (typically 11×11) gaussian weighted window.

$$\begin{aligned} l(x,y) &= \frac{2 * \mu_X(x,y) * \mu_Y(x,y) + C_1}{\mu_X^2(x,y) + \mu_Y^2(x,y) + C_1} \\ c(x,y) &= \frac{2 * \sigma_X(x,y) * \sigma_Y(x,y) + C_2}{\sigma_X^2(x,y) + \sigma_Y^2(x,y) + C_2} \\ s(x,y) &= \frac{\sigma_{XY}(x,y) + C_3}{\sigma_X(x,y) + \sigma_Y(x,y) + C_3} \end{aligned} \tag{3.10}$$

Where

$$\mu_X(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) * X(x+p, y+q)$$

$$\sigma_X^2(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) * [X(x+p, y+q) - \mu_X(x,y)]^2$$

$$\sigma_{XY}(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) * [X(x+p, y+q) - \mu_X(x,y)] * [Y(x+p, y+q) - \mu_Y(x,y)]$$

Where $w(p,q)$ is a Gaussian weighing function such that $\sum_{p=-P}^{P}\sum_{q=-Q}^{Q} w(p,q) = 1$ and C_1, C_2, C_3 are small constants that provide stability when the denominator approaches zero. Typically, they are set as follows:

$$C_1 = (K_1 * L)^2, C_2 = (K_1 * L)^2, C_3 = \frac{C_2}{2}$$

Where L is the dynamic range of the image and $K_1, K_2 \ll 1$ are small constants. At the end, the three similarity functions are combined in the general form:

$$SSIM(x,y) = l(x,y) * c(x,y) * s(x,y) \tag{3.11}$$

Chapter 4
Development

In this chapter, the main work performed in this master thesis is presented.

4.1 EDAN35 Project Improvements

For this master thesis, we decided to use our project from High Performance Computer Graphics (EDAN35) as the base. During the course, we implemented a Temporal Reprojection Anti-Aliasing technique based on the presentations of Ke Xu and Lasse Fuglsang [19, 16], from the games Inside and Uncharted 4. It proved to be reliable and well documented; and allowed us to put into practice the fundamentals of the technique in an academic environment, providing the base for improvements performed for this master thesis.

The EDAN35 project implementation had errors in the jittering process that were corrected by properly expanding what both implementations meant by camera jittering , see Appendix B for a full explanation. The management of the Halton points was redone to accomplish the improved camera jittering with the inclusion of support of up to 128 points to work as the jittering of the Accumulation Buffer (See Figure 4.1). Note, though, for Temporal Anti-Aliasing only the first 16 points are used as suggested by Xu and Fuglsang. [19, 16]

Specular Lighting Anti-Aliasing is a complex problem by itself, which requires specialized solutions that work directly with light reflections. Anti-Aliasing techniques do not correct it by themselves, they usually work with suitable already made solutions. To avoid problems with Specular Lighting, it was decided to be turned off.

Also, there were added models for a sphere, wall, pipe, hairball, a window with blinds and an arched window to test the improvements done to the Temporal Anti-Aliasing. All models, except the wall, were added with one color-solid texture to avoid introducing lighting errors in the calculations of the image metrics for the comparisons between the Uncharted 4 implementation and the one developed in this master thesis. The wall model uses a white texture with black letters on it, this is because letters use hard edges to define

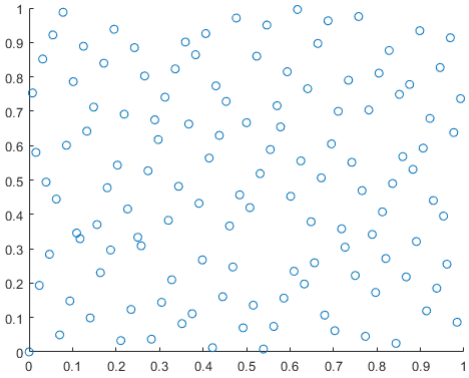


Figure 4.1: The 128 $HaltonSequence(2, 3)$ points available to use.

their shape and must stay that way after applying any Anti-Aliasing technique.

4.1.1 Fast Approximate Anti-Aliasing (FXAA)

For this master thesis, the white paper version of Fast Approximate Anti-Aliasing (FXAA) by Lottes [10] was used to compare it against Temporal Anti-Aliasing, on the ground that both techniques are Post-Processing Anti-Aliasing and that FXAA is a popular technique used in the industry. It was implemented under the highest quality preset according to the white paper without taking into consideration the performance impact because we wanted to compare the raw improvement that both techniques can provide.

4.1.2 Enhanced Subpixel Morphological Antialiasing (SMAA)

In order to test a more complex and newer post-processing technique, SMAA was implemented following the instructions provided by Jimenez et al. [8]. It was implemented using the highest preset that works with the deferred shading pipeline.

4.1.3 Accumulation Buffer

We use an Accumulation Buffer to provide a reference image of what the scene is. It was implemented following Haeberli and Akeley [7]. The points used to jitter the camera are Halton Sequence Points as for Temporal Anti-Aliasing, although the Accumulation Buffer can use up to 128. The reasons for using Halton Sequence are that: it fulfils the requirements established by Haeberli and Akeley; it is easy to extend the current camera system to support more Halton Points; and for Temporal Anti-Aliasing, it provides pseudo-random points that do not follow a pattern to help gather as much information of the scene as possible.

4.1.3.1 Sample Number Selection

The number of samples selected for the Accumulation Buffer is 128. This is because it provides the best representation possible of the scene even though it causes a substantial loss of performance.

In order to show the difference between using 16 and 128 samples, we performed four tests to observe how the metrics behave under different arrangements of scene objects and lighting. In the next table we see the results of one such test:

Table 4.1: Metrics behavior comparison between using 16 samples versus 128 for Accumulation Buffer.

Test D				
Samples / **Tests**	**16**	**128**	**Difference**	**Relative Difference** $(\%)$
MSE of Temporal	40.647863	38.947297	-1.700566	4.183654132%
RMSD of Temporal	6.375568	6.240777	-0.134791	2.114180258%
MSE of No AA	24.872104	24.524992	-0.347112	1.395587603%
RMSD of No AA	4.987194	4.952271	-0.034923	0.700253489%
Peak-SNR of Temporal	32.040426	32.22603	0.185604	0.575944353%
SNR of Temporal	30.30596	30.492374	0.186414	0.611346299%
Peak-SNR of No AA	34.173678	34.234715	0.061037	0.178289786%
SNR of No AA	32.439212	32.501058	0.061846	0.190289190%
SSIM of Temporal	0.993302	0.993451	0.000149	0.014998223%
SSIM of No AA	0.996826	0.996891	$6.5E-05$	0.006520272%

The changes are large enough on some metrics to be noticeable, especially on MSE and SSIM of Temporal Anti-Aliasing. We are going to notice the effects of the use of 128 samples when we reach the comparisons between Anti-Aliasing techniques.

4.2 Testing Framework Implementation

To measure any improvement achieved, we developed a testing framework that allows us to save the important information when the test is performed. The framework allows us to select which technique to use as the main renderer: the Master Thesis Temporal Anti-Aliasing implementation; Uncharted 4 Temporal Anti-aliasing implementation; Enhanced Subpixel Morphological Antialiasing (SMAA); or Fast Approximate Anti-Aliasing (FXAA). It also allows us to zoom any part of the screen and then perform all the image metrics calculations using MATLAB.

When a test is performed, selected rendered images are saved as PNGs with 4 color channels and no compression. Also, basic information regarding the date when the test was performed; the camera information and data regarding the values used for Temporal Anti-Aliasing are saved in a plain text file. To quantify if improvements were achieved on the main objectives of this thesis, ghosting, and blurriness, two different types of tests were developed: Static Test and Ghosting Test.

4.2.1 Static Test

This type of test consists of letting the History Buffer fill for 16 frames by rendering them with the Temporal Anti-Aliasing technique selected and then saving the last frame rendered. Immediately rendering the last frame using the Accumulation Buffer, to generate the ground truth image of the scene, and SMAA, FXAA and No Anti-Aliasing (No AA), for comparison purposes.

4.2.2 Ghosting Test

This type of test is performed only with the sphere and hairball models; the first is moving through the alley of the scene simulating a moving object in an application and the second one is rotating in a static position simulating many edges moving. The test consists of rendering the scene for a selected number of frames, with the Master Thesis Temporal Anti-Aliasing implementation and the Uncharted 4 Temporal Anti-Aliasing implementation at the same time. After each frame is rendered, each image, the position of the sphere and the rotation of the hairball are saved.

Once the selected number of frames has run through, the sphere is returned to its original position and the movement is repeated using the positions saved before; and the hairball is returned to its original rotation and the movement is also repeated. The difference this time is that every frame is rendered using the Accumulation Buffer and then saved, it is done to avoid the heavy performance loss caused by the Accumulation Buffer impacting the Temporal Anti-Aliasing techniques.

4.2.3 MATLAB Image Metrics

Once all the test results are saved, one script takes all the images and organizes them into folders by name and type of test. Afterward, all the test results are processed using MATLAB.

For Static Tests, we perform MSE, RMSD, PSNR, SNR and SSIM measurements to the TAA, FXAA, SMAA and No AA rendered images, using the Accumulation Buffer rendered images as the reference to compare with. Also, we generate a local SSIM map for each rendered image, apart from the one rendered with the Accumulation Buffer, which are saved as PNGs and MATLAB FIGs. All the measurements results are stored in the folder with the test results as plain text.

For Ghosting Tests, we perform MSE, RMSD, PSNR, SNR and SSIM measurements to each frame rendered with the Temporal Anti-Aliasing of Uncharted 4 technique and the Temporal Anti-Aliasing of the Master Thesis using the corresponding Accumulation Buffer rendered frame. A local SSIM map is generated for each rendered image, except the Accumulation Buffer ones, and saved as PNGs and MATLAB FIGs. All the results of all images are stored in a plain text file. In the SSIM maps, dissimilarity is represented as colors and white as similarity.

4.3 Temporal Reprojection Anti-Aliasing Modifications

For this thesis, the Color Clipping Box technique was modified to be affected by the values calculated from the new techniques applied in this thesis. These changes follow the rationale that we want to apply the full force of the Temporal Anti-Aliasing technique when needed, and not elsewhere, to minimize the effects of ghosting and blurring.

The first change consists in that the Colors that were calculated from the average between the 3×3 and Cross neighborhoods by the sampling patterns are now mixed in a variable amount. This follows the idea that we prefer the Cross neighborhood over the 3×3 neighborhood if the pixel we are currently calculating is not considered to be aliased, this is because the Cross neighborhood is less likely to introduce unwanted information in the minimum, maximum and average calculations. But, if the pixel is considered to be aliased, we prefer the 3×3 neighborhood because it provides more information about the surroundings of the pixel to create the unaliased image.

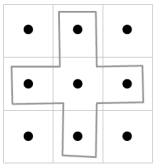


Figure 4.2: Sampling Pattern used. Image taken from Fuglsand presentation. [16]

The second change is that the size of the Color Clipping Box depends on how much the pixel is considered to be aliased. By using smaller Color Clipping boxes than the original technique on unaliased pixels, we increase the elimination of unwanted colors from the history, reducing the effects of ghosting since we are rejecting colors faster on pixels that we know are not considered to be aliased. This is implemented by linearly interpolating the current pixel color and the minimum and maximum colors calculated for the Color Clipping Box.

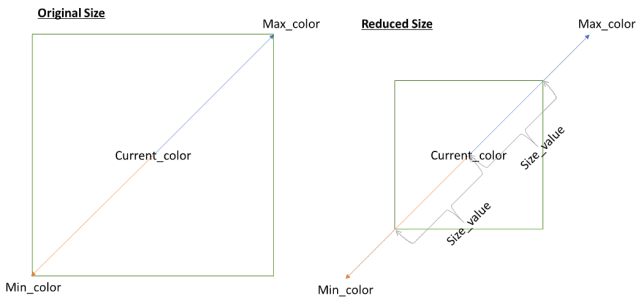


Figure 4.3: Color Clipping Box size reduction

4.3.1 Triangle Indexing Improvements Implementation

The main idea behind the application of this technique is to detect pixels that we considered aliasing by using the number of different models the pixel is surrounded with. We want to detect the edges between different models because aliasing normally occurs there. Once we have this information, we proceed to alter the Color Clipping Box that controls the application of TAA.

To implement this technique, all models in the scene receive a unique integer index. Then, in the renderization geometry pass, all triangles belonging to the same model receive the model index as its ID. Subsequently, it is used in the pixel shader to generate a texture in which every pixel contains the ID of the model it belongs to.

On the Temporal Reprojection pass, the average on the number of pixels belonging to different models in the 3×3 neighborhood of the current pixel is calculated. With this average, we proceed to skew the color linear interpolation between the minimum, maximum and average between the Cross and 3×3 neighborhoods of the pixel and we proceed to change the size of the Color Clipping Box.

If the average is close to zero, meaning that the pixel is surrounded by pixels of its same model, we interpolate towards colors from the Cross neighborhood and reduce the size of the Color Clipping Box. But, if the average is close to one, meaning that the pixel is surrounded by many pixels of other models, we interpolate towards colors from the 3×3 neighborhood and let the Color Clipping Box stay on its original size.

$$modelAverage_i = \frac{\sum_{j=1}^{9} ModelDiff(i,j)}{9} \tag{4.1}$$

Where

$$ModelDiff(i,j) = \begin{cases} 1 & if \quad ModelID_i \neq ModelID_j \\ 0 & else \end{cases}$$

4.3.2 Depth Pseudo-Variance and Depth Temporal Pseudo-Variance

The key point behind this technique is that we want to detect pixels that we consider aliased because they are in a neighborhood of pixels separated by relatively long depth distances or that they change their depth relatively a long distance through frames.

First, we calculate the minimum, maximum and average linear depth from the 3×3 neighborhood. Then we proceed to use the next formula to calculate the value we are going to use to normalize results:

$$\begin{aligned} maxDepthDistance = min\,(& |depthMin - depthAvg|\,, \\ & |depthMax - depthAvg|) \end{aligned} \tag{4.2}$$

We calculate the normalizing value using the minimum to avoid the interference of outliers. If $maxDepthDistance$ is below 0.002 everything that follows is set to zero

because pixels are so close that it's probably not aliased. This threshold was defined by experimenting which values would not get noise inside the calculations but would let the interesting pixels in.

Then we calculate the depth pseudo variance as:

$$depthPseudoVariance = \left(\frac{|currentDepth - depthAvg|}{maxDepthDistance} \right)^2 \tag{4.3}$$

This provides us with a pseudo-variance that measures the distance between the average depth and the current pixel depth. Note that normally the value is going to be between 0 and 1 and but, if the pixel depth is an outlier, this value is going to be over 1.

Finally, we calculate how the depth of the pixel in the last frame relates to the current neighborhood, using the depth temporal pseudo variance. Note that we use the fourth power to reduce the noise in the calculations.

$$depthTemporalPseudoVariance = \left(\frac{|pastDepth - depthAvg|}{maxDepthDistance} \right)^4 \tag{4.4}$$

4.3.3 Sobel Improvements Implementation

The main idea behind applying Sobel edge detection technique is to concentrate the TAA effects on pixels on edges to correct aliasing, if necessary.

We apply the Sobel Operator to the luminance of the light scene produced by the Deferred Shading Pipeline, the luminance of the unlighted scene from the Geometric Buffer, and the current linear depth. We use luminance because the human eye is better at recognizing sudden changes in it and we use the light and unlight scene to avoid problems detecting edges because of lights or shadows.

Each Sobel Operator is performed separately, and their magnitudes mixed at the end as follows:

$$g = (u * 0.3 + l * 0.7) + d \tag{4.5}$$

Where u is the magnitude of the sobel operator of the luminance of the unlight scene, l is the magnitude of the sobel operator of the luminance of the lighted scene and d is the magnitude of the sobel operator of the current linear depth.

Then, we clamp g between 0.0 and 1.0 to finally apply the smoothstep polynomial as follows:

$$sobel = \sqrt{g^2 * (3.0 - 2.0 * g)} \tag{4.6}$$

After all the Sobel Operators are applied, we save the results to a texture and perform a simplified version of TRAA to keep the results stable through time. This simplified version is almost the same as the one we use as a base of this master thesis, the differences come from the use of Clamping rather than a Clipping Box because the texture values are one dimensional; and we do not apply a sharpen filter.

The output of this TRAA is used to calculate the Sobel value of the current pixel, the average Sobel value in the Cross Neighborhood and the average Sobel value in the 3×3 Neighborhood.

4.3.4 Final Mixing

Finally, we use the values previously calculated to modify how the Clipping Box is calculated.

First , we define the mix function as follows:

$$Mix(x, y, t) = x * (1 - t) + y * t \quad with\ 0 \leq t \leq 1 \tag{4.7}$$

Then, the final mixing is applied as follows:

$$sobelAvgMixVal = Clamp01(modelAverage_i + sobel) \tag{4.8}$$

Where $sobel$ is the Sobel value of the current pixel, Equation 4.6, $modelAverage_i$ comes from the Equation 4.1 and $Clamp01$ is the clamping function between 0 and 1.

$$sobelAvg = Mix(sobelAvgCross, sobelAvg3x3, sobelAvgMixVal) \tag{4.9}$$

Where $sobelAvgCross$ is the average Sobel value of the Cross Neighborhood and $sobelAvg3x3$ is the average Sobel value of the 3×3 Neighborhood around the current pixel.

And, we use Equations 4.9, 4.3, 4.4 and 4.1 to calculate how much this pixel is considered to be aliased:

$$\begin{aligned} aliasedValue = Clamp01(&sobelAvg + depthPseudoVariance \\ &+ depthTemporalPseudoVariance \\ &+ modelAverage_i) \end{aligned} \tag{4.10}$$

With this $aliasedValue$ we proceed to modify the Color Clipping Box. First, we change how the colors are calculated:

$$\begin{aligned} colorMin &= Mix(colorMinCross, colorMin3x3, aliasedValue) \\ colorMax &= Mix(colorMaxCross, colorMax3x3, aliasedValue) \\ colorAvg &= Mix(colorAvgCross, colorAvg3x3, aliasedValue) \end{aligned} \tag{4.11}$$

Then we modify the size of the box:

$$\begin{aligned} clipColoorMin &= Mix(colorCurrent, colorMin, aliasedValue) \\ clipColorMax &= Mix(colorCurrent, colorMax, aliasedValue) \end{aligned} \tag{4.12}$$

The Figures 4.4, 4.5 and 4.6 are examples of the aliased values calculated. The white color represents a value of 1 and the black color of 0.

Figure 4.4: Image made of the Aliased Values of each pixel.

Figure 4.5: Image made of the Aliased Values of each pixel.

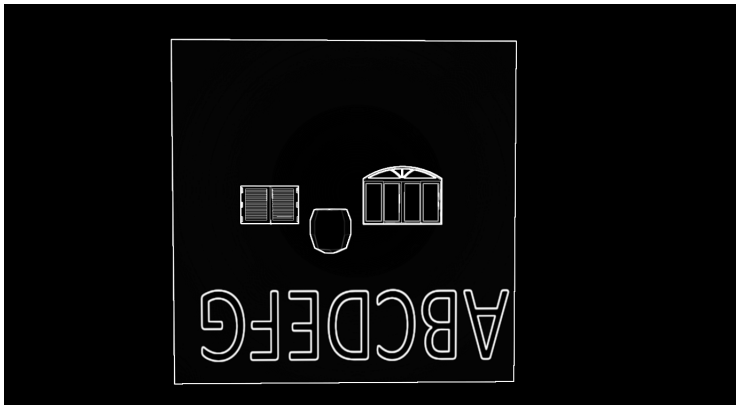


Figure 4.6: Image made of the Aliased Values of each pixel.

4.3.5 Sharpen Filter Modifications

The Sharpen Filter was normalized to avoid creating a color peak. It was changed from 3.1 to:

$$\begin{bmatrix} 0 & -0.25 & 0 \\ -0.25 & 2 & -0.25 \\ 0 & -0.25 & 0 \end{bmatrix} \tag{4.13}$$

Equation 4.13 shows the new Sharpen Filter Convolution Matrix used.

Chapter 5
Results

In this chapter, we explain how we evaluated the improvements done to the Temporal Anti-Aliasing. We show the numerical and visual results obtained. Finally, we explain the results and their meaning compared to the previous implementation of TAA and other Anti-Aliasing solutions.

5.1 Evaluation Methodology

To evaluate the improvements achieved to the Temporal Anti-Aliasing technique we selected models and camera angles that place the technique under stress. Then, using the Testing Framework we developed, we proceed to render and save images of each of those models to compare how the technique behaves in comparison to the original TAA implementation and other Anti-Aliasing techniques to determine if the proposed techniques in this thesis provided images with better quality without incurring heavy memory usage or time consumption. Also, a special test was taken to measure if changing the Sharpen Filter was the sole mechanism providing an improvement.

The models used on the tests were:

- Pipe: A brown pipe with hard edges.
- Window with Blinds: A blue window with closed blinds.
- Arched Window: A blue window with an arch at the top.
- Wall: A white wall with black text.
- Sponza Atrium: exemplifies a general scene.
- Sponza Atrium Flowers: exemplifies a cutout model.
- Hairball: contains many fine details for its numerous fibers.

Note that Sponza Atrium and Hairball models were downloaded from Morgan McGuire's Computer Graphics Archive [12]; the Pipe model is from Spencer Arts [4]; the Arched Window model is from Isabela H. [6]; and the Window with Blinds is from Channa Yim [21].

The tests were performed on the computer provided by Lund University which has the following specification:

- CPU: Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz, 3601 Mhz, 4 Cores, 8 Logical Processors
- RAM: 64.0 GB
- GPU: NVIDIA GeForce GTX 1080 with 8 GB of VRAM
- Rendering Resolution when not zooming: 1600 x 900

5.2 Results and Comparisons

5.2.1 Sharpen Filter

For this test, we rendered the Sponza Atrium Model and a Static Sphere Model to evaluate the effects of the Normalized Sharpen Filter to the general quality of the rendered image, especially, regarding the blurring normally caused by TAA. Everything was rendered using both implementations of TAA with and without the Normalized Sharpen Filter to see if this change was the only improvement that was increasing the quality of the rendered image. The scene was selected for the test because it provided an example of a general scene that should not generate any blur. As we see from the Table 5.1, even if the Master Thesis TAA and the Uncharted TAA used the new Sharpen Filter, our other improvements still rendered a higher quality image.

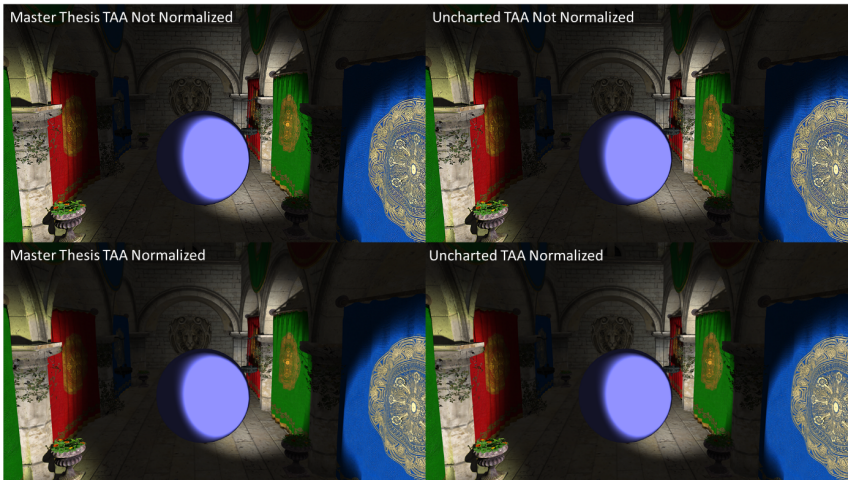


Figure 5.1: Rendered Images comparison.

Table 5.1: Sharpen Filter Test numerical results

Sharpen Filter Test						
AA / Tests	**Uncharted TAA Not Normalized**	**Uncharted TAA Normalized**	**Master TAA Not Normalized**	**Master TAA Normalized**	**Best**	**Master TAA Normalized Against Best**
MSE	149.271	8.835	148.036	8.224	Master TAA Normalized	0.000
RMSD	12.218	2.972	12.167	2.868	Master TAA Normalized	0.000
Peak-SNR	26.391	38.669	26.427	38.980	Master TAA Normalized	0.000
SNR	16.245	28.522	16.281	28.833	Master TAA Normalized	0.000
SSIM	0.932	0.992	0.933	0.992	Master TAA Normalized	0.000

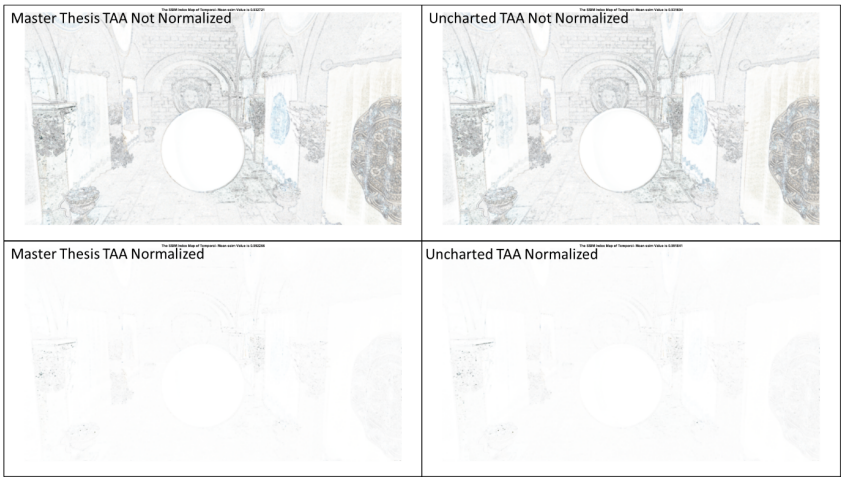


Figure 5.2: SSIM Maps comparison.

5.2.2 Pipe

For this test, we rendered the Pipe Model in order to test how the improvements behaved when rendering a model with hard edges. We wanted to test if our improvements reduced the amount of blur around those hard edges while still fixing the aliasing problem. We rendered the Pipe twice, the first time using a regular camera angle, for normal aliasing around the edges, and a skewed camera angle, for increased aliasing effects. As we see from the results, TAA with our improvements is at the same quality level as SMAA.

5.2.2.1 Regular

Table 5.2: Numerical results of the Pipe Test with regular camera inclination.

Pipe Regular Test							
AA / Tests	**No AA**	**FXAA**	**SMAA**	**Uncharted TAA**	**Master TAA**	**Best**	**Master TAA Against Best**
MSE	8.608	3.573	1.278	14.602	1.574	SMAA	-0.296
RMSD	2.934	1.890	1.130	3.821	1.254	SMAA	-0.124
Peak-SNR	38.782	42.601	47.066	36.487	46.162	SMAA	0.904
SNR	36.451	40.270	44.735	34.156	43.831	SMAA	0.904
SSIM	0.999	0.999	1.000	0.996	1.000	SMAA	0.000

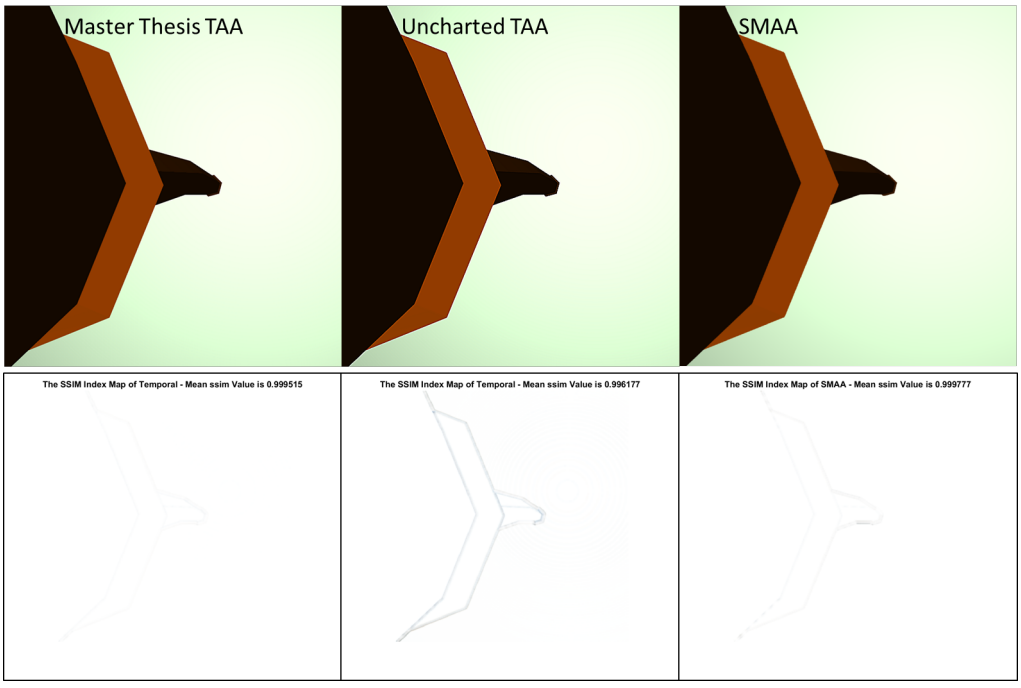


Figure 5.3: Pipe Regular comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

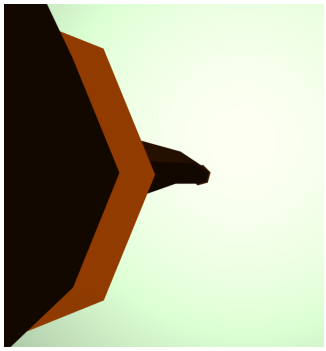


Figure 5.4: Pipe Regular Test ground truth.

5.2.2.2 With Camera Inclination

Table 5.3: Numerical results of the Pipe Test with a skewed camera inclination.

Pipe with Camera Inclination Test							
Tests \ **AA**	**No AA**	**FXAA**	**SMAA**	**Uncharted TAA**	**Master TAA**	**Best**	**Master TAA Against Best**
MSE	16.112	6.470	2.810	14.349	2.664	Master TAA	0.000
RMSD	4.014	2.544	1.676	3.788	1.632	Master TAA	0.000
Peak-SNR	36.059	40.022	43.644	36.563	43.876	Master TAA	0.000
SNR	32.474	36.437	40.059	32.978	40.291	Master TAA	0.000
SSIM	0.998	0.999	1.000	0.996	0.999	SMAA	0.000

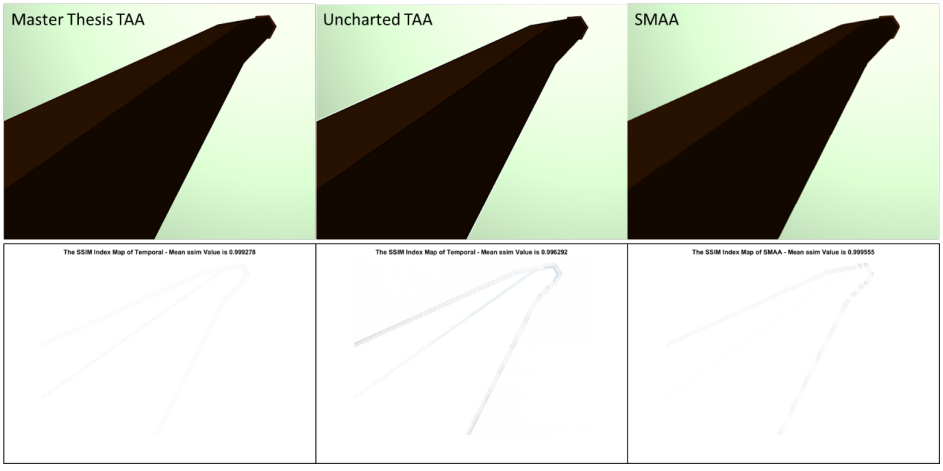


Figure 5.5: Pipe with Camera Inclination comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

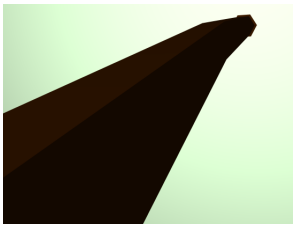


Figure 5.6: Pipe with Camera Inclination Test ground truth.

5.2.3 Window with Blinds

On this test, we used the Window with Blinds model for its small details at the blinds. We tested how our improvements behaved with this kind of details and discovered that it did not react in a proper way even if the numerical results showed otherwise.

Table 5.4: Numerical results of the Window with Blinds Test.

Window with Blinds Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	96.044	70.486	35.134	170.229	32.115	Master TAA	0.000
RMSD	9.800	8.396	5.927	13.047	5.667	Master TAA	0.000
Peak-SNR	28.306	29.650	32.674	25.820	33.064	Master TAA	0.000
SNR	25.467	26.810	29.834	22.981	30.224	Master TAA	0.000
SSIM	0.986	0.990	0.995	0.976	0.995	SMAA	0.000

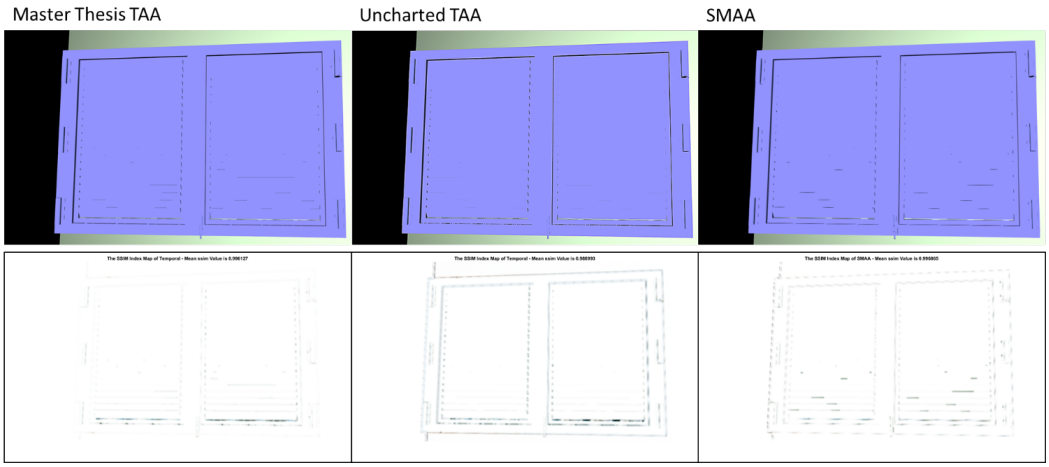


Figure 5.7: Window with Blinds comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

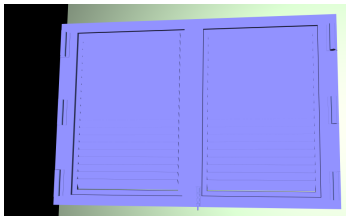


Figure 5.8: Window with Blinds ground truth.

5.2.4 Arched Window

We used the Arched Window Model to test how our improved implementation with the small details of the window's door and the aliasing from the arch. The results surprised us, as the best visual quality was achieved by the Uncharted TAA while the numerical test showed SMAA as the best one.

Table 5.5: Numerical results of the Arched Window Test.

Arched Window Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	56.313	39.103	19.849	76.483	21.983	SMAA	-2.134
RMSD	7.504	6.253	4.455	8.745	4.689	SMAA	-0.233
Peak-SNR	30.625	32.209	35.153	29.295	34.710	SMAA	0.443
SNR	27.325	28.909	31.854	25.996	31.410	SMAA	0.443
SSIM	0.992	0.994	0.997	0.989	0.996	SMAA	0.001

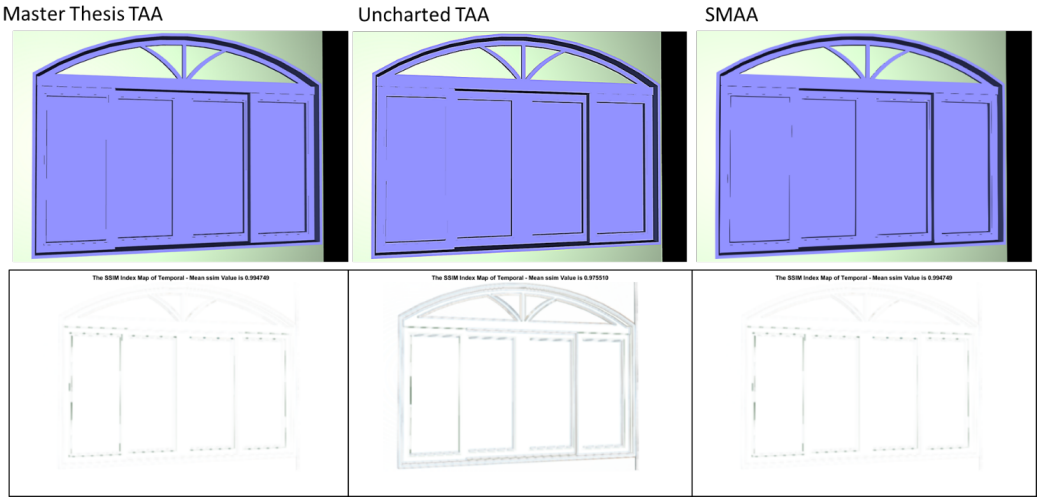


Figure 5.9: Arched Window comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

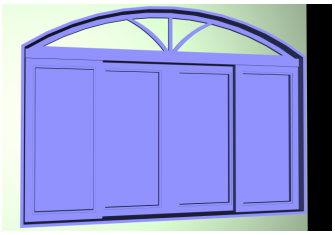


Figure 5.10: Arched Window ground truth.

5.2.5 Sponza Atrium

For this test, we wanted to analyze how the Master TAA implementation behaved with a general scene with lights and shadows, we used the Sponza Atrium Model with the Sphere Model being static in the center. The visual and numerical results showed us how that our technique properly handles this type of general scene.

Table 5.6: Numerical results of the Sponza Atrium Test.

Sponza Atrium Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	13.458	8.290	8.610	42.728	3.972	Master TAA	0.000
RMSD	3.669	2.879	2.934	6.537	1.993	Master TAA	0.000
Peak-SNR	36.841	38.945	38.781	31.824	42.141	Master TAA	0.000
SNR	20.056	22.160	21.996	15.038	25.356	Master TAA	0.000
SSIM	0.988	0.991	0.991	0.938	0.991	FXAA	0.000

Figure 5.11: Sponza Atrium comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

Figure 5.12: Sponza Atrium ground truth.

5.2.6 Sponza Atrium Flowers

On this test, we looked at how do our improvements handle the details of a model with transparent parts, as in the Flowers from the Sponza Atrium Model, because the aliasing they present is considered hard to properly detect and correct, as our results show.

Table 5.7: Numerical results of the Sponza Atrium Flowers Test.

Sponza Atrium Flowers Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	122.795	66.062	72.279	490.281	36.162	Master TAA	0.000
RMSD	11.081	8.128	8.502	22.142	6.013	Master TAA	0.000
Peak-SNR	27.239	29.931	29.541	21.226	32.548	Master TAA	0.000
SNR	19.590	22.282	21.891	13.577	24.899	Master TAA	0.000
SSIM	0.959	0.975	0.972	0.863	0.985	Master TAA	0.000

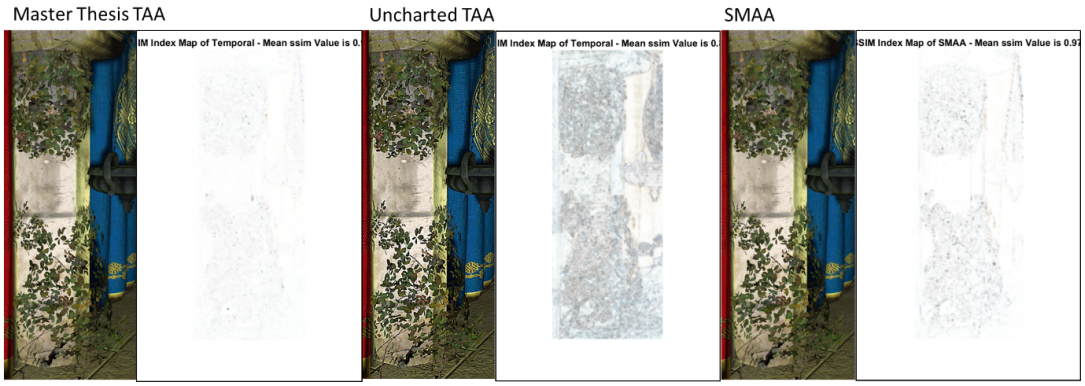


Figure 5.13: Sponza Atrium Flowers comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

Figure 5.14: Sponza Atrium Flowers ground truth.

5.2.7 Hard Edges

For this test, we explored how the Master Thesis Implementation with many small details at a far distance and how it behaved with hard edges. We used both Windows Models for the small details and the Pipe and Wall Models for the hard edges. The result showed us that, both visually and numerically, our improvements achieved better quality.

Table 5.8: Numerical results of the Hard Edges Test.

Hard Edges Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	12.463	9.342	8.019	31.385	5.012	Master TAA	0.000
RMSD	3.530	3.057	2.832	5.602	2.239	Master TAA	0.000
Peak-SNR	37.174	38.426	39.090	33.164	41.131	Master TAA	0.000
SNR	30.327	31.579	32.242	26.316	34.283	Master TAA	0.000
SSIM	0.997	0.997	0.998	0.989	0.998	Master TAA	0.000

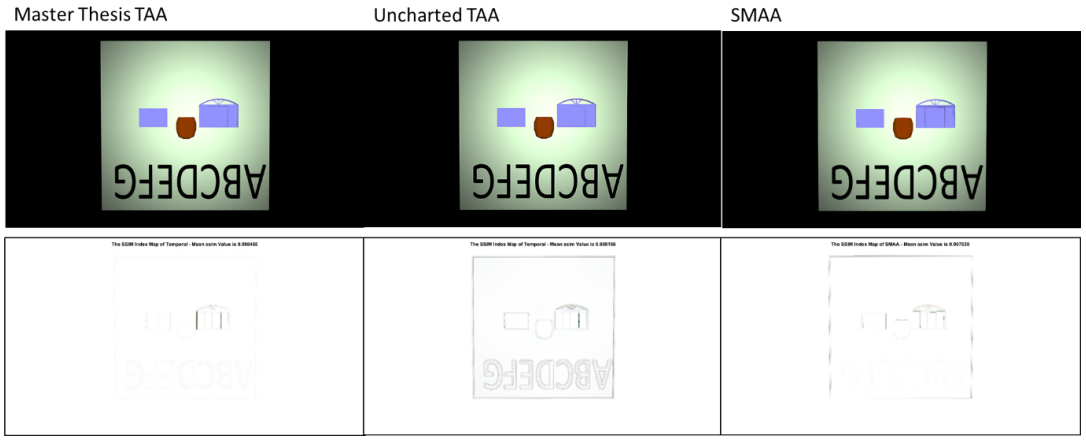


Figure 5.15: Hard Edges comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

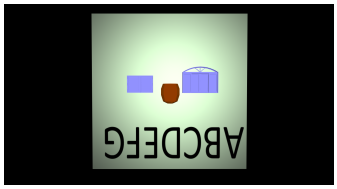


Figure 5.16: Hard Edges ground truth.

5.2.8 Sphere Ghosting

For this test, we wanted to measure how the improvements we performed decreased the effects of ghosting that were present in the Uncharted Implementation. For this reason, we selected the Sphere Model to move in the hall of the Sponza Atrium and rendered it under a camera angle that showed us ghosting trails. As we see from the visual results, ghosting effects were diminished in our implementation.

Table 5.9: Numerical results summary of the 100 tests performed for the Sphere Ghosting Test.

Sphere Ghosting Test Summary				
AA / Tests	Uncharted TAA	Test Index	Master TAA	Test Index
Best MSE	0.000	63.000	0.000	63.000
Worst MSE	100.871	19.000	91.766	29.000
Average MSE	28.634	N/A	19.501	N/A
Best Peak-SNR	Inf	63.000	Inf	63.000
Worst Peak-SNR	28.093	19.000	28.504	29.000
Average Peak-SNR	Inf	N/A	Inf	N/A
Best SNR	Inf	63.000	Inf	63.000
Worst SNR	16.818	11.000	18.524	29.000
Average SNR	Inf	N/A	Inf	N/A
Best SSIM	1.000	63.000	1.000	63.000
Worst SSIM	0.964	99.000	0.972	29.000
Average SSIM	0.965	N/A	0.990	N/A

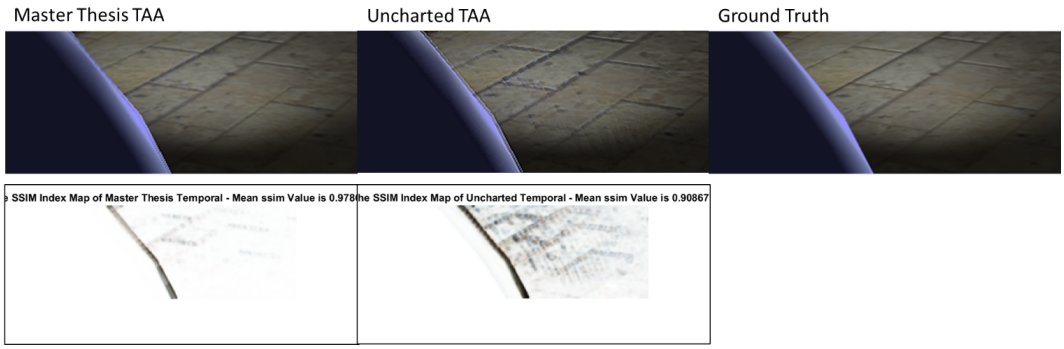


Figure 5.17: Example of Ghosting. Comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 19.

5.2.9 Hairball

These tests were the hardest for all the techniques we tried. The Hairball Model contains many fibers with many details that provide a complex test for any Anti-Aliasing solution. We performed all the tests without light, to see the silhouette, and with light, to see how its reaction to the fibers affected all the Anti-Aliasing solutions. We did two sets of tests, the first one was static, to show us the behavior of blurring and aliasing correction; and the second set was rotating, to show us how ghosting behaved on the fibers. The results from the static tests were surprising, as we did not expect the Master Thesis Implementation to be the best handling the fibers.

5.2.9.1 Static Shadow

Table 5.10: Numerical results of the Hairball Static Shadow Test.

Hairball Static Shadow Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	44.367	21.101	25.379	88.976	10.293	Master TAA	0.000
RMSD	6.661	4.594	5.038	9.433	3.208	Master TAA	0.000
Peak-SNR	31.660	34.888	34.086	28.638	38.005	Master TAA	0.000
SNR	18.808	22.036	21.234	15.786	25.154	Master TAA	0.000
SSIM	0.962	0.978	0.974	0.934	0.985	Master TAA	0.000

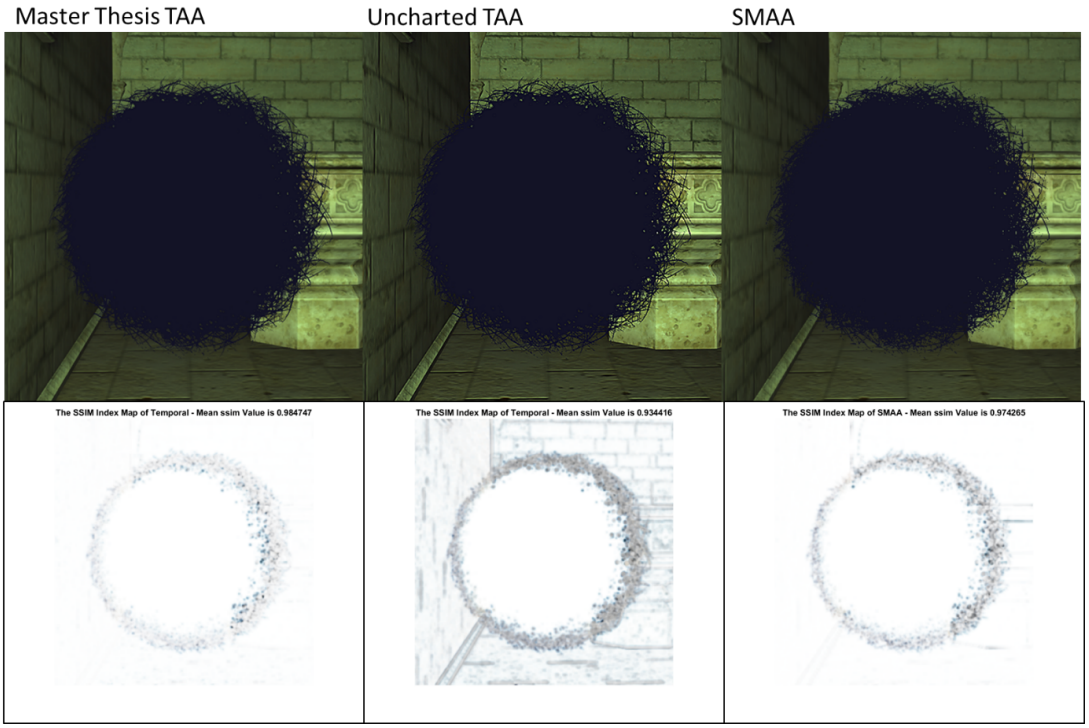


Figure 5.18: Hairball Static Shadow comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

Figure 5.19: Hairball Static Shadow ground truth.

5.2.9.2 Static Light

Table 5.11: Numerical results of the Hairball Static Light Test.

Hairball Static Light Test							
AA / Tests	No AA	FXAA	SMAA	Uncharted TAA	Master TAA	Best	Master TAA Against Best
MSE	1294.649	649.940	847.702	1444.095	226.567	Master TAA	0.000
RMSD	35.981	25.494	29.115	38.001	15.052	Master TAA	0.000
Peak-SNR	17.009	20.002	18.848	16.535	24.579	Master TAA	0.000
SNR	8.446	11.439	10.285	7.971	16.015	Master TAA	0.000
SSIM	0.801	0.865	0.841	0.785	0.937	Master TAA	0.000

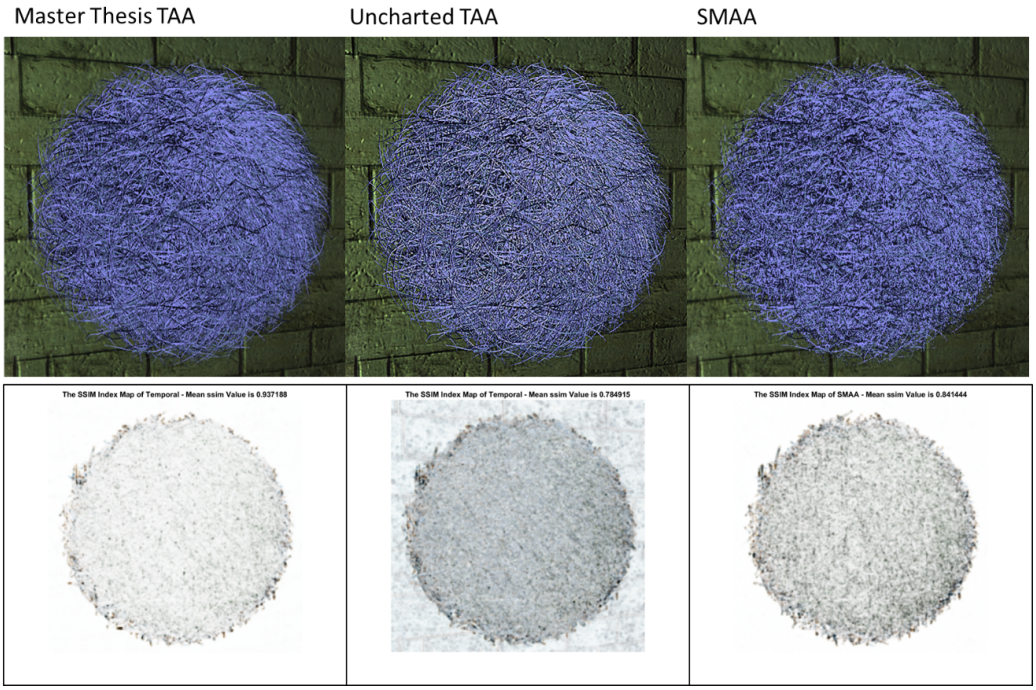


Figure 5.20: Hairball Static Lighted comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

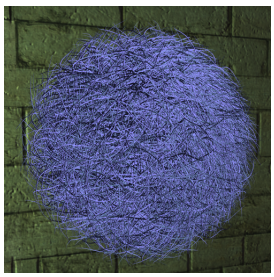


Figure 5.21: Hairball Static Lighted ground truth.

5.2.9.3 Ghosting Shadow

Table 5.12: Numerical results summary of the 100 tests performed for the Hairball Ghosting Shadow Test.

Hairball Ghosting Shadow Test Summary				
AA / Tests	Uncharted TAA	Test Index	Master TAA	Test Index
Best MSE	70.261	17	32.692	0
Worst MSE	93.024	90	42.962	90
Average MSE	81.887	N/A	38.534	N/A
Best Peak-SNR	29.664	17	32.986	0
Worst Peak-SNR	28.445	90	31.800	90
Average Peak-SNR	29.009	N/A	32.278	N/A
Best SNR	16.940	17	20.278	0
Worst SNR	15.846	90	19.201	90
Average SNR	16.333	N/A	19.602	N/A
Best SSIM	0.925	17	0.947	17
Worst SSIM	0.911	99	0.934	68
Average SSIM	0.913	N/A	0.940	N/A

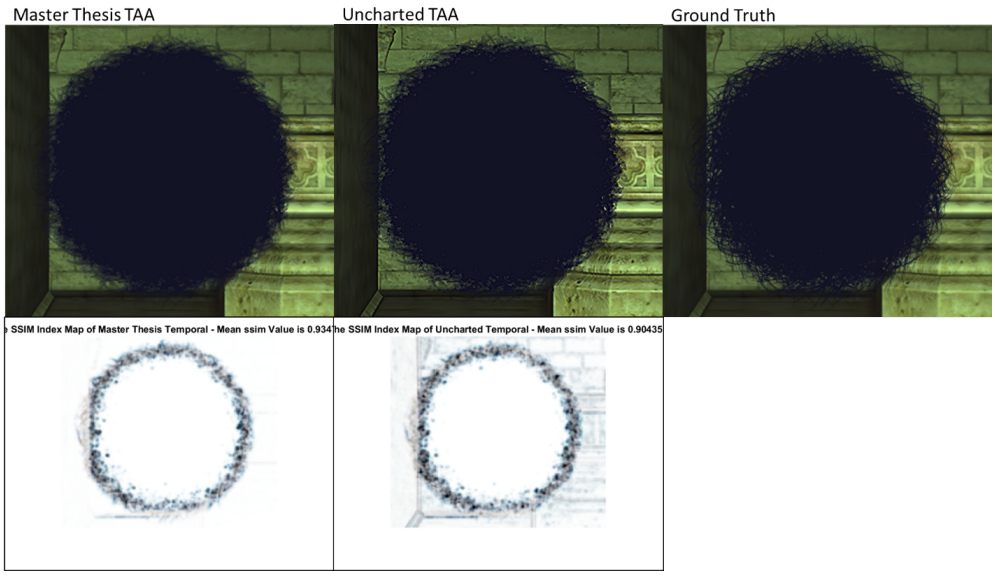


Figure 5.22: Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 90.

5.2.9.4 Ghosting Light

Table 5.13: Numerical results summary of the 100 tests performed for the Hairball Ghosting Light Test.

Hairball Light Shadow Test Summary				
AA / Tests	Uncharted TAA	Test Index	Master TAA	Test Index
Best MSE	714.811	62	603.190	1
Worst MSE	980.701	83	749.516	99
Average MSE	875.687	N/A	671.202	N/A
Best Peak-SNR	19.589	62	20.326	1
Worst Peak-SNR	18.215	83	19.383	99
Average Peak-SNR	18.737	N/A	19.867	N/A
Best SNR	10.037	62	10.913	1
Worst SNR	8.666	83	9.902	99
Average SNR	9.261	N/A	10.390	N/A
Best SSIM	0.738	62	0.766	1
Worst SSIM	0.662	99	0.694	83
Average SSIM	0.691	N/A	0.722	N/A

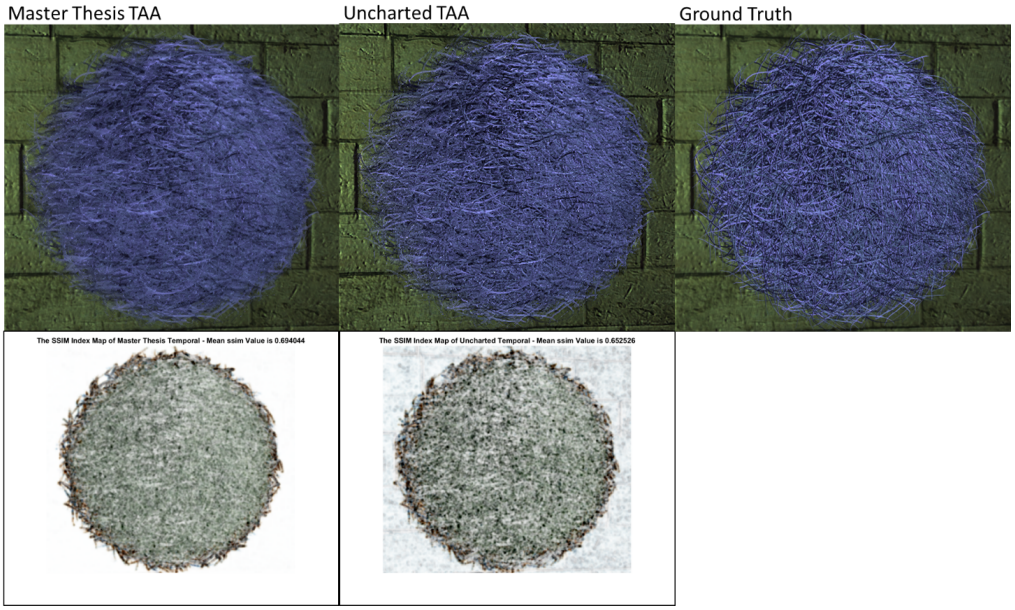


Figure 5.23: Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 83.

5.2.10 Timing

The measured time the Master Thesis TAA technique took to run was between 0.5 and 0.6 ms on average. The Sobel pass took between 0.2 ms and 0.3 ms, and the reprojection was around 0.3 ms.

5.3 Discussion

As we appreciate the results of the Sharpen Filter Test, see Table 5.1, the improvements achieved in this master thesis go beyond modifying the filter from the Uncharted implementation. We believe that their use of that specific filter is of artistic nature. It tends to pronounce the edges at the cost of creating white pixels that flicker, especially, when the camera moves while the foreground is illuminated but the background is on shadows or vice versa.

From both Pipe Tests we can observe that the results from the Master Thesis TAA are close to SMAA results when drawing hard edges. We can quantify the reduction of blurring when comparing to the Uncharted TAA implementation in the numerical results from the Tables 5.2 and 5.3. When we compare the SSIM maps results (Figures 5.3 and 5.5), we observe that the thick error line around the edges in the Uncharted implementation is not present our Master Thesis implementation. But, the blurring reduction is not perfect, as we see on the tests scores, the blurring that remains around the edges lets SMAA the high score on some tests.

From the Window with Blinds and Arched Window Tests we can appreciate how the techniques react to small, almost pixel sized, features like the blinds and the doors from the Arched Window. Although the Master Thesis TAA and SMAA appear numerically (See Tables 5.4 and 5.5) better than the Uncharted TAA, they are not able to reconstruct all the small details leaving pixel thin stripes that flicker when the camera moves. We believe that in this case, admitting the blurring of the Uncharted TAA benefits its final application because losing some small details is better than having many pixels flickering each time the camera moves.

The Sponza Atrium Test shows us that the Master Thesis TAA is more than capable of handling a general scene with lighting and shadows. As seen in the Table 5.6, our implementation proved to be better than the other Anti-Aliasing techniques by a fair margin in almost all the tests.

We consider the Sponza Atrium Flowers Test a distinctive experiment because the flowers are a 2D flat surface with many complex transparent holes and they are rotated around the column. As we observe from the numerical results in the Table 5.7 and SSIM maps on Figure 5.13, all the techniques struggle with those transparent holes but our Master Thesis TAA proves to be the best at handling them. We see from this test that our implementation is good at handling this type of small details, compared to the Windows Tests, because they are larger than just a pixel.

From the Hard Edges Test we continue to observe that the Master Thesis technique handles better the blur compared to the Uncharted TAA (See Table 5.8), especially on the letters, the pipe, and the square; and that the Master Thesis technique still has a hard time handling the super fine details from the windows, at this distance we still believe the blurring of the Uncharted TAA helps to hide the unwanted pixel stripes that flicker (See Figure 5.15).

In the Sphere Ghosting Test we see a clear example of the improvements accomplished in this Thesis. Figure 5.17 shows one example of the ghosting that is created by the Uncharted TAA implementation, we can clearly perceive the stripes that are left by the sphere while it moves, whereas on our Master Thesis TAA implementation they are barely visible.

Finally, we have the four Hairball Tests, the most complex tests performed. The Hair-

ball model has many gaps and small details that react to lighting and shadows. We anticipated a high amount of errors due to them because all Anti-Aliasing techniques have difficulties reconstructing this high density of fine details.

On the Shadow and Light Static Test (See Tables 5.10 and 5.11) we can observe that the Master Thesis Implementation is the best at handling the hair fibers. It is able to reconstruct smoother edges than the Uncharted TAA and reconstructs more details than SMAA technique. Especially on the light version (See Figure 5.20), we can appreciate how smooth the result is; it looks almost like the ground truth. This test far exceeded the expectation of our improvements, the visual and numerical (See Table 5.11) results shows a big increase in quality compared to the other Anti-Aliasing solutions.

On the Shadow and Light Ghosting Test we observe that both techniques results are blurred excessively (See Figures 5.22 and 5.23), especially the Master Thesis TAA implementation on the light version. We believe this to be caused by the History Buffer and the Sobel Temporal Pass, for the Master Thesis TAA, not being able to stabilize as fast as the colors change when the fibers move thanks to the color rejection being slower than needed. The numerical values from Tables 5.12 and 5.13 confirm the effects of blurring thanks to the MSE being higher than normal.

From our timing results (See 5.2.10), we can see that our improvements fit the time requirements to run on real-time applications.

Chapter 6
Conclusions and Recommendations

As we have shown numerically and visually with the tests performed, the Master Thesis implementation accomplished its objective of reducing the effects of blurring and ghosting of the Temporal Anti-Aliasing technique with the use of edge detection of both color and depth, and triangle indexing. Our results show that this technique can provide the same or better quality than other standard Anti-Aliasing solutions.

As possible improvements, we are confident that our implementation could be optimized to run faster than our current timing. Our current implementation was made with flexibility in mind to help us test different approaches, this could be simplified to reduce the number of passes required.

As for recommendations for further research, we suggest improving the technique's behavior under high detail density moving objects, like on the Hairball Tests. Another improvement subject is the stability for pixel size details that cause flickering, like on the Windows Tests. Furthermore, a Specular Lighting Anti-Aliasing solution compatible with Temporal Anti-Aliasing is still required to provide a full range solution to Aliasing in real-time applications. Also, we suggest searching for more specific Image Metrics for Computer Graphics, especially, finding tuning values for SSIM to provide more numerical resolution when comparing different rendered images.

Bibliography

[1] Tomas Akenine-Möller. [mobile] graphics hardware. Pages 7-8.

[2] Yusra A. Y. Al-Najjar and Dr. Der Chen Soong. Comparison of image quality assessment: PSNR, HVS, SSIM, UIQI. *International Journal of Scientific & Engineering Research*, August 2012. Volume 3, Issue 8.

[3] E. Angel and D. Shreiner. *Computer Graphics A Top-Down Approach with Shader-Based OpenGL 6th Edition*. Addison-Wesley, 2011.

[4] Specter Arts. Pipe model, March 2010. Accessed: 2018-02-15, `https://www.turbosquid.com/3d-models/pipe-stone-bowl-3d-model/522479`.

[5] John Chapman. Per-Object Motion Blur, September 2012. Accessed: 2017-11-30, `http://john-chapman-graphics.blogspot.se/2013/01/per-object-motion-blur.html`.

[6] Isabela H. Arched window model, January 2016. Accessed: 2018-02-15, `https://www.cgtrader.com/free-3d-models/architectural/window/window-arch`.

[7] Paul Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. *ACM Computer Graphics*, August 1990. Volume 24, Number 4, August 1990.

[8] Jorge Jimenez, Jose I Echevarria, Tiago Sousa, and Diego Gutierrez. SMAA: Enhanced subpixel morphological antialiasing. *EUROGRAPHICS 2012 / P. Cignoni, T. Ertl Volume 31 (2012), Number 2*, 2012.

[9] Jorge Jimenez, Diego Gutierrez, Jason Yang, Alexander Reshetov, Pete Demoreuille, Tobias Berghoff, Cedric Perthuis, Henry Yu, Morgan McGuire, Timothy Lottes, Hugh Malan, Emil Persson, Dmitry Andreev, and Tiago Sousa. Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses*, 2011.

[10] Timothy Lottes. FXAA. *NVIDIA Docs*, 2009. The Document was last edited in 2011.

[11] W. S. Malpica and A. C. Bovik. Range image quality assessment by structural similarity. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1149–1152, April 2009.

[12] Morgan McGuire. Computer graphics archive, July 2017. Accessed: 2018-02-15, `https://casual-effects.com/data`.

[13] Michael Doggett. EDAF80 Computer Graphics, September 2017.

[14] Michael Doggett. EDAN35 High Performance Computer Graphics, November 2017.

[15] D. Nehab, P. V. Sander, J. Lawrence, N. Tatarchuk, and J. R. Isidoro. Accelerating real-time shading with reverse reprojection caching. *In Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware (2007), pp. 25–35. 3, 8*, 2007.

[16] Lasse Jon Fuglsang Pedersen. Temporal Reprojection Anti-Aliasing in INSIDE. *GDC Vault*, 2016. Accessed: 2017-11-28, `http://www.gdcvault.com/play/1022970/Temporal-Reprojection-Anti-Aliasing-in`.

[17] Irwin Sobel. An isotropic 3x3 image gradient operator. *Irwin Sobel Correspondence*, 02 2014.

[18] Bart Wronski. Temporal Supersampling and Antialiasing. `https://bartwronski.com/2014/03/15/temporal-supersampling-and-antialiasing/`, March 2014. Accessed: 2017-12-01.

[19] Ke XU. Temporal Antialiasing In Uncharted 4. *SIGGRAPH 2016*, 2016.

[20] L. Yang, D. Nehab, P. V. Sander, P. Sitthiamorn, J. Lawrence, and H. Hoppe. Amortized supersampling. *ACM Trans. Graph. 28 (2009), 135:1–135:12*, 2009.

[21] Channa Yim. Window with blinds model, October 2015. Accessed: 2018-02-15, `https://www.cgtrader.com/free-3d-models/architectural/window/window-brown`.

Appendices

Appendix A
GitHub Repository

- Full Repository.
- Complete Computer Specification.
- All Tests.
- Accumulation Buffer Tests.
- Most of the Master Thesis Tests.
- Sharpening Tests.
- Ghosting Tests.
- HairBall Tests.
- Code.
- LaTeXReport.

Appendix B

Camera Jittering Explanation

The $HaltonSequence(2,3)$ generates points in the $[0,1] \times [0,1]$ space. First, we need to transform it to the $[-1,1] \times [-1,1]$ space because we consider the pixel to be at the center, i.e. in OpenGL the first pixel is at $(0.5, 0.5)$, so we apply the transformation $T_1(x,y) = 2 * (x,y)(1,1)$.

Now, we only want to jitter inside the pixel because we should only be sampling inside of it. We want to control this but for explanation purposes, we can assume that we only want it inside the pixel. So, we apply the transformation $T_2(x,y) = \frac{(x,y)}{2}$.

From now on, we need to change how we interpret the process we are performing; we are calculating the distance we are going to move the pixel center, so we need to see it as a vector rather than a point. We need to transform the vector to a vector normalized by the screen size. Consequently, we apply the transformation $T_3(x,y) = (x,y)/(w,h)$ with (w,h) being the Width and Height of the screen and "/" operator as component-wise division.

Now we need to transform our vector normalized by the screen size $[0,1]$ to the Normalized Device Coordinates which go in the range $[-1,1] \times [-1,1]$. This is done by using $T_4(x,y) = 2 * (x,y)$ (to transform points we would use $2 * (x,y)(1,1)$). At the end, our transformation would look like this $T(x,y) = T_4(T_3(T_2(T_1(x,y)))) = (2 * (x,y)(1,1))/(w,h)$ with (x,y) being a Halton Sequence point.

We now need to modify our Projection Matrix, which takes points from the View Space into the Clip Space. The resulting matrix is taken from the Ke Xu presentation page 14 [19]. Let (h_x, h_y) be jitter we have previously calculated.

$$
\begin{aligned}
JitteredProjection &= \begin{bmatrix} a & 0 & h_x & 0 \\ 0 & b & h_y & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{bmatrix} = JitterMatrix \times Projection \\
&= \begin{bmatrix} 1 & 0 & 0 & -h_x \\ 0 & 1 & 0 & -h_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{bmatrix}
\end{aligned} \tag{B.1}
$$

Let's see its effect to a point in View Space, let $p_{view} = (x, y, z, 1)$.

$$
JitteredProjection \times p_{view} = \begin{bmatrix} a * x + h_x * z \\ b * y + h_y * z \\ c * z + d \\ -z \end{bmatrix} \tag{B.2}
$$

We proceed to do the Perspective Divide to transform the point to NDC Coordinates, this is accomplished by dividing the vector by the w component.

$$
\begin{bmatrix} -a * \frac{x}{z} - h_x \\ -b * \frac{y}{z} - h_y \\ -c + \frac{d}{z} d \\ 1 \end{bmatrix} = p_{NDC} + \begin{bmatrix} -h_x \\ -h_y \\ 0 \\ 0 \end{bmatrix} \tag{B.3}
$$

Accordingly, we only need to be sure we can apply the Jitter Matrix alone to use it inside Pixel Shaders with points in NDC space.

$$
JitterMatrix \times p_{NDC} = \begin{bmatrix} x_{NDC} - h_x \\ y_{NDC} - h_y \\ z_{NDC} \\ 1 \end{bmatrix} = p_{NDC} + \begin{bmatrix} -h_x \\ -h_y \\ 0 \\ 0 \end{bmatrix} \tag{B.4}
$$