# Improved Sampling for Temporal Anti-Aliasing

(A Sobel Improved Temporal Anti-Aliasing)

Christian Alexander Oliveros Labrador
christianol_01@hotmail.com

April 26, 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 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 frames pixel 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.

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

# Acknowledgements

If you want to thank people, do it here, on a separate right-hand page. Both the U.S. *acknowledgments* and the British *acknowledgements* spellings are acceptable.

We would like to thank Lennart Andersson for his feedback on this template.

We would also like thank Camilla Lekebjer for her contribution on this template, as well as Magnus Hultin for his popular science summary class and example document.

# 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 on 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 videogames. 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 Temporal Anti-Aliasing implementation, 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 in an average computer or mobile device. [15, 3]

## 1.1 Problem Definition

Temporal Anti-Aliasing (TAA) is a relatively new real time technique that provides good results without incurring in heavy memory or processing power costs of other techniques. Edge detection and triangle indexing 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 [15]. 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. [16], in which they describe how pixel shaders could use save information and reproject it in the next frame; and Amortized Supersampling by Yang L., Nehab D., Sander P. V., Sitthiamorn P., Lawrence J., Hoppe H. [25] 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 [13] 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 [12].

Enhanced Subpixel Morphological Antialiasing (SMAA) by Jorge Jimenez, Jose I. Echeverria, Tiago Sousa and Diego Gutierrez [11] 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. This two last implementations are used the base of this master thesis. [18, 23]

# Chapter 2
# Technological Background

## 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 is 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 Graphic Processor Units (GPUs) 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 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 applica-

tions because of its wide library of algorithms for this purpose, including image metrics which are used in this thesis.

# Chapter 3

# Theoretical Background

## 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 prepare 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.

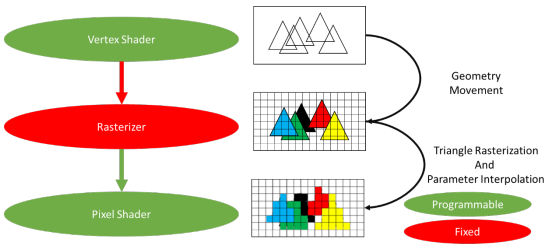


**Figure 3.1:** Rendering Pipeline, based on EDAF80 5th Lecture. [14]

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. [14]

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.

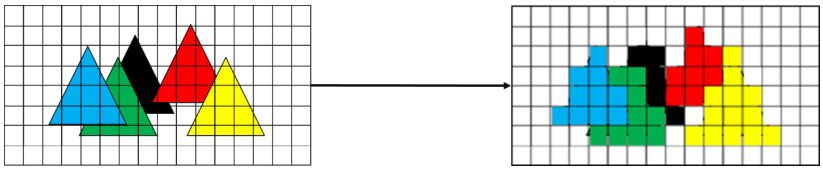

**Figure 3.2:** Example of the results of the Rasterization Process. *Note*: colors where added to differentiate the triangles but they would only be added by the Pixel Shader.

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 whether is it 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 precision of the representation used for the vertices.

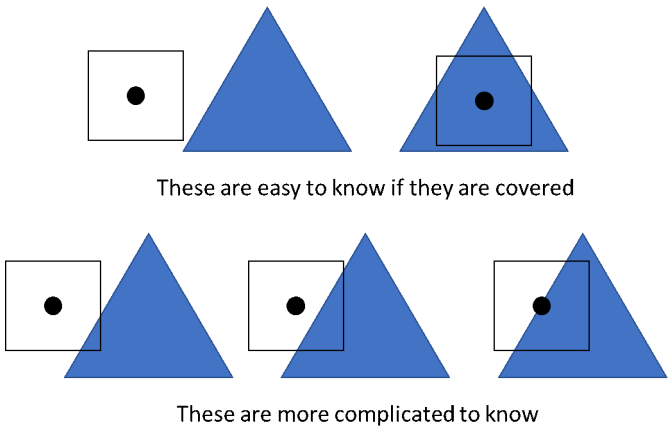


**Figure 3.3:** Example of the Partial Cover problem, based on EDAN35 2th Lecture. [15]

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, 15]

## 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 relates 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 with Doggett and Wronski [15, 22]. 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 Michael Dogget [15], 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, it first renders the scene, without light calculations, to a buffer called the Geometry Buffer. There, information regarding colors, normals, depths, specific objects information to interact with lights, etc. is saved so we do not need to recalculate that for each light. After all information is saved in the Geometry Buffer, we calculate the Shadow Map of every light but only performing depth calculations. Then we calculate the effect of the light using it. At the end, we take all the information of the lights, shadows and the Geometry Buffer to render the lighted scene.

## 3.5 Anti-Aliasing

As we know, 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. This increase is of great importance for real-time applications which try to use the best possible solutions that increase the least the rendering time.

Another important factor that decides which Anti-Aliasing 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)

As explained by Michael Dogget [15], this technique consists in rendering the scene at 4 times the size of the screen and then averaging pixels 4x4 to calculate the result. 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.

As explained by Michael Dogget [15], the biggest problem this technique has is that it does not work properly with Deferred Shading. This make 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 smooth them, as explained by Timothy Lottes. [13]

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 regarding 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, as explained by Jorge Jimenez, Jose I. Echeverria, Tiago Sousa and Diego Gutierrez. [11]

## 3.6 Temporal Anti-Aliasing

As explained by Ke Xu and Lasse Fuglsang in their respective presentations [18, 23], 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 at the same moment.

One of such techniques 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 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. [18, 23]

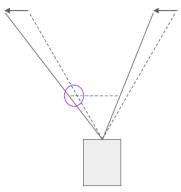

**Figure 3.5:** Jittering the Projection Process. Image taken from Fuglsand presentation. [18]

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. [18, 23].

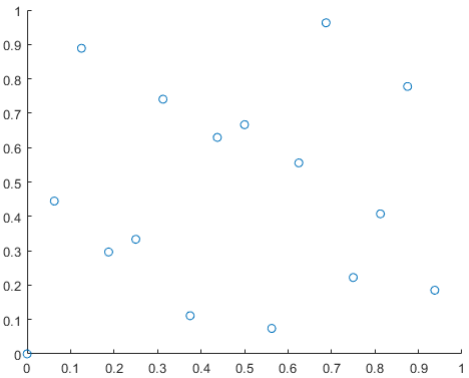


**Figure 3.6:** Values from the $HaltonSequence(2,3)$ used.

The Figure 3.6 shows the representation of the 16 points used to jitter the projection the current implementation, as proposed by Fuglsand [18]. 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 [4] 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 [23], 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 on both patterns for the minimum and maximum of colors of the current frame, next we average them and use it in the Clipping of History [18].

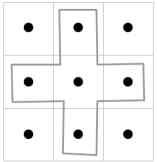

**Figure 3.7:** Sampling Pattern used. Image taken from Fuglsand presentation. [18]

On the $3 \times 3$ 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 [18]. We use this velocity to reproject the position of the current frame in the history. [18, 23]

After we have the history, 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 history. This mix stabilizes the image, removing the jittering and smoothing the edges [18, 23]. Because history is accumulated, we get the effect that each frame weights less the more time the history is not rejected. [18]

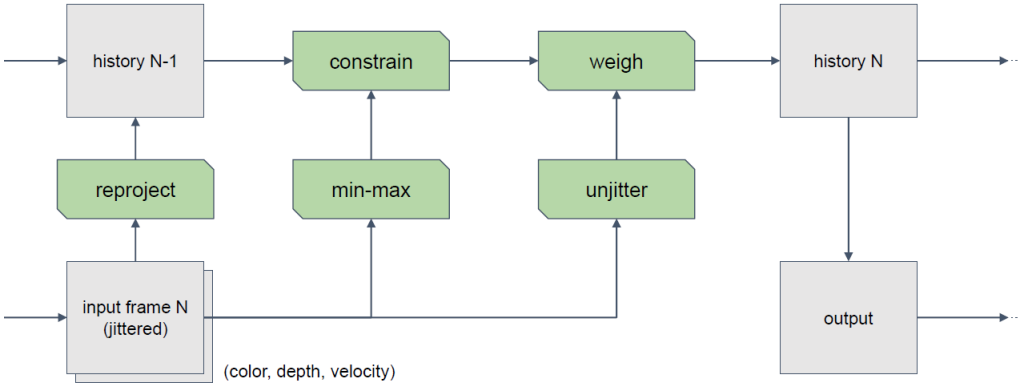


**Figure 3.8:** Temporal Reprojection Anti-Aliasing process. Image taken from Fuglsand presentation [18]

### 3.6.4 Clipping Color Box

A Clipping Color Box is used to handle color rejection when history 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 prevent color clustering that would happen if Clamp is applied (See Figure 3.9). [18].

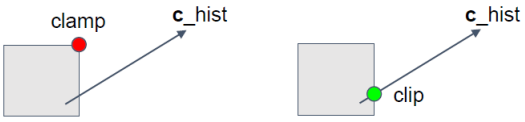


**Figure 3.9:** Color Clamping versus Color Clipping. Image taken from Fuglsand presentation. [18]

### 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. [23]

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

Equation 3.1 being the Sharpen Filter Convolution Matrix used in Xu presentation. [23]

### 3.6.6 Motion Blur

Because 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 [18, 23] proposed to use Motion Blur solutions to hide these artifacts.

The Motion Blur used is the one proposed by Chapman [4]. 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 [23], it solves blurriness reasonably well but it cannot eliminate some artifacts.

#### 3.6.7.2 Ghosting

Some Ghosting are 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 increase the size of everything using an Stencil technique and manual tagging of objects [23]. Pederson implementation allows the jitter in the Velocity Buffer calculations to avoid ghosting but it creates some unwanted blurriness [18].

## 3.7 Accumulation Buffer

The Accumulation Buffer is an anti-aliasing technique that consists, according to Paul Haeberli and Kurt Akeley [9], 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 at the cost of rendering everything several times per frame.

## 3.8 Sobel Operator

It is an efficiently computable $3 \times 3$ isotropic gradient operator, as explained by Irwin Sobel. [20]

It works by taking the four-possible simple central gradient estimates in a $3 \times 3$ 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.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3.2}$$

Let 3.2 be the $3 \times 3$ 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 $3 \times 3$ 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 to 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 new 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 Yusra A. Y. Al-Najjar and Dr. Der Chen 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 in three groups, as described by Yusra A. Y. Al-Najjar and Dr. Der Chen Soong [2]:

- **No-Reference:** In which we have no reference image to compare with.
- **Reduced-Reference:** Where we have partially 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.

# Chapter 4
# Formatting

Avoid empty spaces between *chapter-section*, *section-sub-section*. For instance, a very brief summary of the chapter would be one way of bridging the chapter heading and the first section of that chapter.

## 4.1 Page Size and Margins

Use A4 paper, with the text margins given in Table 4.1.

**Table 4.1:** Text margins for A4.

| margin | space |
|---|---|
| top | 3.0cm |
| bottom | 3.0cm |
| left (inside) | 2.5cm |
| right (outside) | 2.5cm |
| binding offset | 1.0cm |

## 4.2 Typeface and Font Sizes

The fonts to use for the reports are **TeX Gyre Termes** (a **Times New Roman** clone) for serif fonts, **TeX Gyre Heros** (a **Helvetica** clone) for sans-serif fonts, and finally **`TeX Gyre Cursor`** (a **`Courier`** clone) as mono-space font. All these fonts are included with the TeXLive 2013 installation. Table 4.2 lists the most important text elements and the associated fonts.

**Table 4.2:** Font types, faces and sizes to be used.

| Element | Face | Size | LaTeXsize |
|---|---|---|---|
| **Ch. label** | **serif, bold** | 24.88pt | `\huge` |
| **Chapter** | **serif, bold** | 24.88pt | `\Huge` |
| **Section** | **sans-serif, bold** | 20.74pt | `\LARGE` |
| **Subsection** | **sans-serif, bold** | 17.28pt | `\Large` |
| **Subsubsection** | **sans-serif, bold** | 14.4pt | `\large` |
| Body | serif | 12pt | `\normalsize` |
| Header | serif, SmallCaps | 10pt | |
| Footer (page numbers) | serif, regular | 12pt | |
| **Figure label** | **serif, bold** | 12pt | |
| Figure caption | serif, regular | 12pt | |
| In figure | sans-serif | *any* | |
| **Table label** | **serif, bold** | 12pt | |
| Table caption and text | serif, regular | 12pt | |
| `Listings` | `mono-space` | $\leq$ 12pt | |

### 4.2.1 Headers and Footers

Note that the page headers are aligned towards the outside of the page (right on the right-hand page, left on the left-hand page) and they contain the section title on the right and the chapter title on the left respectively, in SmallCaps. The footers contain only page numbers on the exterior of the page, aligned right or left depending on the page. The lines used to delimit the headers and footers from the rest of the page are $0.4pt$ thick, and are as long as the text.

### 4.2.2 Chapters, Sections, Paragraphs

Chapter, section, subsection, etc. names are all left aligned, and numbered as in this document.

Chapters always start on the right-hand page, with the label and title separated from the rest of the text by a $0.4pt$ thick line.

Paragraphs are justified (left and right), using single line spacing. Note that the first paragraph of a chapter, section, etc. is not indented, while the following are indented.

### 4.2.3 Tables

Table captions should be located above the table, justified, and spaced 2.0cm from left and right (important for very long captions). Tables should be numbered, but the numbering is up to you, and could be, for instance:

- **Table X.Y** where X is the chapter number and Y is the table number within that chapter. (This is the default in LaTeX. More on LaTeX can be found on-line, including

whole books, such as [8].) or

- **Table Y** where Y is the table number within the whole report

As a recommendation, use regular paragraph text in the tables, bold headings and avoid vertical lines (see Table 4.2).

### 4.2.4 Figures

Figure labels, numbering, and captions should be formed similarly to tables. As a recommendation, use vector graphics in figures (Figure 4.1), rather than bitmaps (Figure 4.2). Text within figures usually looks better with sans-serif fonts.


**Figure 4.1:** A PDF vector graphics figure. Notice the numbering and placement of the caption. The caption text is indented 2.0cm from both left and right text margin.


**Figure 4.2:** A JPEG bitmap figure. Notice the bad quality of such an image when scaling it. Sometimes bitmap images are unavoidable, such as for screen dumps.

For those interested in delving deeper into the design of graphical information display, please refer to books such as [21, 7].

## 4.3 Mathematical Formulae and Equations

You are free to use in-text equations and formulae, usually in *italic serif* font. For instance: $S = \sum_i a_i$. We recommend using numbered equations when you do need to refer to the specific equations:

$$E = \int_0^\delta P(t)dt \quad \longleftrightarrow \quad E = mc^2 \tag{4.1}$$

The numbering system for equations should be similar to that used for tables and figures.

## 4.4 References

Your references should be gathered in a **References** section, located at the end of the document (before **Appendices**). We recommend using number style references, ordered as appearing in the document or alphabetically. Have a look at the references in this template in order to figure out the style, fonts and fields. Web references are acceptable (with restraint) as long as you specify the date you accessed the given link [19, 6]. You may of course use URLs directly in the document, using mono-space font, i.e. `http://cs.lth.se/`.

## 4.5 Colours

As a general rule, all theses are printed in black-and-white, with the exception of selected parts in selected theses that need to display colour images essential to describing the thesis outcome (*computer graphics*, for instance).

A strong requirement is for using **black text on white background** in your document's main text. Otherwise we do encourage using colours in your figures, or other elements (i.e. the colour marking internal and external references) that would make the document more readable on screen. You may also emphasize table rows, columns, cells, or headers using white text on black background, or black text on light grey background.

Finally, note that the document should look good in black-and-white print. Colours are often rendered using monochrome textures in print, which makes them look different from on screen versions. This means that you should choose your colours wisely, and even opt for black-and-white textures when the distinction between colours is hard to make in print. The best way to check how your document looks, is to print out a copy yourself.

# Chapter 5
# Language

You are strongly encouraged to write your report in English, for two reasons. First, it will improve your use of English language. Second, it will increase visibility for you, the author, as well as for the Department of Computer Science, and for your host company (if any).

However, note that your examiner (and supervisors) are not there to provide you with extensive language feedback. We recommend that you check the language used in your report in several ways:

**Reference books** dedicated to language issues can be very useful. [10]

**Spelling and grammar checkers** which are usually available in the commonly used text editing environments.

**Colleagues and friends** willing to provide feedback your writing.

**Studieverkstaden** is a university level workshop, that can help you with language related problems (see Studieverkstaden's web page).

**Websites** useful for detecting language errors or strange expressions, such as

- `http://translate.google.com`
- `http://www.gingersoftware.com/grammarcheck/`

## 5.1 Style Elements

Next, we will just give some rough guidelines for good style in a report written in English. Your supervisor and examiner as well as the aforementioned **Studieverkstad** might have a different take on these, so we recommend you follow their advice whenever in doubt. If you want a reference to a short style guide, have a look at [17].

#### 5.1.0.1 Widows and Orphans

Avoid *widows* and *orphans*, namely words or short lines at the beginning or end of a paragraph, which are left dangling at the top or bottom of a column, separated from the rest of the paragraph.

#### 5.1.0.2 Footnotes

We strongly recommend you avoid footnotes. To quote from [5], *Footnotes are frequently misused by containing information which should either be placed in the text or excluded altogether. They should be avoided as a general rule and are acceptable only in exceptional cases when incorporation of their content in the text [is] not possible.*

#### 5.1.0.3 Active vs. Passive Voice

Generally active voice (*I ate this apple.*) is easier to understand than passive voice (*This apple has been eaten (by me).*) In passive voice sentences the actor carrying out the action is often forgotten, which makes the reader wonder who actually performed the action. In a report is important to be clear about who carried out the work. Therefore we recommend to use active voice, and preferably the plural form *we* instead of *I* (even in single author reports).

#### 5.1.0.4 Long and Short Sentences

A nice brief list of sentence problems and solutions is given in [24]. Using choppy sentences (too short) is a common problem of many students. The opposite, using too long sentences, occurs less often, in our experience.

#### 5.1.0.5 Subject-Predicate Agreement

A common problem of native Swedish speakers is getting the subject-predicate (verb) agreement right in sentences. Note that a verb must agree in person and number with its subject. As a rough tip, if you have subject ending in *s* (plural), the predicate should not, and the other way around. Hence, *only one s*. Examples follow:

**incorrect** He have to take this road.

**correct** He has to take this road.

**incorrect** These words forms a sentence.

**correct** These words form a sentence.

In more complex sentences, getting the agreement right is trickier. A brief guide is given in the *20 Rules of Subject Verb Agreement* [26].

# Chapter 6
# Structure

It is a good idea to discuss the structure of the report with your supervisor rather early in your writing. Given next is a generic structure that is a starting point, but by no means the absolute standard. Your supervisor should provide a better structure for the specific field you are writing your thesis in. Note also that the naming of the chapters is not compulsory, but may be a helpful guideline.

**Introduction** should give the background of your work. Important parts to cover:

- Give the context of your work, have a short introduction to the area.
- Define the problem you are solving (or trying to solve).
- Specify your contributions. What does this particular work/report bring to the research are or to the body of knowledge? How is the work divided between the co-authors? (This part is essential to pinpoint individual work. For theses with two authors, it is compulsory to identify which author has contributed with which part, both with respect to the work and the report.)
- Describe related work (literature study). Besides listing other work in the area, mention how is it related or relevant to your work. The tradition in some research area is to place this part at the end of the report (check with your supervisor).

**Approach** should contain a description of your solution(s), with all the theoretical background needed. On occasion this is replaced by a subset or all of the following:

- **Method**: describe how you go about solving the problem you defined. Also how do you show/prove that your solution actually works, and how well does it work.
- **Theory**: should contain the theoretical background needed to understand your work, if necessary.

- **Implementation**: if your work involved building an artefact/implementation, give the details here. Note, that this should not, as a rule, be a chronological description of your efforts, but a view of the result. There is a place for insights and lamentation later on in the report, in the Discussion section.

**Evaluation** is the part where you present the finds. Depending on the area this part contains a subset or all of the following:

- **Experimental Setup** should describe the details of the method used to evaluate your solution(s)/approach. Sometimes this is already addressed in the **Method**, sometimes this part replaces **Method**.
- **Results** contains the data (as tables, graphs) obtained via experiments (benchmarking, polls, interviews).
- **Discussion** allows for a longer discussion and interpretation of the results from the evaluation, including extrapolations and/or expected impact. This might also be a good place to describe your positive and negative experiences related to the work you carried out.

Occasionally these sections are intermingled, if this allows for a better presentation of your work. However, try to distinguish between measurements or hard data (results) and extrapolations, interpretations, or speculations (discussion).

**Conclusions** should summarize your findings and possible improvements or recommendations.

**Bibliography** is a must in a scientific report. LaTeX and `bibtex` offer great support for handling references and automatically generating bibliographies.

**Appendices** should contain lengthy details of the experimental setup, mathematical proofs, code download information, and shorter code snippets. Avoid longer code listings. Source code should rather be made available for download on a website or on-line repository of your choosing.

# 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] 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`.

[5] Commonwealth Forestry Association. On-line guide to scientific writing. `http://www.cfa-international.org/ONGSWfinish.html`.

[6] CTAN. Comprehensive tex archive network. `http://www.ctan.org`.

[7] S. Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. Analytics Press, 2012.

[8] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[9] 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.

[10] J.A.W. Heffernan, J.E. Lincoln, and J. Atwill. *Writing: A College Handbook*. W.W. Norton, 2000.

[11] 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.

[12] 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.

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

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

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

[16] 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.

[17] University of Washington. Style points for scientific writing. `http://www.psych.uw.edu/writingcenter/writingguides/pdf/style.pdf`.

[18] 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`.

[19] Will Robertson and Khaled Hosny. The fontspec package, May 2012. `http://ctan.uib.no/macros/latex/contrib/fontspec/fontspec.pdf`.

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

[21] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, Cheshire, CT, USA, 1986.

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

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

[24] Yale Graduate Writing Center. The most common sentence structure problems. `http://www.yale.edu/graduateschool/writing/forms/The%20Most%20Common%20Sentence%20Structure%20Problems.pdf`.

[25] 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.

[26] yourdictionary.com. 20 rules of subject verb agreement. `http://grammar.yourdictionary.com/sentences/20-Rules-of-subject-verb-agreement.html`.

# Appendices

# Appendix A
# About This Document

---

The following environments and tools were used to create this document:

- operating system: Mac OS X 10.10.1
- tex distribution: MacTeX-2014, `http://www.tug.org/mactex/`
- tex editor: Texmaker 4.4.1 for Mac, `http://www.xm1math.net/texmaker/` for its XeLaTeX flow (recommended) or pdfLaTeX flow
- bibtex editor: BibDesk 1.6.3 for Mac, `http://bibdesk.sourceforge.net/`
- fonts `cslthse-msc.cls` document class):

  for XeLaTeX: TeX Gyre Termes, TeX Gyre Heros, `TeX Gyre Cursor` (installed from the TeXLive 2013)

  for pdfLaTeX: TeX Gyre font packages: tgtermes.sty, tgheros.sty, tgcursor.sty, gtxmath.sty (available through TeXLive 2013)
- picture editor: OmniGraffle Professional 5.4.2

A list of the essential LaTeXpackages needed to compile this document follows (all except `hyperref` are included in the document class):

- `fontspec`, to access local fonts, needs the XeLaTeX flow
- `geometry`, for page layout
- `titling`, for formatting the title page
- `fancyhdr`, for custom headers and footers
- `abstract`, for customizing the abstract

- `titlesec`, for custom chapters, sections, etc.
- `caption`, for custom tables and figure captions
- `hyperref`, for producing PDF with hyperlinks
- `appendix`, for appendices
- `printlen`, for printing text sizes
- `textcomp`, for text companion fonts (e.g. bullet)
- `pdfpages`, to include the popular science summary page at the end

Other useful packages:

- `listings`, for producing code listings with syntax colouring and line numbers

# Appendix B
# List of Changes

---

#### B.0.0.1 Since 2016/04/29

- A better template for the popular science summary, by Magnus Hultin.

#### B.0.0.2 Since 2015/09/11

- Added a template for the popular science summary, in the `popsci` directory.
- Added code in the report that imports the one page popular science `pdf` at the end of the document.

#### B.0.0.3 Since 2015/04/27

- Improved the **Structure** chapter and added more detailed comments for each part.

#### B.0.0.4 Since 2014/02/18

- Added the possibility to specify two supervisors. Use either of the `\supervisor{}` or `\supervisors{}{}` commands to set the names and contacts on the first page.

#### B.0.0.5 Since 2013/09/23

- Added missing colon ":" after *Examiner* on the front page.

#### B.0.0.6 Since 2013/08/30

- Changed fonts from Garamond (Times New Roman), Helvetica (Arial), Courier (Source Code Pro) to Tex Gyre fonts, namely Termes, Heros, Cursor, which are

freely available with TexLive 2013 installation. These are all clones of Times New Roman, Helvetica and Courier, respectively. Garamond is problematic on some systems, being a non-freely available font.

- Corrected the *Face* column in Table 4.2 to correctly depict the font face.

### B.0.0.7 Since 2013/02/22

- Number of words required in the abstract changed to 150 (from 300).

### B.0.0.8 Since 2013/02/15

- Made a separate document class, for clarity.
- made it work with pdfLaTeX and garamond.sty, in addition to XeLaTeX and true type fonts. It is up to the user to get the hold of the garamond.zip from `http://gael-varoquaux.info/computers/garamond/index.html`.

**EXAMENSARBETE** Application Specific Instruction-set Processor Using a Parametrizable multi-SIMD Synthesizeable Model Supporting Design Space Exploration
**STUDENT** Magnus Hultin
**HANDLEDARE** Flavius Gruian (LTH)
**EXAMINATOR** Krzysztof Kuchcinski (LTH)

# Parametrisk processor modell för design utforskning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Magnus Hultin**

Applikations-specifika processorer är allt mer vanligt för få ut rätt prestanda med så lite resurser som möjligt. Detta arbete har en parametrisk modell för att kunna testa hur mycket resurser som behövs för en specifik applikation.

För att öka prestandan i dagens processorer finns det vektorenheter och flera kärnor i processorer. Vektorenheten finns till för att kunna utföra en operation på en mängd data samtidigt och flera kärnor gör att man kan utföra fler instruktioner samtidigt. Ofta är processorerna designade för att kunna stödja en mängd olika datorprogram. Detta resulterar i att det blir kompromisser som kan påverka prestandan för vissa program och vara överflödigt för andra. I t.ex. videokameror, mobiltelefoner, medicinsk utrustning, digital kameror och annan inbyggd elektronik, kan man istället använda en processor som saknar vissa funktioner men som istället är mer energieffektiv. Man kan jämföra det med att frakta ett paket med en stor lastbil istället för att använda en mindre bil där samma paketet också skulle få plats.

I mitt examensarbete har jag skrivit en modell som kan användas för att snabbt designa en processor enligt vissa parametrar. Dessa parametrar väljs utifrån vilket eller vilka program man tänkta köra på den. Vissa program kan t.ex. lättare använda flera kärnor och vissa program kan använda korta eller längre vektorenheter för dess data.

För att kunna välja vilken typ av processor som är rätt för den specifika applikationen krävs det ofta att man snabbt kan testa olika prototyper. Att implementera dessa till hårdvara kan ofta vara tidskrävande och ifall det visar sig att implementationen inte klarar dem kraven man ställt för prestanda och energieffektivitet, måste man designa för nya parametrar och mer tid har blivit slösat. Om den här processen istället kan göras automatiskt utifrån dessa design-parametrar kan man teoretiskt spara en massa tid. Modellen testades med olika multimedia program. Den mest beräkningsintensiva och mest upprepande delen av programmen användes. Dessa kallas för kärnor av programmen. Kärnorna som användes var ifrån MPEG och JPEG, som används för bildkomprimering och videokomprimering.

Resultatet visar att det finns en prestanda vinst jämfört med generella processorer men att detta också ökar resurserna som behövs. Detta trots att den generella processorn har nästan dubbelt så hög klockfrekvens än dem applikationsspecifika processorerna. Resultatet visar också att schemaläggning av instruktionerna i programmen spelar en stor roll för att kunna utnyttja resurserna som finns tillgängliga och därmed öka prestandan. Med den schemaläggningen som utnyttjade resurserna bäst var prestandan minst 79% bättre än den generella processorn.