UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Lázaro Ribeiro Monteiro Júnior

Anti-Aliasing Method Based on Rotated Spatial Filtering

Mossoró, RN

2021

Lázaro Ribeiro Monteiro Júnior

Anti-Aliasing Method Based on Rotated Spatial Filtering

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Computer Science in the Programa de Pós-Graduação em Ciência da Computação at UERN/UFERSA.

Advisor: Sílvio Roberto Fernandes de Araújo, Prof. Dr.
Co-Advisor: Leandro Carlos de Souza, Prof. Dr.

Mossoró, RN

2021

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei n° 9.279/1996 e Direitos Autorais: Lei n° 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

M772a Monteiro Júnior, Lázaro Ribeiro.
Anti-Aliasing Method Based on Rotated Spatial Filtering / Lázaro Ribeiro Monteiro Júnior. - 2021.
73 f. : il.

Orientador: Sílvio Roberto Fernandes de Araújo.
Coorientador: Leandro Carlos de Souza.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em Ciência da Computação, 2021.

1. Computação Gráfica. 2. Anti--Aliasing. 3. Filtragem Espacial. 4. Regressão Linear. I. Araújo, Sílvio Roberto Fernandes de, orient. II. Souza, Leandro Carlos de, co-orient. III. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

Lázaro Ribeiro Monteiro Júnior

Anti-Aliasing Method Based on Rotated Spatial Filtering

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Computer Science in the Programa de Pós-Graduação em Ciência da Computação at UERN/UFERSA.

Thesis approved in August 31, 2021:

Sílvio Roberto Fernandes de Araújo, Prof. Dr.
Advisor - UFERSA

Leandro Carlos de Souza, Prof. Dr.
Co-Advisor - UFPB

Angélica Félix de Castro, Profa. Dra.
Internal Examiner - UFERSA

Renata Maria Cardoso Rodrigues de Souza, Profa. Dra.
External Examiner - UFPE

Mossoró, RN
2021

"Everything is important- that
success is in the details."
(Steve Jobs)

Abstract

Due to the evolution of graphics processors over the last decades has become possible to produce high-quality and realistic three-dimensional scenes. However, aliasing is produced during the sampling performed in the rasterization process. It causes a serrated effect on the edges of the objects presented in the scene, highlighting the unreal aspect of the image and displeasing the viewer. The present thesis aims to develop an anti-aliasing treatment based on rotated spatial filtering. It performs edges recognition applying spatial filtering along with simple linear regression technique. Then, a smoothing spatial filter is rotated to match the direction of the inspected border and applied in the affected regions. The testing was performed on an OpenGL application, processing the rendered image from the framebuffer. Later, the application was modified to process images from Blender, a 3D modeling software that allows more complex graphics scenes. The results show the proposed method's effectiveness in smoothing aliasing with good quality and preserving the scene details. Hence, the problem was handled effectively with a post-filtering approach and without oversampling. The algorithm's execution time is $O(n)$, and the memory used is $O(n)$.

Keywords: Computer Graphics. Anti-Aliasing. Spatial Filtering. Linear Regression.

Resumo

Com a constante evolução dos processadores gráficos ao longo das últimas décadas, tornou-se possível produzir cenas tridimensionais com alta qualidade e realismo. No entanto, o *aliasing* é um problema produzido durante a amostragem de sinais realizada no processo de rasterização. Ele causa um efeito serrilhado nas bordas dos objetos apresentados em cena, ressaltando o aspecto irreal da imagem e causando desconforto visual ao espectador. O presente estudo busca desenvolver um tratamento *anti-aliasing* baseado em filtragem espacial rotacionada. O método proposto identifica as bordas na cena utilizando filtragem espacial junto a regressão linear simples. Então, um filtro espacial de suavização é rotacionado na mesma direção da borda inspecionada e aplicado nas regiões afetadas. Os testes foram realizados em uma aplicação *OpenGL*, processando a imagem renderizada do *framebuffer*. Posteriormente, a aplicação foi modificada para processar imagens do *Blender*, um software de modelagem 3D que permite a produção de cenas mais complexas. Os resultados mostram a eficácia do método ao suavizar o *aliasing* com boa qualidade e preservando os detalhes da cena. Desta forma, o problema foi tratado de forma eficaz com abordagem pós-filtragem e sem superamostragem. O tempo de execução do algoritmo é $O(n)$, assim como seu consumo de memória é $O(n)$.

Palavras-chave: Computação Gráfica. *Anti-Aliasing*. Filtragem Espacial. Regressão Linear.

List of Figures

List of Tables

List of abbreviations and acronyms

3D	*three-dimensional*
AA	*Adaptive Anti-Aliasing*
ACAA	*Accumulative Anti-Aliasing*
AGAA	*Aggregate G-Buffer Anti-Aliasing*
API	*Application Programming Interface*
CAD	*Computer-Aided Design*
CPU	*Central Processing Unit*
CSAA	*Coverage Sampling Anti-Aliasing*
DCAA	*Decoupled Coverage Anti-Aliasing*
DEAA	*Distance-to-Edge Anti-Aliasing*
DLAA	*Directionally Localized Anti-Aliasing*
FPGA	*Field Programmable Gate Array*
FXAA	*Fast ApproXimate Anti-Aliasing*
GBAA	*Geometry Buffer Anti-Aliasing*
GPAA	*Geometric Post-Process Anti-Aliasing*
GPU	*Graphics Processing Unit*
LSI	*Linear Shift-Invariant*
MLAA	*Morphological Anti-Aliasing*
MR	*Magnetic Resonance*
MSAA	*Multisample Anti-Aliasing*
NURBS	*Non-Uniform Rational B-Splines*
REPAIR	*Rotated Spatial Filtering*
SBAA	*Surface Based Anti-Aliasing*
S-MSAA	*Selective Multi-Sample Anti-Aliasing*

SNS	*Signal to Noise Ratio*
SRAA	*Subpixel Reconstruction Anti-Aliasing*
SSAA	*Supersampled Anti-Aliasing*

Contents

1 INTRODUCTION

Computer graphics is the science and art of visual communication through a screen and its interactive devices. It is a multidisciplinary knowledge field involving physics, math, human-machine interaction, engineering, graphics design, and mainly human perception (HUGHES et al., 2014).

An enormous advance in dedicated hardware to three-dimensional (3D) graphics processing happened in the last decades. In the earlier 1980s, Graphics Processing Units (GPUs) were large and expensive systems. Over the decades, these devices became small and their prices accessible, integrating workstations and laptops easily. Currently, GPUs are capable of executing trillions of float point operations per second (teraflops). Simultaneously, the development techniques also evolved and became more sophisticated (KIRK; HWU, 2013).

Nowadays, computer graphics allows the rendering of high-quality images. Also, digital media applies it in animation movies, games, simulations, and medical images, for example. New hardware devices have high performance making it possible to produce images using standard devices, like personal computers and smartphones (HUGHES et al., 2014).

Although, the presentation of digital images has a quality limitation, denominated aliasing. This problem occurs during the raster process, which tries to draw the colors of the image on discrete integer coordinates of the screen. These discrete areas are called pixels, and the image has limited resolution (HEARN; BAKER, 1997). Thus, it is inevitable that information is lost during the discretization process (GONZALEZ; WOODS, 2008)

In the viewer's perspective, the aliasing is a distortion of the scene information that makes lines and other graphics primitives look jagged, with a stair-step appearance (HEARN; BAKER, 1997). It disturbs mainly objects edges and it seems that affected objects are unrealistic and separated from the scene (HUGHES et al., 2014).

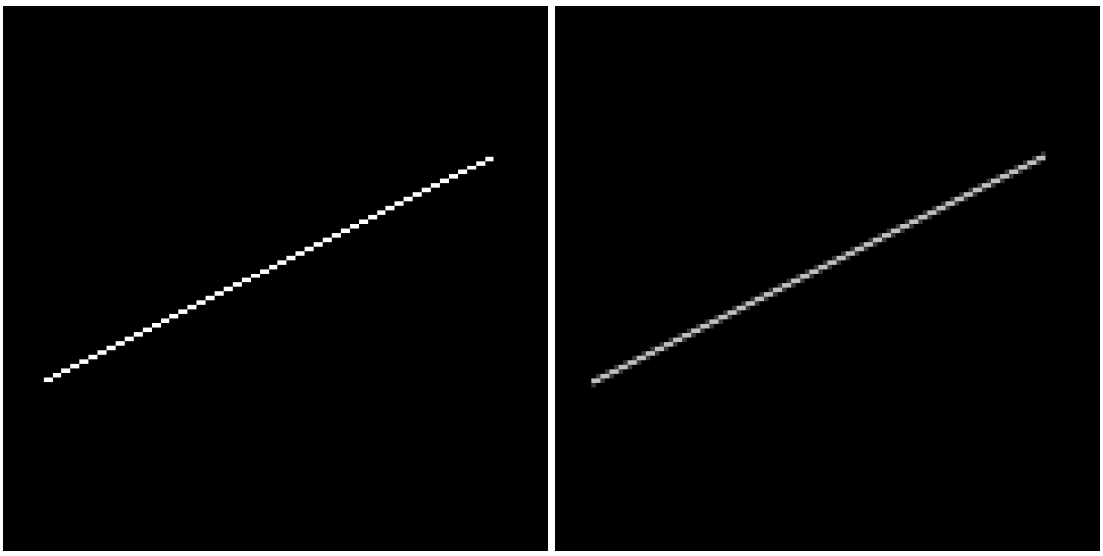
Even though aliasing is an unavoidable problem in scenes presentation, its effects can be reduced using methods to improve the appearance of displayed graphic primitives (HEARN; BAKER, 1997). These methods are called anti-aliasing (GONZALEZ; WOODS, 2009) and they are classified into three techniques: post-filtering, pre-filtering, and pixel phasing.

The post-filtering techniques are applied to rendered images. Usually needs a supersampled image and reduces its resolution to display on the screen; The pre-filtering techniques perform during the render process and usually compute overlap areas to determine object boundaries, where aliasing is more likely to appear. Pixel phasing techniques execute anti-aliasing by shifting the display location of pixel areas (HEARN; BAKER, 1997).

In literature, there is a variety of Anti-Aliasing methods. The Supersampled Anti-Aliasing

(SSAA) computes multiple samples independently for each pixel and the final result is resolved by averaging all samples. Thus, SSAA increases the sampling rate, increasing the computing effort to render the scene. The A-Buffer renderer computes, at each sample, the color for a fragment of a pixel, storing a list of fragments for each pixel. The pixel color is made up from a group of fragments, which allows a more accurate measurement of coverage and aliasing reduction. Although, the memory consumption is increased to store one extra buffer. Multisample Anti-Aliasing (MSAA) is similar to SSAA, but it applies a depth test to oversample only edge regions, reducing SSAA drawbacks. Although, it only reduces aliasing on geometry edges and may leave aliased regions untreated. An example of MSAA is shown in Figure 1.1. The Coverage Sampling Anti-Aliasing (CSAA) combines the strengths of the A-buffer and MSSA. It keeps a set of high-resolution samples, but instead of storing them, it stores pointers to a small set of colors. So, it allows a more accurate estimate of the area within the pixel using less memory than MSAA and A-Buffer (HUGHES et al., 2014).

Figure 1.1 – Example of MSAA treatment.



(a) Image without anti-aliasing. (b) Image with MSAA.

Source: Own authoring.

1.1 Objectives

The usual anti-aliasing methods improve the quality of the scene. However, they have significant drawbacks that hamper its use. Most anti-aliasing methods need to surpersample the image, at least specific regions, or store auxiliary buffer(s) (HUGHES et al., 2014), which implies high computing cost or high memory use. These drawbacks have a significant impact on performance, particularly on real-time applications.

This research proposes a post-filtering Anti-Aliasing method that smooths aliasing effects applying low-pass spatial filtering only in the regions where the distortion has a higher

tendency to occur. The algorithm identifies the object edges in the scenes through segmentation techniques. Moreover, rotation techniques are used to match the filter mask with the detected edge inclination, improving the smoothing quality. Regarding execution time, the method has complexity $O(n)$, being n the number of pixels in the image, with no additional rendering effort to produce a high-resolution image.

The **R**otat**e**d S**pa**tial F**i**lt**er**ing (REPAIR) Anti-Aliasing is a post-filtering anti-aliasing method that uses a rotated low-pass spatial filter to blur the edge regions where aliasing has a higher probability to occur. Like other Anti-Aliasing methods, REPAIR-AA aims to improve the visual perception of digital images by smoothing aliasing effects caused by unavoidable undersampling problems.

1.2 Research Methodology

A graphic scene displaying geometric shapes on a gradient background is built with OpenGL to perform the tests. The rendered image is taken from OpenGL's framebuffer since the REPAIR anti-aliasing is a post-filtering technique. It is processed and then replaced in the framebuffer to be displayed.

Blender (Blender Foundation, 2020a) is used to test the REPAIR method on more complex scenes, with detailed models and illumination. Blender is a free software that allows modeling 3D scenes easily, with a friendly user interface. The scenes are created using 3D models available in BlenderKit (BlenderKit, 2017), a repository of 3D models for blender. BlenderKit provides free and paid 3D models from several authors under "Royalty Free" and "CC0 - No Rights Reserved" licenses. Only free models were used in tests.

The performance is evaluated through the asymptotic analysis of execution time and space on Section 3.4. This approach predicts the necessary resources to execute the algorithm independently of hardware or compiler. Due to the scarcity of documentation of usual anti-aliasing methods, it is hard to compare the proposed method with others, since anti-aliasing are usually embedded in proprietary solutions.

1.3 Outline

This thesis is organized as outlined below:

- **Chapter 2** reviews the critical theory to understand the method development and assesses the related works that developed anti-aliasing techniques utilizing different approaches.
- **Chapter 3** describes the operation of the REPAIR anti-aliasing, its asymptotic analysis, and the algorithm.
- **Chapter 4** presents and evaluates the obtained results.

- **Chapter 5** presents the final considerations and proposes improvements for future works.

2 THEORY

This chapter presents a review of the essential theory necessary to understand the method development. Section 2.1 approaches the YIQ color system used to convert an image to grayscale. Section 2.2 approaches spatial filtering, a fundamental digital image processing technique. Section 2.3 approaches simple linear regression, a statistical model used to estimate values based on pairs of observations. Finally, Section 2.4 presents studies that developed or reviewed anti-aliasing solutions.

The proposed method aims to process a digital image with spatial filtering from digital image processing area and linear regression from statics. Spatial filtering is one of the main tools used in digital image processing field for a broad spectrum of applications. It is a versatile tool because it does not need to process images in the frequency domain. Regression analysis is a collection of statistical tools used to model relationships between two or more variables.

2.1 YIQ Color System

A color system provides a specification of a coordinate system and a subspace to represent colors in single points. It specifies standards to represents images. The choice of color system is oriented to the application or the hardware, such as monitors, printers, and TVs (GONZALEZ; WOODS, 2008).

The RGB color system is usually employed to represent digital images to be displayed on the screen. It is based on a Cartesian coordinate system, and the colors are represented by its primary components: red, green, and blue (GONZALEZ; WOODS, 2008).

The YIQ color system is used in NTSC color TV system. The Y component represents the luma information, I and Q components represent the chrominance information. Equation (2.1) converts an image represented in RGB to the YIQ color system.

$$\begin{aligned} Y &= 0.299 \cdot R + 0.5959 \cdot G + 0.2115 \cdot B \\ I &= 0.587 \cdot R - 0.2746 \cdot G - 0.5227 \cdot B \\ Q &= 0.114 \cdot R - 0.3213 \cdot G + 0.3112 \cdot B \end{aligned} \tag{2.1}$$

The Y component of the YIQ color system must be isolated to convert an RGB image to grayscale since it contains information about the intensity of the pixels. The YIQ grayscale produces good results because it takes advantage of human color-response characteristics, preserving the changes in the orange-blue range than in the purple-green range (MATHWORKS). The grayscale pixel of an RGB image is obtained replacing all channels of RGB system by the Y value: $R = Y$, $G = Y$, $B = Y$. Figure 2.1 shows an example of grayscale conversion.

Figure 2.1 – Example of grayscale conversion using YIQ color system.

(a) Color image. (b) Grayscale image.

Source: Own authoring.

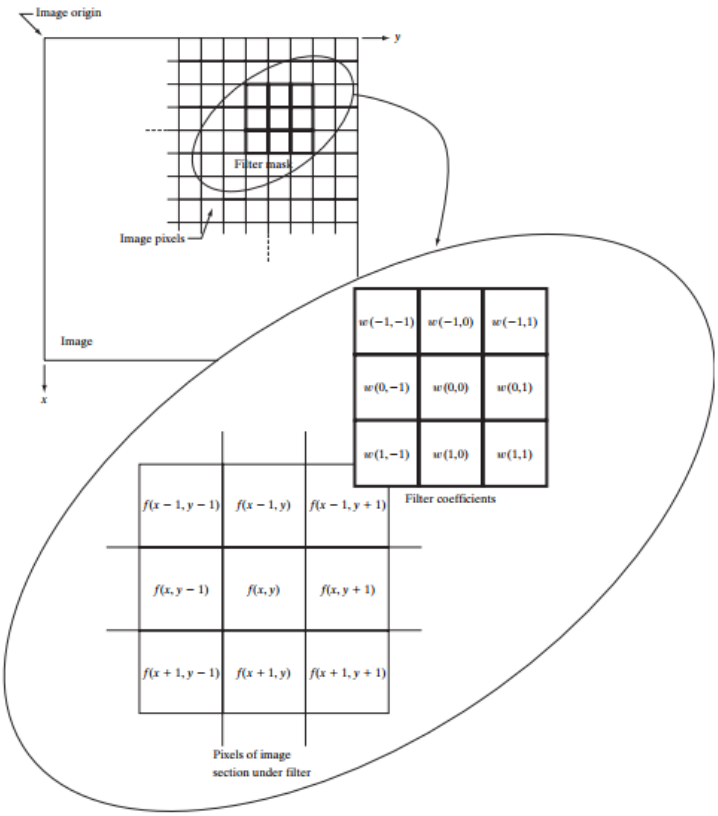
2.2 Spatial Filtering

Spatial filtering is one of the main tools used in digital image processing. It performs a predefined operation on a neighborhood of the image pixels. Filtering operation creates a new pixel, replacing the original value in the center of the neighborhood. A way to obtain a filtered image is using correlation. This process moves a filter mask over the entire image, one pixel at time (GONZALEZ; WOODS, 2008). The result of each step is the sum of the products. The correlation of a filter $m(w, h)$ of size $p \times q$ with an image $f(w, h)$ is given by:

$$m(w, h) \star f(w, h) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} m(w, h) f(w + s, h + t) \tag{2.2}$$

where $a = (p - 1)/2$ and $b = (q - 1)/2$ are the center of the filter mask. p and q are odd integers. Using correlation or convolution to perform spatial filtering is a matter of preference (GONZALEZ; WOODS, 2008). Figure 2.2 illustrates how a spatial filter works.

Figure 2.2 – Mechanics of spatial filtering using a 3×3 filter mask.



Source: MONTGOMERY; RUNGER (2014).

Spatial filters are divided into smoothing filters (low pass filter) and sharpening filters (high pass filter). Smoothing filters are used for blurring and noise reduction. Blurring is used to remove small details and bridging of small gaps in lines or curves. The *Gaussian blur* and *average filter* are examples of smoothing spatial filters. The principal objective of sharpening is to highlight color transitions intensities. The image sharpening process is used in electronic printing and medical imaging applications to industrial inspection and autonomous guidance in military systems. *Laplace*, *Roberts* and *Sobel gradients* are examples of sharpening spatial filter (GONZALEZ; WOODS, 2008).

The output of a smoothing filter is simply the average of the pixels in the neighborhood delimited by the filter mask. This average can be weighted or not and these filters can be called averaging filters. By replacing the value of every pixel in an image with the average of the intensity levels in its neighborhood, this process results in an image with reduced sharp intensity transitions, which have the side effect of blurring edges (GONZALEZ; WOODS, 2008).

In the *average filter*, each term of the mask $z = 1/n$, where $n = p \cdot q$ is the mask size. The sum of terms are $\sum_{i=1}^{n} z_i = 1$. The standard average of the pixels under a mask 3×3 is shown in Figure 2.3. An example of average filtering is shown in Figure 2.4.

Figure 2.3 – Average filter mask 3×3

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Source: Own authorship.

Figure 2.4 – Average filter.

(a) Original image. (b) Filtered image.

Source: Own authoring.

The Gaussian blur mask shown in Figure 2.5 is a commonly used smoothing filter. Its primary strategy is to weigh the terms maximizing the center point and minimizing the points according to its distance from the center, replicating the Gaussian distribution behavior (GONZALEZ; WOODS, 2008). This approach adds less blur to the image. Figure 2.6 shows the utilization of the *Gaussian blur filter*.

Figure 2.5 – Gaussian filter mask 5×5

$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{6}{256}$	$\frac{24}{256}$	$\frac{36}{256}$	$\frac{24}{256}$	$\frac{6}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$

Source: Own authorship.

Figure 2.6 – Gaussian blur filter.

(a) Original image. (b) Filtered image.

Source: Own authoring.

Since averaging is analogous to integration, sharpening can be accomplished by spatial differentiation. The strength of the response of a derivative operator is proportional to the degree of intensity discontinuity of the image. Image differentiation enhances edges and other discontinuities and deemphasizes areas with slowly varying intensities. Thus, sharpening filters will produce images with grayish edge lines and other discontinuities, all on a dark, featureless background (GONZALEZ; WOODS, 2008).

The *Laplacian operator* is implemented in the filter mask presented in Figure 2.7a. The diagonal directions can be incorporated, as shown in Figure 2.7b.

Figure 2.7 – (a) Filter mask implementing Laplace operators, (b) Filter mask implementing an extension of Laplace operators including diagonal terms.

0	1	0
1	-4	1
0	1	0

(a)

1	1	1
1	-8	1
1	1	1

(b)

Source: Own authorship

The masks presented in Figures 2.8 and 2.9 show the implementation of first-order derivatives filters. Sobel operators use a weight value of 2 in the center coefficient to achieve some smoothing by giving more importance to the center point (GONZALEZ; WOODS, 2008). Figures 2.8 and 2.9 present the *Roberts gradient* and *Sobel operators*, respectively.

Figure 2.8 – Filter masks implementing Roberts gradient.

-1	0	0
0	1	0
0	0	0

(a)

0	0	-1
0	1	0
0	0	0

(b)

Source: Own authorship.

Figure 2.9 – Filter masks implementing Sobel gradient.

-1	-2	-1
0	0	0
-1	-2	-1

(a)

-1	0	-1
-2	0	-2
-1	0	-1

(b)

Source: Own authorship.

Note that the coefficients in the filter masks sum to 0, indicating that they would give a response of 0 in an area of constant intensity. Moreover, masks of even sizes are awkward to

implement because they do not have a center of symmetry. The smallest filter masks of interest have a size 3×3 (GONZALEZ; WOODS, 2008). Figure 2.10 shows an example of high-pass filtering with the *Sobel operators*.

Figure 2.10 – Sobel gradient.

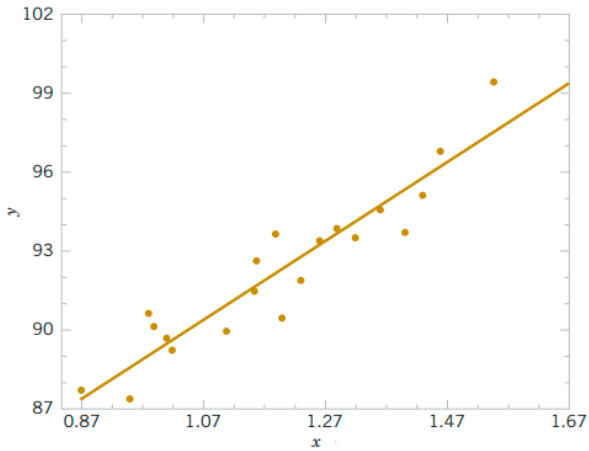
(a) Grayscale image. (b) Filtered image.

Source: Own authoring.

2.3 Simple Linear Regression

Simple linear regression is a model that explores linear dependency between variables to predict unknown or unobserved values (SOUZA, 2016). Its primary goal is to find a linear equation that represents this relationship based on pairs of observations (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n) (MONTGOMERY; RUNGER, 2014). The observed values are used to fit the equation's parameters to match the observed data. Figure 2.11 shows a scatter plot where the dots are the observed values, and the line is the fitted linear equation. The estimated values will follow the tendency of the observed values.

Figure 2.11 – Scatter plot demonstrating the relationship between two variable. The line represents the fitted linear function.



Source: MONTGOMERY; RUNGER (2014).

Equation (2.3) presents the linear relationship between x and y variables (MONTGOMERY; PECK; VINING, 2012). The single regressor variable x_i has a linear relationship with a response variable y_i. The variables x are also called independent variables and the variables y are called dependent variables (MONTGOMERY; RUNGER, 2014; MONTGOMERY; PECK; VINING, 2012).

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \tag{2.3}$$

where the parameters β_0 and β_1 are called regression coefficients. β_0 is the intercept and β_1 is the slope (MONTGOMERY; PECK; VINING, 2012).

Regression coefficients have unknown values and must be fitted to the observed data. The least-squares method is used to estimate them, according to Equations (2.4) and (2.5) (MONTGOMERY; PECK; VINING, 2012).

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{2.4}$$

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} \tag{2.5}$$

where $S_{xy} = \sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})$, $S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2$, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. The fitted simple linear regression model is given by Equation (2.6), which can be used to estimate the response for new values of the variable x.

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \tag{2.6}$$

For example, a rocket motor is manufactured by bonding an igniter propellant and a sustainer propellant together inside a metal housing. The shear strength of the bond between the two propellants is an important quality characteristic. It might have a relationship between the shear strength and the sustainer propellant's age (in weeks) (MONTGOMERY; PECK; VINING, 2012).

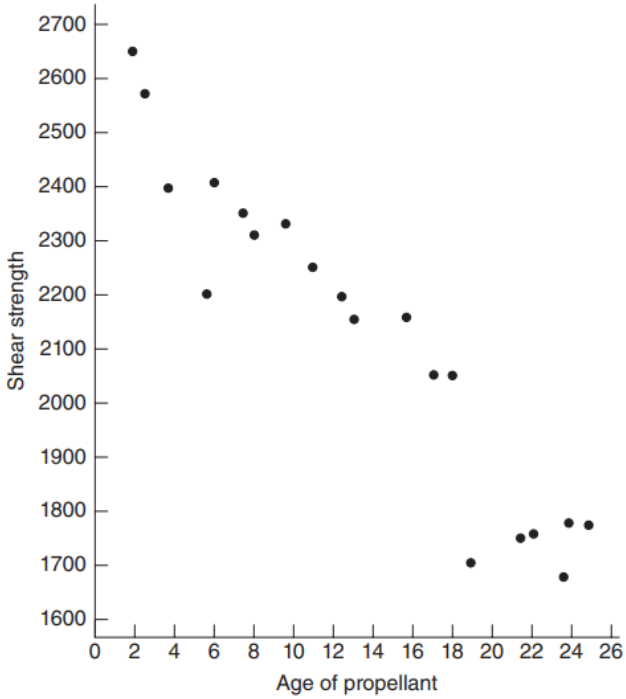
Thus, twenty pairs of observations have been collected to verify this behavior, and they are shown in Table 2.1. The shear strength is considered the dependent variable y, and the age of the propellant is the independent variable x to model this relationship with simple linear regression (MONTGOMERY; PECK; VINING, 2012). Figure 2.12 shows the scatter plot of the observations.

Table 2.1 – Data for rock propellant data.

i	$\mathbf{y_i}$	$\mathbf{x_i}$
1	2,158.70	15.50
2	1,678.15	23.75
3	2,316.00	8.00
4	2,061.30	17.00
5	2,207.50	5.50
6	1,708.30	19.00
7	1,784.70	24.00
8	2,575.00	2.50
9	2,357.90	7.50
10	2,256.70	11.00
11	2,165.20	13.00
12	2,399.55	3.75
13	1,779.80	25.00
14	2,336.75	9.75
15	1,765.30	22.00
16	2,053.50	18.00
17	2,414.40	6.00
18	2,200.50	12.50
19	2,654.20	2.00
20	1,753.70	21.50

Source: MONTGOMERY; PECK; VINING (2012).

Figure 2.12 – Scatter plot of shear strength versus propellant age.



Source: MONTGOMERY; PECK; VINING (2012).

With, $\bar{x} = 13.36$ and $\bar{y} = 2,131.36$

$$\begin{aligned} \hat{\beta}_1 &= \frac{S_{xy}}{S_{xx}} = \frac{-41,112.65}{1,106.56} = -37.15 \\ \hat{\beta}_0 &= 2,131.36 - (-37.15) \cdot 13.36 = 2,627.82 \end{aligned} \tag{2.7}$$

The least-squares fit is

$$\hat{y} = 2,627.82 - 37.15x \tag{2.8}$$

The slope -37.15 is the average weekly decrease in propellant shear strength due to the age of the propellant. The intercept $2,627.82$ represents the shear strength in a new batch of propellant (MONTGOMERY; PECK; VINING, 2012).

2.4 Related Work

Several anti-aliasing solutions are found in the literature. Among them are the splatting technique, self-similar crowd-based algorithms, approximation method, edge-directed adaptive filters, oversampling techniques, and machine learning methods. In most cases, these solutions seek the best visual quality of images with less memory consumption and less processing time.

Next, in chronological order of publication, the solutions related to the method proposed in this work will be presented, as noting this theme continues relevant for decades.

An anti-aliasing technique for splatting is proposed in 1997 by (Swan II et al.), which delivers high-quality splatted images. Splatting is a popular rendering algorithm that does not correctly render cases where the volume sampling rate is higher than the image sampling rate, resulting in potentially severe spatial and temporal aliasing artifacts. Previous splatting algorithms do not have a mechanism for avoiding aliasing artifacts. The proposed mechanism has the potential for very efficient hardware implementation.

The anti-aliasing algorithm suggested by (KVETNY; KOSTROVA; BOGATCH, 2001) in 2001 uses interpolation based on self-similar multitudes to remove the jaggies. This approach does not change the brightness or the colors of a magnified image. This anti-aliasing algorithm was implemented in the special software for low vision people — the L&H Magnifier. The preliminary tests confirmed that the developed technology improves the quality of zoomed images much better than the standard algorithms, but it needs a large number of computer operations. So, using the anti-aliasing algorithm based on self-similar multitudes is reasonable when the magnification level is four and higher.

The algorithm of shadowmap and its basic principles are studied in 2007 by (XIAO-LIANGI et al., 2007), which explains the cause of self-shadow and aliasing mathematically. The paper presents a multi-sampling algorithm to smooth shadow and reduce aliasing by increasing the precision of depth information and using z-bias. The algorithm is implemented by DirectX3D. Experimental results show the feasibility of the algorithm.

In papers from 2009, it was observed that post-processing anti-aliasing algorithms were widely used for real-time rendering because of their simplicity, performance, and suitability for deferred shading. Fast approximate anti-aliasing (FXAA) is the fastest method among them. Thus, many games support FXAA to get anti-aliased images. However, FXAA can easily lose texture details and text sharpness due to its excessive blurring (NAH et al., 2009).

In the same year, (IOURCHA; YANG; POMIANOWSKI, 2009) describes an adaptive anti-aliasing (AA) filter for real-time rendering on the GPU, as such device provides direct access to multisample anti-aliasing (MSAA) rendering data. The intelligent reconstruction filter uses the existing pixel subsample values calculated using programmable GPU shader units. The improved quality is achieved by using information from neighboring pixel samples to calculate the leading edge gradient approximation and the final pixel color.

Motion-based object detection is very used nowadays, and several implementations use linear shift-invariant (LSI) filters. One of the most important factors falsifying object detection results obtained using linear LS) is aliasing. Then, in (SCHAULAND; VELTEN; KUMMERT, 2010), an innovative, in 2010, anti-aliasing approach for motion-based object detection using LSI is presented. The approach's applicability is demonstrated using scenes recorded by a camera installed in a blind-spot warning system.

The surface based anti-aliasing (SBAA) is a new approach proposed by (SALVI; VIDIMCE, 2012). SBAA is a pré-processing anti-aliasing technique used in conjunction with deferred renderers to resolve visibility of sub-pixel features, minimizing spatial and temporal artifacts. It is a real-time anti-aliasing for deferred renderers that improves the performance and lowers the memory requirements for anti-aliasing methods that sample sub-pixel visibility.

In 2013 (BARRINGER; AKENINE-MOLLER, 2013) presented a novel algorithm that uses shared memory between the GPU and the CPU to solve the edge aliasing problem. The system renders the scene as usual on the GPU with one sample per pixel. At the same time, it executes asynchronously on the CPU. First, a sparse set of important pixels is created. Then, the CPU runs a sparse rasterizer and fragment shader, directly accessing shared resources. The method can render a scene with shadow mapping and adaptive anti-aliasing with 16 samples per important pixel faster than the GPU with 8 samples per pixel using multi-sampling anti-aliasing.

In the same year, (CRASSIN et al., 2013) presented Aggregate G-Buffer Anti-Aliasing (AGAA), a technique for the anti-aliased deferred rendering of complex geometry using modern graphics hardware. AGAA uses the rasterization pipeline to generate a compact, pre-filtered geometric representation inside each pixel. Then, it shades this at a fixed rate, independent of geometric complexity. By decoupling shading rate from geometric sampling rate, the algorithm reduces the storage and bandwidth costs of a geometry buffer, allowing scaling to high visibility sampling rates for anti-aliasing. AGAA with 2 aggregate surfaces per pixel generates results comparable to 8x MSAA, requiring 30% less memory.

Accumulative anti-aliasing (ACAA) is a simple modification of forward-rendered multi-sample anti-aliasing (MSAA). ACAA stores multiple depth samples, computed by a depth-only pre-pass, but stores only one color sample per pixel, which is used to accumulate final color as the sum of shaded fragment colors weighted by visibility. ACAA makes higher sample rates practical, improving image quality. It produces the same image quality but consumes half as much multi-sample framebuffer memory and reduces both render time and off-chip bandwidth by 20% to 30% (ENDERTON et al., 2015).

The paper (WANG et al., 2015) presents Decoupled Coverage Anti-Aliasing (DCAA), which improves upon MSAA by further decoupling coverage from visibility for high-quality geometric anti-aliasing. Since all fragments at a pixel can be consolidated into a small set of visible surfaces, each consolidated surface is represented with a 64-bit binary mask for coverage and a single decoupled depth value, reducing the overhead for high-quality anti-aliasing. Surface merging heuristics and resolution mechanisms are used to manage the decoupled depth and coverage samples. DCAA runs in real-time on current graphics hardware and significantly reduces geometric aliasing with less memory overhead than 8×MSAA.

In 2016 (RAJARAPOLLU; MANKAR, 2016) designed and analyzed the anti-aliasing methods for the rendering of graphical entities. The performance of super-sampling, multi-sample, and morphological anti-aliasing methods were compared. It has been observed that super-sampling

and multi-sample give better results, although they suffer from the extra burden of computational resources. The morphological anti-aliasing method has a scope to improve the performance, yet it can yield real-time performance that can be achieved using parallelism.

An anti-aliasing algorithm for mobile vector graphics has been proposed by (LEE et al., 2016). It aims to reduce sample counts by selective multi-sampling only for the pixels that meet the image's contour. The pixel block algorithm reduces the cost of the intersection test, processing adjacent pixels together. Hierarchical sampling is used for more optimization. The optimized S-MSAA (Selective Multi-Sample Anti-Aliasing) is 29.4 times faster than the original 16× MSAA.

An anti-aliasing algorithm based on saliency map for virtual reality applications has been introduced by (SUNG; CHOI, 2017). It first renders the whole scene into a single texture image and feeds it into saliency map construction. The resulting saliency map is then input to the second rendering step with the original texture image. The second rendering step performs the anti-aliasing algorithm selectively based on the value of saliency map. The user study shows that participants do not distinguish between full anti-aliasing and selective anti-aliasing based on saliency map. The rendering time has a 5-10% performance increase if the selective anti-aliasing is used (SUNG; CHOI, 2017).

To achieve the fast and accurate anti-aliasing line rendering, using the modified DDA and Wu straight line generation algorithm, a two-pixel anti-aliasing point-by-point rendering algorithm for straight lines using integer arithmetic was proposed by (LIU; LI, 2018). The algorithm calculates the coordinate position of the next pixel and its brightness level based on the midpoint error value. Theoretical analysis and experiments show that the new algorithm is simple and has good execution efficiency and anti-aliasing effect.

The paper (ZHAO et al., 2019) reviews the SMORE algorithm and then demonstrates its performance in four applications to demonstrate its potential in research and clinical scenarios. SMORE is a deep learning approach that carries anti-aliasing and super-resolution on Magnetic resonance (MR) images using no external atlas or exemplars. It is shown to improve the visualization of brain white matter lesions, the visualization of scarring in cardiac left ventricular remodeling after myocardial infarction, multi-view images of the tongue, and finally, it improves performance in parcellation of the brain ventricular system. Both visual and selected quantitative metrics of resolution enhancement are demonstrated.

A direct rendering method for trimmed Non-Uniform Rational B-Splines (NURBS) models based on their parametric description is presented by (SCHOLLMEYER; FROEHLICH, 2019). NURBS are a common model representation for export, simulation, and visualization in Computer-Aided Design (CAD). The proposed approach builds a trimming method and a three-pass pipeline which allow for a sub-pixel precise visualization. The rendering pipeline bypasses the tessellation limitations of current hardware using a feedback mechanism. The proposed method scales well with many trim curves and estimates the trimmed surface's footprint

in screen space, allowing for anti-aliasing with minimal performance overhead. Fragments with trimmed edges are routed into a designated off-screen buffer for subsequent blending with background faces. The curve coverage estimation used for anti-aliasing provides an efficient trade-off between quality and performance compared to multisampling or screen-space anti-aliasing approaches.

In 2020, (LUO; ZHANG, 2020) proposes an anti-aliasing algorithm of the #-filter anti-aliasing based on sub-pixel continuous edges. It can solve the geometry edges aliasing and the flicker problem in deferred shading. First, the geometry scene with multi-sampling anti-aliasing (MSAA) is rendered to a G-Buffer designed elaborately. Second, the geometry edges are detected on the sub-pixel-level. It mainly takes advantage of the Chebyshev inequality to determine the edges from the probability statistic and the view frustum location. Third, the continuous geometry edges are reconstructed by a #-filter method. Finally, the edge pixels are shaded adaptively. The implementation demonstrates that the algorithm is efficient and scalable for generating high-quality anti-aliasing geometry and reducing shading calculation overhead.

2.4.1 Real Time Anti-Aliasing

In 2011, the course (JIMENEZ et al., 2011) presented how industry and academia have been exploring alternative anti-aliasing approaches, sharing concepts and ideas. The main goal of the course is to establish a conceptual link between them, identifying novelties and differences. The course includes an overview of both research and industry filter-based anti-aliasing techniques in games for all modern platforms.

For over a decade, supersample anti-aliasing (SSAA) and multisample anti-aliasing (MSAA) have been the gold-standard anti-aliasing solutions in games. However, these techniques are not well suited for deferred shading or fixed environments like the seventh generation of consoles. The original morphological anti-aliasing (MLAA) method led to an explosion of real-time anti-aliasing techniques that rival MSAA (JIMENEZ et al., 2011).

Morphological Anti-Aliasing (MLAA) algorithm belongs to a family of data-dependent filters allowing efficient anti-aliasing at a post-processing step. The algorithm infers sub-pixel coverage by estimating plausible silhouettes from a collection of axis-aligned separation lines that fall between perceptually different pixels (JIMENEZ et al., 2011).

The main idea of MSAA is that in areas of low edge curvature, samples outside of a pixel can be used on the isolines of the image to reconstruct a pixel. The basic idea is to use samples inside and outside of the pixel. The weight of the sample is determined by how much the isoline intersects with the pixel (JIMENEZ et al., 2011).

The MLAA version developed by (JIMENEZ et al., 2011) detects borders (using color, depth, normals, or instance id's information) and then finds specific patterns. Anti-aliasing is achieved by blending pixels in the borders intelligently. Practical Morphological Anti-Aliasing

(JIMENEZ et al., 2011) presents an optimized GPU adaptation.

Hybrid Morphological Anti-Aliasing presents an implementation of MLAA for the Xbox 360 where the edge detection and filtering states of the algorithm are performed on a GPU, and the blend weight calculations are performed on CPUs (JIMENEZ et al., 2011).

Subpixel Reconstruction Anti-Aliasing (SRAA) treats aliasing by taking sub-pixel samples to find edges while keeping the shading at 1 sample per pixel to guarantee good coherence and minimal changes to the existing pipeline. In particular, SRAA requires an edge detector similar to Jiminez's MLAA. It determines sample similarity to reconstruct the color of each sample. Then, the samples get filtered using a box filter and can be sent to the post-processing pipeline (JIMENEZ et al., 2011).

The Fast approXimate Anti-Aliasing (FXAA) algorithm applies 2 or 4-tap variable-length box filter 90 degrees to luma gradient, with an adjustment on filter length and direction to help remove sub-pixel aliasing. It works as one full-screen pixel shader pass, taking color as input and writing color as output (JIMENEZ et al., 2011).

Distance-to-edge AA (DEAA) simulates anti-aliasing by selective blurring. Different from MLAA, the distance-to-edge values are calculated with sub-pixel precision. The post-process blur can simulate subpixel effects that are not possible by simple inspection of a frame-buffer. However, DEAA is unable to provide anti-aliasing in several situations where MLAA can (JIMENEZ et al., 2011).

The geometric post-process anti-aliasing (GPAA) is proposed as an alternative approach to address these problems. Geometric edge information is stored to a fullscreen render target during the main rendering pass. In the end, the buffer is analyzed and resolved in a fullscreen pass, similar in concept to how traditional MSAA works (JIMENEZ et al., 2011).

Directionally Localized Anti-Aliasing (DLAA) is a solution designed with simplicity. It makes GPU and CPU friendly and allows efficient implementations on modern gaming consoles such as the PlayStation 3 and Xbox 360. It is temporally stable and very effective, offering high quality to performance ratio (JIMENEZ et al., 2011).

2.4.2 Conclusion

This subsection presented several related works that developed or reviewed anti-aliasing methods. Most of them used the pre-filtering approach, where the method runs within the render process. Most of them have high execution time or high memory use as drawbacks, impacting real-time applications performance considerably.

It has become evident that different approaches can be used to treat the aliasing problem. While the first related works presented innovative methods, the course developed by (JIMENEZ et al., 2011) makes a review of the main methods utilized at retail. They are SSAA, MSAA, MLAA, SRAA, FXAA, DEAA, GPAA, and DLAA.

None of the related works utilized any approach similar to the proposed in this thesis. Although, the absence of similar studies denotes its innovative potential. Moreover, there are no established metrics for quality or performance. In most studies, the results are displayed to the reader's eye assessment, assigning them the task of examining the results subjectively. For performance measurement, execution time and memory consumption are the primary metrics used. Although, with these metrics, the results change according to the hardware used.

3 REPAIR: AN ANTI-ALIASING METHOD

This chapter describes the operation of REPAIR anti-aliasing and its asymptotic analysis. Section 3.1 approaches the steps related to digital image processing. The steps related to statistics are approached in Section 3.2. Section 3.3 shows the rotation procedure of the spatial filtering mask. Section 3.4 accomplishes the asymptotic analysis of the algorithm, and Section 3.5 presents the pseudo-code.

The aliasing problem occurs during the raster process of digital images when colors need to be placed in the discrete regions of the pixels. Aliasing has no definitive solution. Usually, it is reduced by supersampling the image. However, it can have a high computing cost.

Therefore, this work purpose the **R**otat**e**d S**pa**tial F**i**lt**er**ing (REPAIR) Anti-Aliasing method aims to smooth aliasing effects without the need for a supersampled image or interfering in the raster process. It is a post-filtering technique and must treat with low computational complexity.

The REPAIR anti-aliasing method is divided into five steps: grayscale conversion, sharpening filter with Sobel gradient, threshold cleaning, slope angle estimation with simple linear regression, and rotated smoothing filter. The steps are outlined in the following section, sequentially and divided by knowledge area.

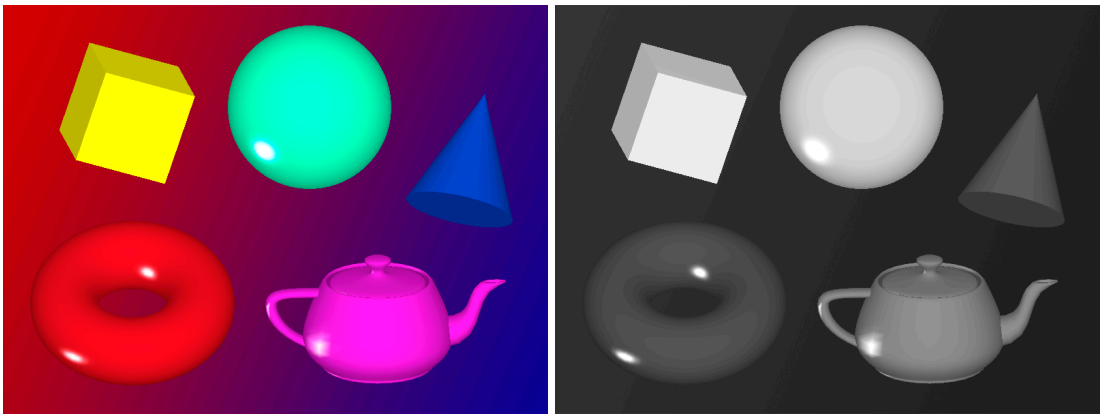
First, to perform the REPAIR AA, it is necessary to recognize the edges, where aliasing usually appears. After that, the tangent angle of each edge pixel is computed. These angles are used to rotate the filter mask, used by correlation. Finally, the smoothing filter is applied to the selected regions, blurring the edges.

The method utilizes knowledge from digital image processing and statics fields. Section 3.1 approaches the steps related to the digital image processing field; section 3.2 approaches the steps related to the statistics field; section 3.3 explains the process to rotate the smoothing spatial filter; section 3.5 shows the pseudo-code of REPAIR AA and the asymptotic analysis.

3.1 Image Processing

REPAIR method aims to treat aliasing selectively in regions with abrupt transitions of colors. These regions are identified through the pixel intensity obtained by grayscale conversion. For this, the Y channel of the YIQ color system can be used, as presented in Equation (2.1). Figure 3.1 shows the grayscale conversion result.

Figure 3.1 – Grayscale conversion example on an image containing random geometric shapes created with OpenGL.

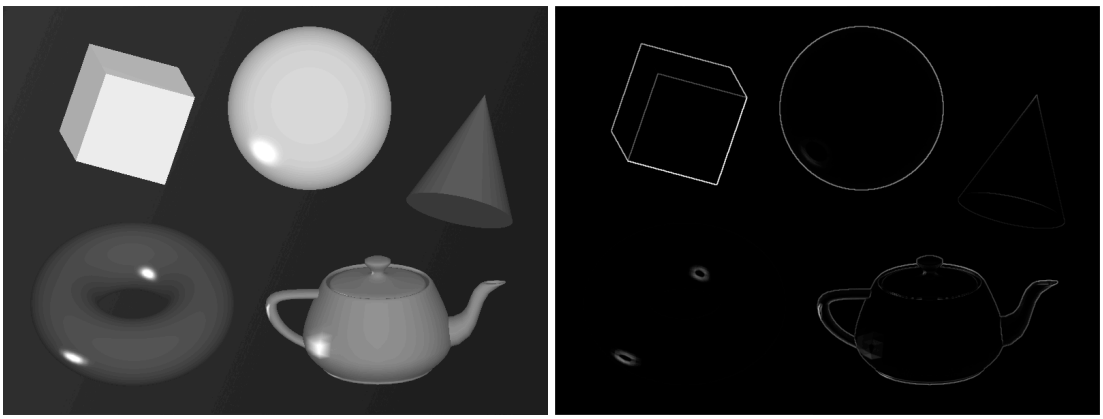


(a) Color image. (b) Grayscale image.

Source: Own authoring

In the second step, a sharpening spatial filter is applied to the grayscale image. This type of spatial filter tends to highlight the edge regions of the objects and erase the background, resulting in a black image with grayish lines. The mask of Sobel gradient that has been utilized in this step is showed in Figure 2.9. Then, the result of this process is illustrated in Figure 3.2.

Figure 3.2 – Sharpening filtering example with Sobel gradient applied on the grayscale image.



(a) Grayscale image. (b) Filtered image with Sobel gradient.

Source: Own authoring

3.2 Statistics

The filter applied in the second step highlights the borders. However, it can also highlight noise and unimportant regions without aliasing, resulting in a polluted image. In the third step, a threshold is applied to eliminate a part of the greyish pixels to clean the filtered image.

The threshold T is based on the mean μ of the nonblack pixels values $z_{i,j}$. A weight κ multiplies the population variance σ to adjust the threshold. The value of κ can be positive or negative, determining the number of pixels that must be eliminated. Low values of κ tend to preserve the image, while high values tend to erase all pixels. Finally, the pixels with intensity less than T are removed from the image, remaining those with higher intensities.

Being n the number of nonblack pixel, the threshold T of a $w \times h$ size image is given by

$$T = \mu + \kappa\sigma \tag{3.1}$$

with

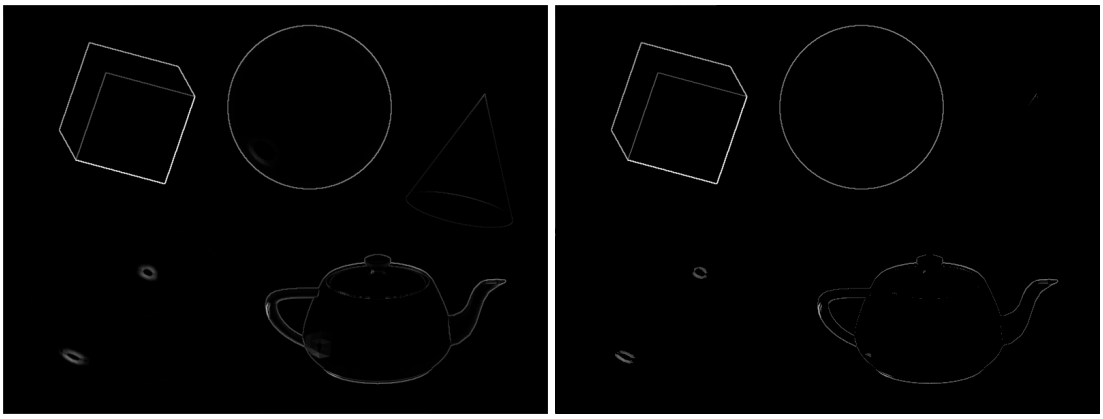
$$\mu = \frac{1}{n}\sum_{i=1}^{w}\sum_{j=1}^{h} z_{i,j}$$

and

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{w}\sum_{j=1}^{h}(z_{i,j} - \mu)^2.$$

The result from the third step is an image with thinner lines and less noise but still preserving the edge regions. The remaining nonblack pixels are the selected regions where the smoothing filter must be applied. It is shown in Figure 3.3.

Figure 3.3 – Threshold applied on the filtered image with sharpening filter.

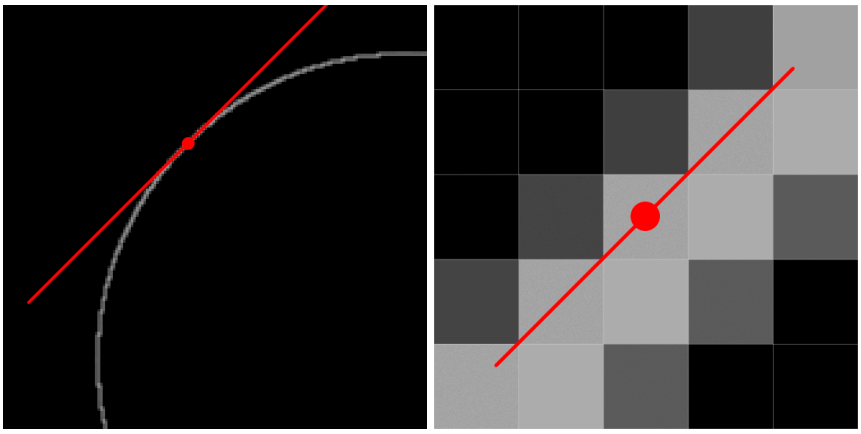


(a) Filtered image with Sobel gradient. (b) Image with threshold applied.

Source: Own authoring

The direction of each edge pixel must be known to rotate the spatial filter that will smooth the aliasing. The fourth step utilizes simple linear regression to estimate the tangent angle of each edge pixel. Figure 3.4 illustrates the direction of the center pixel of a 5×5 neighborhood, highlighted as a red line.

Figure 3.4 – The red line represents the estimated slope angle of the pixel marked with the red dot. The estimate is based on the square patch shown in (b).

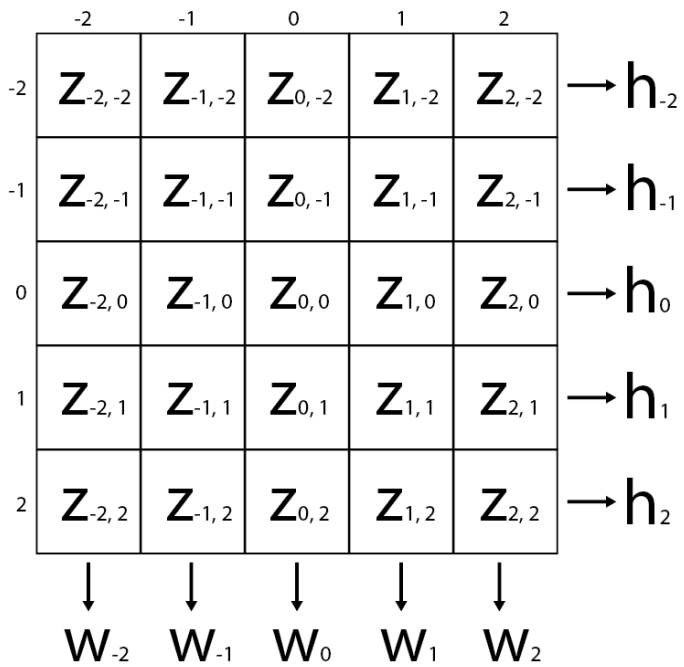


(a) Direction of the marked pixel. (b) Patch of 5×5 size

Source: Own authoring

The neighborhood around each selected pixel is assessed to estimate the slope. Figure 3.5 shows a 5×5 patch with pixels values $z_{i,j}$. Figure 3.5 show a generic neighborhood, where w_i and h_i are the means weighted by position of the columns and lines, respectively.

Figure 3.5 – Patch of pixels $z_{i,j}$ used to estimate its direction. w_i and h_i are the means from each column and line, respectively.



Source: Own authoring.

Being $a = (n-1)/2$ the mask center, the Equation (3.2) computes the w_i and h_i values.

$$
\begin{aligned}
w_i &= \frac{\sum_{j=-a}^{a} z_{i,j} \cdot j}{\sum_{j=-a}^{a} z_{i,j}} \\
h_i &= \frac{\sum_{j=-a}^{a} z_{j,i} \cdot j}{\sum_{j=-a}^{a} z_{j,i}}
\end{aligned}
\tag{3.2}
$$

if $\sum_{j=-a}^{a} z_{i,j}$, then $w_i = 0$, and if $\sum_{j=-a}^{a} z_{j,i}$, then $h_i = 0$.

Then, two simple linear regression models are built to fit w_i and h_i means. One linear regression will fit the vertical distribution of pixels, while the other will fit the horizontal distribution of pixels. Since the problem is is too complex to handle, the estimates are used to facilitate the computation.

Because the intensity values must be estimated from the patch coordinates, the means w_i and h_i the response variables y, and the coordinates being the regressor variables x_i. Table 3.1 arranges the pairs of data to build the linear regression models.

Table 3.1 – Pairs of data to the linear regression model construction.

i	$\mathbf{y_i}$	$\mathbf{x_i}$
1	h_{-2}	-2
2	h_{-1}	-1
3	h_0	0
4	h_1	1
5	h_2	2

i	$\mathbf{y_i}$	$\mathbf{x_i}$
1	w_{-2}	-2
2	w_{-1}	-1
3	w_0	0
2	w_1	1
3	w_2	2

Source: Own authoring.

It is common to have pixels replication in the neighborhood since the application works on images. If the intensity values are similar, then the w_i and h_i values shall be too. Equation (3.3) presents the two linear regression models.

$$
\begin{aligned}
w_i &= \alpha_0 + \alpha_1 x_i + \varepsilon_i^w \\
h_i &= \beta_0 + \beta_1 x_i + \varepsilon_i^h
\end{aligned}
\tag{3.3}
$$

being α_0 and β_0 the intercepts, α_1 and β_1 are the slope. w_i and h_1 are the response variables and x_i is the regressor variable. ε_i^w and ε_i^h are the errors. The function is parameterized by x_i, which is the same in both equations.

Because α_1 and β_1 slopes represents the two separated axes, the equations w_i and h_i are parameterized by x_i to find the resulting slope that represents the pixel direction, illustrated by the red line in Figure 3.4. Because the edges are diffuse, the resulting function must lean to one of the axes. Therefore, the model must match the edge direction.

Thus,

$$\begin{aligned} w_i &= \alpha_0 + \alpha_1 x_i + \varepsilon_i^w \\ x_i &= \frac{w_i - \alpha_0 - \varepsilon_i^w}{\alpha_1} \end{aligned} \tag{3.4}$$

Replacing on h_i

$$\begin{aligned} h_i &= \beta_0 + \beta_1 x_i + \varepsilon_i^h \\ &= \beta_0 + \beta_1 \frac{w_i - \alpha_0 - \varepsilon_i^w}{\alpha_1} + \varepsilon_i^h \\ &= \beta_0 + \frac{\beta_1}{\alpha_1} w_i - \frac{\beta_1}{\alpha 1} \alpha_0 - \frac{\beta_1}{\alpha_1} \varepsilon_i^w + \varepsilon_i^h \end{aligned} \tag{3.5}$$

where h_i is the regressor variable, $\frac{\beta_1}{\alpha_1}$ is the resulting slope angle, β_0 is constant, and the errors $\varepsilon_i^w + \varepsilon_i^h$ are discarded. The slope angle represents the estimated tangent angle in the region assessed. Therefore,

$$\tan\theta = \frac{\beta_1}{\alpha 1} \tag{3.6}$$

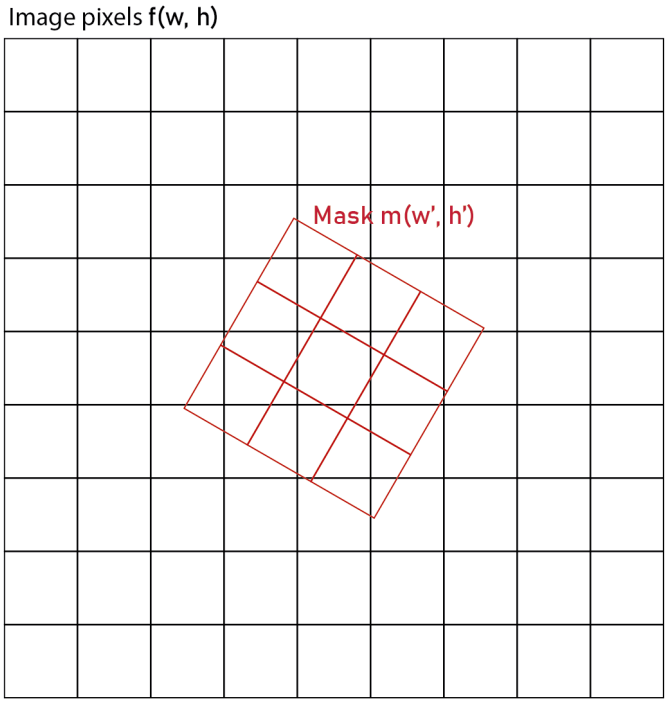
Finally, the arctan θ is the slope angle that indicates the direction of the edges. Such angle must be utilized in the spatial filter rotation.

3.3 Mask Rotation

In the fifth step, the average filter is utilized to smooth the edges of the original color image. This low-pass filter is appropriate because it reduces abrupt transitions and diminishes details blurring the region where it is executed.

According to (GONZALEZ; WOODS, 2008), the average filter is an isotropic filter that applies equally well independent of the direction. However, due to spatial filtering characteristics, is it isotropic in limited angles, steeping in $45°$ angles. Thus, this filter will be rotated to match each edge pixel direction to improve the smoothing effect quality. Figure 3.6 shows the rotation mechanics, where the big matrix represents the image pixels, and the red matrix represents the filter terms. Each term will hit pieces of different pixels.

Figure 3.6 – Rotation mechanics with a filter mask 3×3



Source: Own authorship.

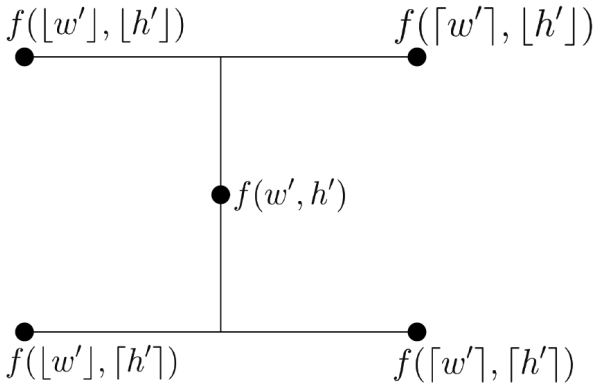
After rotating the mask, its pixels have new coordinates. Equation (3.7) is used to compute the new coordinates of the rotated mask.

$$\begin{aligned} w' &= w \cdot \cos\theta - h \cdot \sin\theta \\ h' &= w \cdot \sin\theta + h \cdot \cos\theta \end{aligned} \tag{3.7}$$

with $w', h' \in \mathbb{R}$. The rotation will result in real numbers coordinates. Therefore, a modification of bilinear interpolation estimates the color of $f(w', h')$ since w' and h' are real numbers and do not fit in the discrete representation of pixels coordinates.

To interpolate the point $f(w', h')$, its four nearest pixels are needed. These pixels are obtained using the floor and ceiling values of w' and h', which round them to match integer pixels coordinates as illustrated by Figure 3.7.

Figure 3.7 – Use of bilinear interpolation to estimate the center dot intensity based on the four nearest pixels.



Source: Own authorship.

Calling the four nearest pixels a, b, c and d, its values are given by Equation (3.8)

$$\begin{aligned} a &= f(\lfloor w' \rfloor, \lfloor h' \rfloor) \\ b &= f(\lceil w' \rceil, \lfloor h' \rfloor) \\ c &= f(\lfloor w' \rfloor, \lceil h' \rceil) \\ d &= f(\lceil w' \rceil, \lceil h' \rceil) \end{aligned} \tag{3.8}$$

Equation (3.9) computes the value of $f(w', h')$.

$$\begin{aligned} f_1 &= a + (h' - \lfloor h' \rfloor) \cdot (b - a) \\ f_2 &= b + (h' - \lfloor h' \rfloor) \cdot (d - c) \\ f(w', h') &= f_1 + (w' - \lfloor w' \rfloor) \cdot (f_2 - f_1) \end{aligned} \tag{3.9}$$

For example, to estimate the color of a pixel $f(10.3, 30, 8)$, the nearest pixel coordinates are given by the following floor and ceiling values

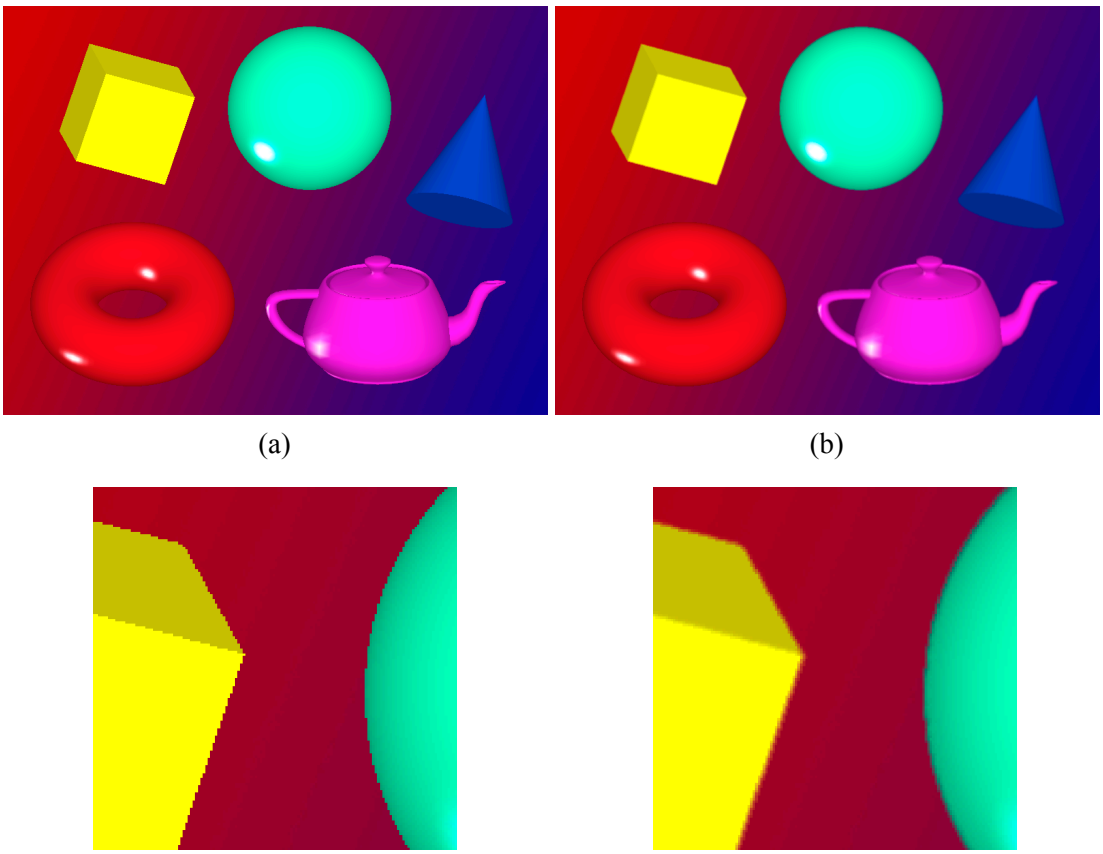
$$\begin{aligned} \lfloor w' \rfloor &= \lfloor 10.3 \rfloor = 10 \\ \lceil w' \rceil &= \lceil 10.3 \rceil = 11 \\ \lfloor h' \rfloor &= \lfloor 30.8 \rfloor = 30 \\ \lceil w' \rceil &= \lceil 30.8 \rceil = 31 \end{aligned} \tag{3.10}$$

Therefore,

$$\begin{aligned} a &= f(10, 30) \\ b &= f(11, 30) \\ c &= f(10, 31) \\ d &= f(11, 31). \end{aligned} \tag{3.11}$$

The bilinear interpolation will be executed one time for each pixel in the neighborhood, and this process will be repeated for each selected edge pixel according to correlation equation (2.2). The pixel value is the interpolated $f(w', h')$, and the mask is the average mask in Figure 2.5. The new pixels result in the smoothed image with the rotated spatial filter, finishing the REPAIR AA treatment. Figure 3.8 shows the smoothing result.

Figure 3.8 – (a) Original image (b) Image treated with REPAIR anti-aliasing (c) Zoom in the original image (d) Zoom in the treated image.

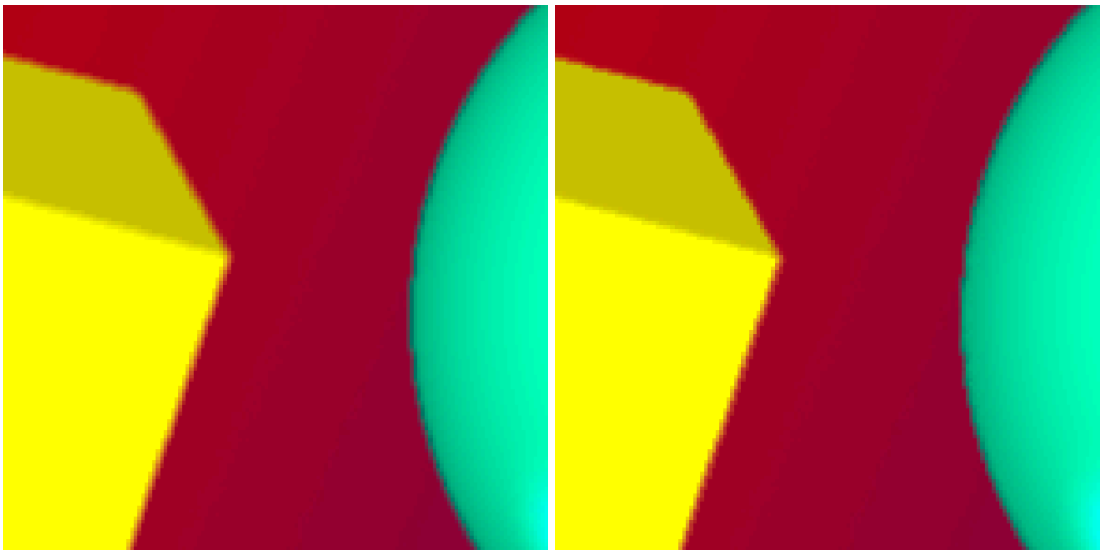


(a) (b) (c) (d)

Source: Own authoring

The rotation process smooths the edges in an adequate direction, blurring more accurately and preserving the scene details. Figure 3.9 shows the difference between the rotated spatial filtering and standard spatial filtering. It is visible that the rotated filter smooths the edges with less blurring, reducing the aliasing but still keeping the edges well marked.

Figure 3.9 – Comparison between two images treated with non rotated and rotated spatial filtering.



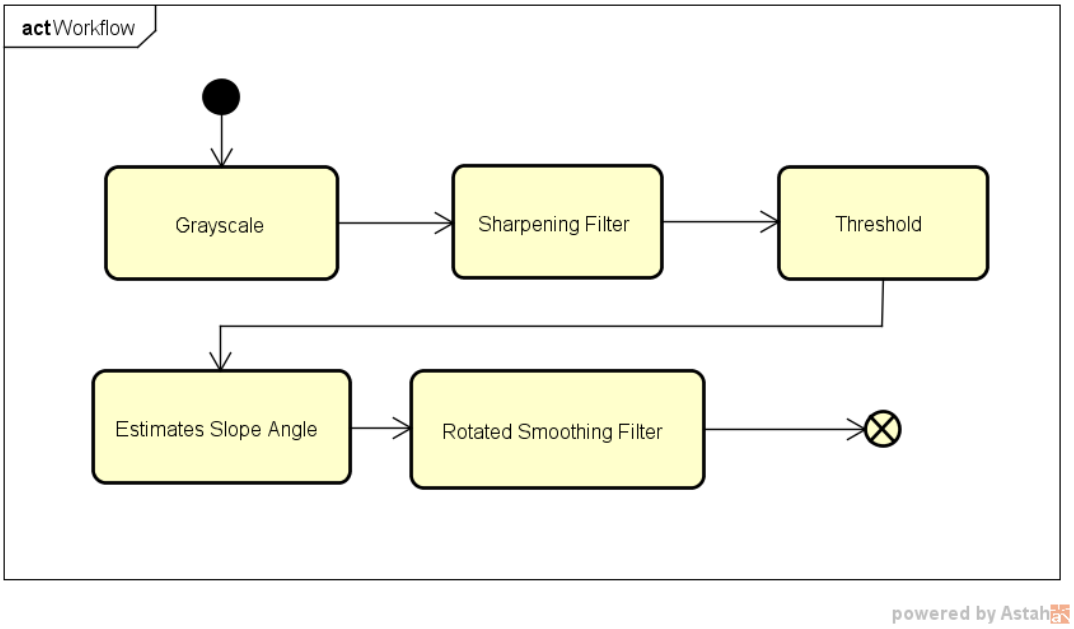
(a) Standard smoothing filter. (b) Rotated smoothing filter.

Source: Own authoring

3.4 Asymptotic Analysis

The REPAIR AA pseudo-code describes the method operation. It is shown in the Algorithm 1. The grayscale step is executed at line 1, the sharpening filter at line 2, and the threshold step at line 3. Between lines 15 and 29, there are the main loops, which walk through all image pixels. The slope angle step is at line 25, and the rotated smoothing filter is at line 26. The diagram in Figure 3.10 summarizes the algorithm steps.

Figure 3.10 – Algorithm's workflow.



Source: Own authorship.

Assuming that w and h are, respectively, the width and height of the source image and that $m \times m$ is the mask size of the spatial filter, $n = w \cdot h$ is the number of the source image pixels, while m^2 is the number of filter terms.

In the first step, the target image is converted to grayscale. For that, Equation (2.1) is computed for each pixel. Since Equation (2.1) executes in constant time $O(1)$, and it will be executed n times, the grayscale function execution time is $O(n)$. The memory use is also $O(n)$, because a new image is stored.

In the second step, a Sobel gradient is applied to highlight the edges of the scene objects. Because Sobel gradient uses two filter masks, the correlation (Equation (2.2)) is executed two times for each pixel, adding the results in the end. Since the Equation (2.2) executes in time $O(m^2)$, and it executed for all n image pixels, the resulting complexity is time $O(2nm^2))$ and memory $O(n)$.

In the third step, a threshold is applied to reduce noise e discard non-relevant regions. It is described in Algorithm 2. In lines 2 and 3, the main loops walk in all image pixels, resulting in time complexity $O(n)$ and space complexity $O(1)$.

The loops in lines 4 and 5 apply the threshold to segment the image, inspecting all pixels. Thus, it results in time $O(n)$ and memory $O(1)$, since no data is stored. The main loops in lines 15 and 16 also walk through all n pixels, and the inner loops inspect the neighborhood of size $m \times m$. Inside the main loop, the slope angle and the bilinear interpolation are computed for each selected pixel.

In the fourth step, the algorithm estimates the slope angle of each pixel from the edge regions using simple linear regression. Its pseudo-code is described in Algorithm 3. This function is applied to the selected patches $m \times m$ to find θ slope angle, and his main loop in line 3 walks through the patch. All image pixels will be selected as edge pixels in the worst-case scenario, resulting in time complexity $O(nm^2)$ and space complexity $O(nm^2)$.

With the edges pixels found and their slope angle estimated, the low-pass spatial filter is rotated and applied to these regions at the color image. Bilinear interpolation in Algorithm 4 is used to estimate the values of the low pass filter of the rotated mask (for each color channel). This function will be executed in each selected pixel to interpolate it. The worst-case scenario takes time $O(nm^2)$ and space $O(n)$.

Overall, the algorithm will process every pixel of the image in the worst-case scenario. The resulting time complexity is $O(n \cdot m^2)$ and space complexity $O(n + m^2)$. Although, this scenario hardly will be reached since most parts of the images usually are not edge regions.

Knowing that the mask size m will usually be small, not surpassing 9, it can be treated as a constant value because it will have little impact on the execution time. Therefore, the time complexity of the REPAIR Anti-Aliasing method can be considered as $O(n)$ and the space complexity also $O(n)$.

3.5 Algorithms

Algorithm 1 presents the pseudo-code of the REPAIR method. It receives a color image, the filter size, and the threshold weight. Its output is the processed image with anti-aliasing. The algorithm follows the workflow in Figure 3.10.

Algorithm 1: REPAIR Anti-Aliasing Pseudo-Code

Input: Color image: img, Filter size: n, Threshold weight: tw

Result: Anti-Aliased image

```
1  bwImg ← grayscale(img)
2  bwImg ← sobelGradientFilter(bwImg)
3  threshold ← meanThreshold(bwImg, tw)
4  for i ← 0 to bwIimg.Height() do
5      for j ← 0 to bwImg.Width() do
6          if bwImg(i, j) < threshold then
7              bwImg(i, j) ← 0
8          end
9      end
10 end
```

```
11 patch_{n×n} ← ∅
12 rgbPatch_{n×n×3} ← ∅
13 blurMask_{n×n} ← averageFilter (n)
14 dstImg ← emptyColorImage()
15 for i ← 0 to img.Height() do
16     for j ← 0 to img.Width() do
17         if bwImg(i, j) ≠ 0 then
18             a ← (n − 1)/2
19             for k ← −a to a do
20                 for l ← −a to a do
21                     patch(k, l) ← bwImg(i + k, j + l)
22                     rgbPatch(k, l) ← img(i + k, j + l)
23             θ ← slopeAngle(patch)
24             dstImg(i, j) ← rotatedBilinear(rgbPatch, blurMask, θ)
25 return dstImg
```

Algorithm 2 calculates a threshold value based on the mean of the pixel intensities of the input image. The threshold is adjusted by a weight κ. This function is called at line 3 of the REPAIR anti-aliasing algorithm, and its output is an integer value.

Algorithm 2: Mean Threshold Function Pseudo-Code

Input: Black & white image: img, Threshold weight: κ

Result: Threshold value

```
1  n, s, s2 ← 0
2  for i ← 0 to img.Width() do
3      for j ← 0 to img.Height() do
4          if img(i, j) ≠ 0 then
5              s ← s + img(i, j)
6              s2 ← s2 + img(i, j)^2
7              n ← n + 1
8          end
9      end
10 end
11 μ ← s/n
12 σ^2 ← s2 − (2 · μ · s) + μ^2
13 t ← μ − κ * σ
14 return t
```

Algorithm 3 estimates the pixel direction using simple linear regression in Equation (3.5). Its input is a neighborhood of pixels, and its output is a real value θ indicating the slope angle of the pixel. This function is called at line 25 of the REPAIR anti-aliasing algorithm.

Algorithm 3: Slope Angle Function Pseudo-Code

Input: Patch $n \times n$ from B&W image

Result: Slope angle

```
1  wVector_n, hVector_n ← ∅
2  a ← (n − 1)/2
3  for i ← −a to a do
4      wVector(i) ← weightedMeanOfColumn(i)
5      hVector(i) ← weightedMeanOfLine(i)
6  end
7  α_1 = computeSlopeCoeff(wVector)
8  β_1 = computeSlopeCoeff(hVector)
9  θ ← arctan(β_1/α_1)
10 return θ
```

Algorithm 4 estimates the intensity of a pixel with real coordinates and applies a spatial filter to it. Its input is a neighborhood of pixels and a spatial filter, and its output is an interpolated pixel with a filter already applied to it. Bilinear interpolation is used to perform the task, and correlation is applied at line 14, Equation (2.2). This function is called at line 26 of the REPAIR anti-aliasing algorithm.

Algorithm 4: Rotated Bilinear Interpolation Function Pseudo-Code

Input: $Patch\ n \times n$ from color image, smoothing filter $mask\ m \times m$, slope angle: θ

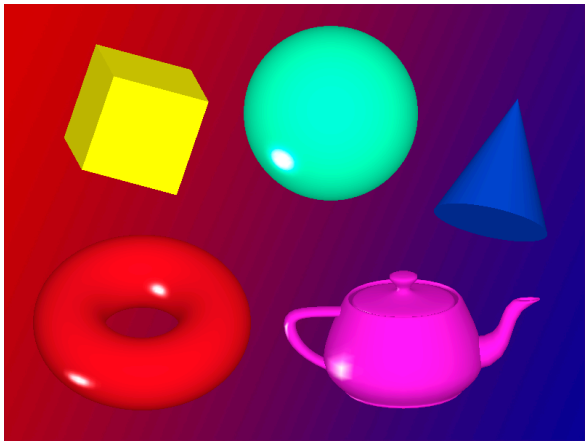
Result: Resulting pixel

1 $result \leftarrow 0$
2 $a = (n-1)/2$
3 **for** $w \leftarrow -a$ *to* a **do**
4   **for** $h \leftarrow -a$ *to* a **do**
5         $w' \leftarrow w \cdot \cos\theta - h \cdot \sin\theta$
6         $h' \leftarrow w \cdot \sin\theta + h \cdot \cos\theta$
7         $a \leftarrow patch(\lfloor w' \rfloor, \lfloor h' \rfloor)$
8         $b \leftarrow patch(\lceil w' \rceil, \lfloor h' \rfloor)$
9         $c \leftarrow patch(\lfloor w' \rfloor, \lceil h' \rceil)$
10         $d \leftarrow patch(\lceil w' \rceil, \lceil h' \rceil)$
11         $f_1 \leftarrow a + (h' - \lfloor h' \rfloor) \cdot (b - a)$
12         $f_2 \leftarrow b + (h' - \lfloor h' \rfloor) \cdot (d - c)$
13         $f(w', h') \leftarrow f_1 + (w' - \lfloor w' \rfloor) \cdot (f_2 - f_1)$
14         $result \leftarrow result + patch(w', h') \cdot mask(w, h)$
15   **end**
16 **end**
17 return $round(result)$

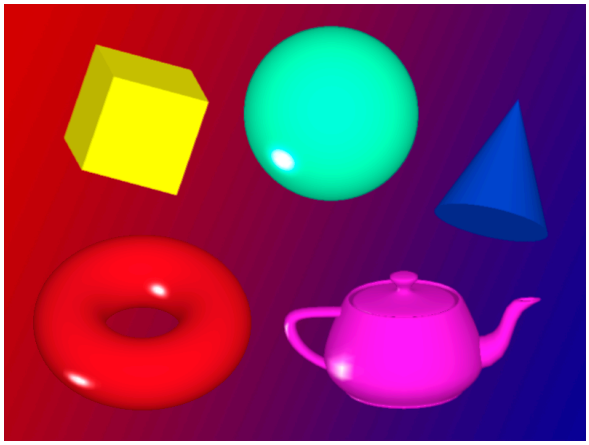
4 RESULTS AND DISCUSSIONS

This chapter presents and evaluates the obtained results during the several tests performed in the method's development. The preliminary results were obtained from a software written in C++ using OpenGL Application Programming Interface (API). It displays diverse geometric shapes in front of a gradient background, resulting in the scene shown in Figure 4.1a. The software reads OpenGL's framebuffer and processes the image. Then, it overwrites the framebuffer to show the processed image on the screen. Figure 4.1 shows a comparison of different smoothing filters applied to Figure 3.1a.

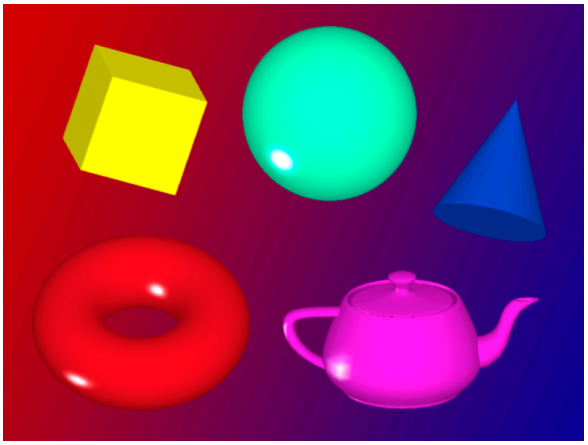
Figure 4.1 – Comparison between different smoothing filter masks.



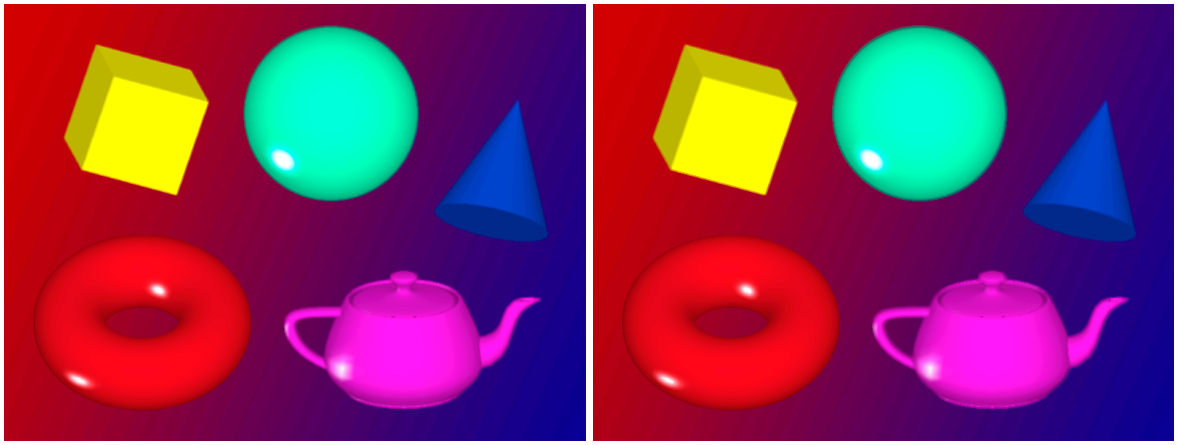
(a) Original image



(b) Gaussian blur mask 5×5



(c) Average mask 3×3



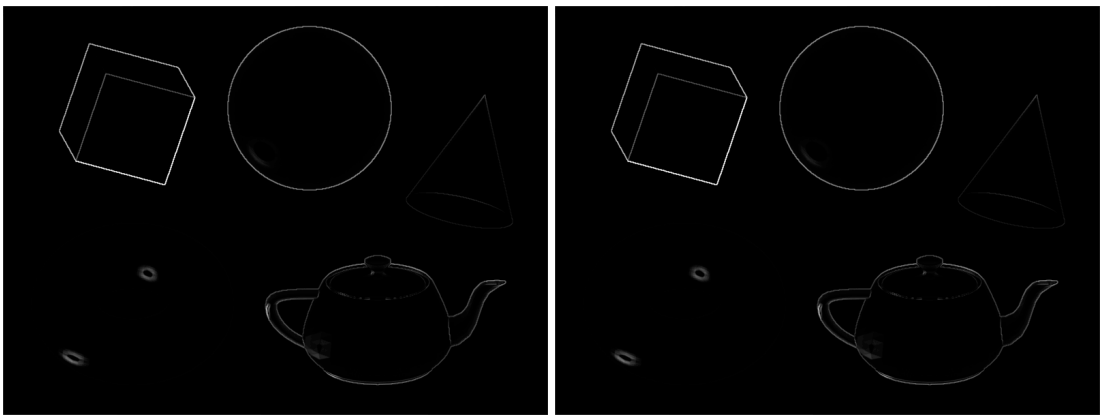
(d) Average mask 5×5 (e) Average mask 7×7

Source: Own authorship.

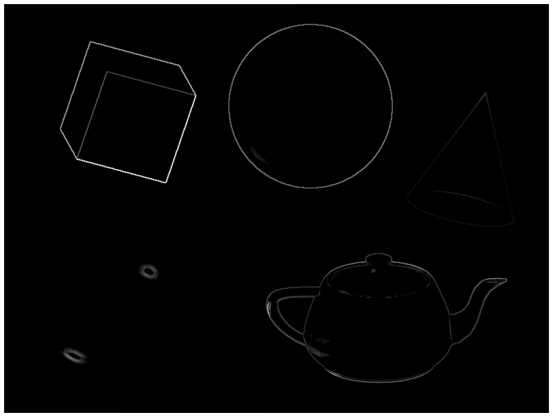
Oversized masks add more blur to the image, reducing the details of objects likewise aliasing. The ideal setting reduces aliasing by inserting as slight a blur as possible, such as Gaussian Blur mask 5×5 or average mask 3×3.

Figure 4.2 shows what happens with different threshold weights. Low values tend not to modify the filtered image, blurring more pixels than necessary, while high values erase all lines, leaving too many untreated regions.

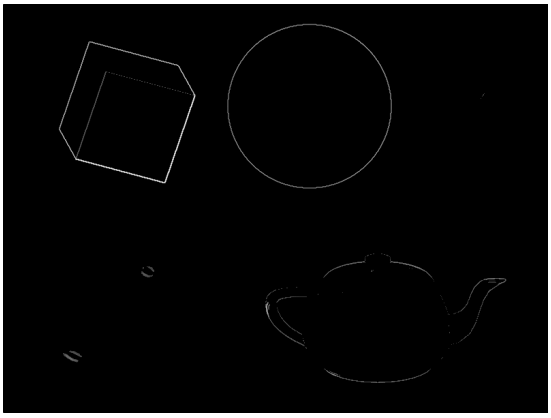
Figure 4.2 – Comparison between different threshold weights κ.



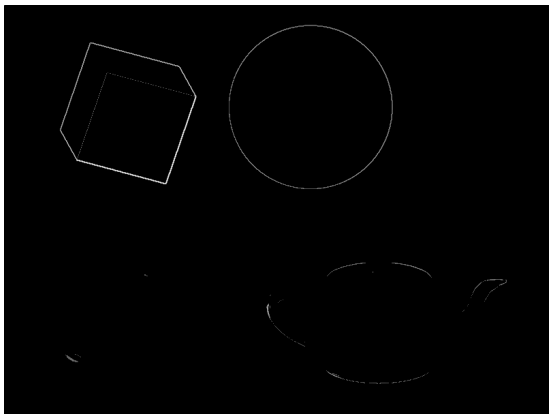
(a) $\kappa = -2.0$. (b) $\kappa = -1.0$.



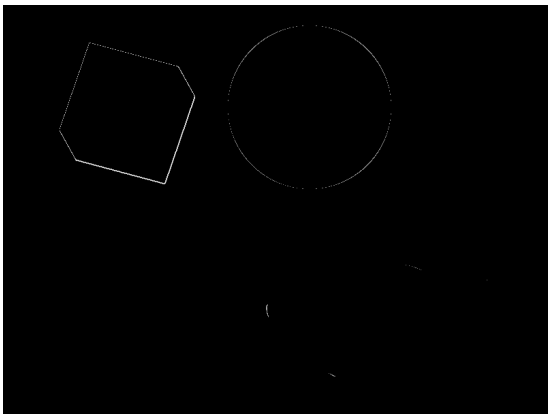
(c) $\kappa = -0.5$.



(d) $\kappa = 0.5$.



(e) $\kappa = 1.0$.

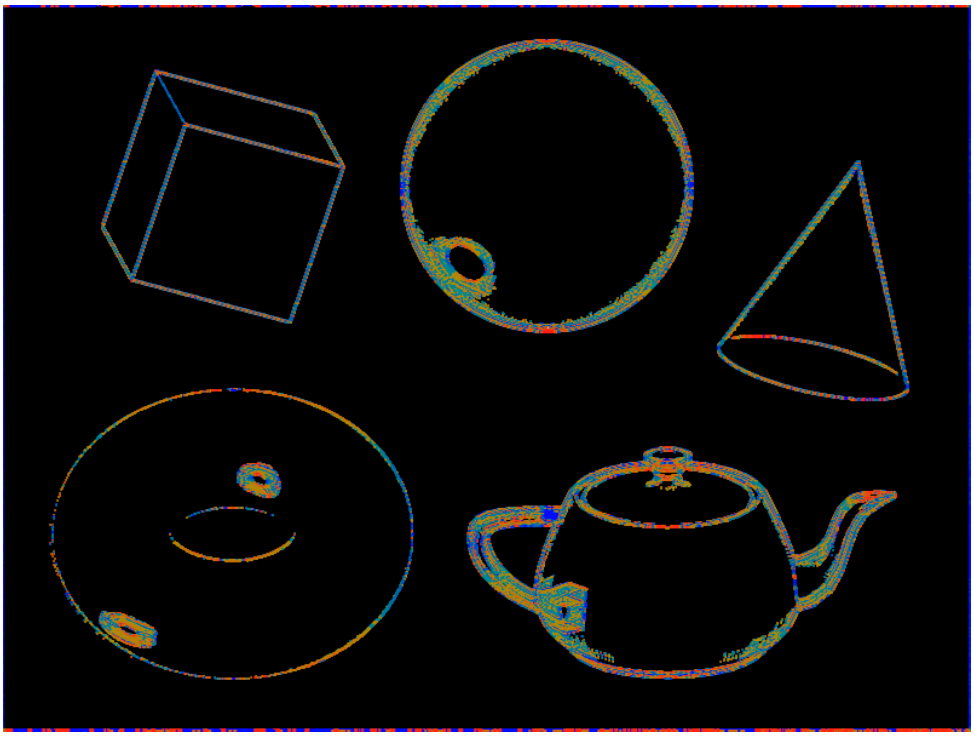


(f) $\kappa = 2.0$.

Source: Own authorship.

Figure 4.3 illustrates the slope angle computation, coloring the pixels according to their inclination. The pixels with direction closer to $0°$ are blue, while pixels closer to $90°$ are red, making visible the edges directions. Notice that $0°$ and $90°$ will provide the same result since the smoothing filters shown in Subsection 2.2 are isotropic. Also, the filter might assess regions with similar pixels intensities inside the line in regions with thicker lines, resulting in angles close to $0°$.

Figure 4.3 – Heatmap representing the direction of the objects edges. As the slope angle is closer 90°, more close to red the pixel is plotted.



Source: Own autorship.

Blender was utilized to produce more realistic scenes to assess the REPAIR anti-aliasing results. Blender is a free and open-source software for 3D modeling. It supports the entire 3D pipeline, allowing modeling, rigging, animation, simulation, rendering, compositing, and motion tracking (Blender Foundation, 2020a).

The rendering of the scenes was made with Eevee, a fast renderer available on Blender. The application was modified to process the rendered files from Blender. It was configured to render with only one sample, without anti-aliasing or any features that might smooth the edges. Because aliasing is automatically reduced when multisampling is used, these cares are taken in the renderer settings.

In order to display the details of the resulting image, they need to be augmented. To preserve the original characteristics of the images, the nearest neighbor zoom, also known as pixel replication, is used. This method replicates the pixels without modifying their values, making it possible to highlight the aliasing to the viewer without any improvement in the scene. The following sections present several images treated with REPAIR anti-aliasing.

4.1 Stanforfd 3D Scanning Repository

The Stanford 3D Scanning Repository is a repository containing scanned 3D models. The scenes are presented on a checkered floor and a solid white background illuminated by

5 CONCLUSION

In this study, a new anti-aliasing method called REPAIR Anti-Aliasing was proposed. It is a post-filtering anti-aliasing treatment that aims to reduce the jagged appearance that affects object borders. The aliasing problem occurs during the render process of a graphic scene, and although there is no definitive solution, it can be diminished (GONZALEZ; WOODS, 2009). The REPAIR Anti-Aliasing is divided into five steps: grayscale conversion, sharpening spatial filtering, threshold application, slope angle estimation, and rotated spatial filtering.

Firstly, the grayscale conversion is based on the YIQ color system because it takes advantage of human eye color response to produce good results. Then, the Sobel gradient is used for sharpening spatial filtering to highlight the image edges, where aliasing usually occurs. Later, the threshold is applied to the filtered image to clean noise and unimportant regions. Finally, the Gaussian blur, low-pass spatial filtering, is applied to blur the jagged edges. The smoothing filter is rotated to match the edge inclination, improving the smoothing quality.

The Related Works Section shows that most anti-aliasing techniques demand supersampling the image or store buffers, consuming computational resources and impacting the application's performance. Some methods work in the render pipeline to achieve the results. The proposed method aims to be simple and cost-efficient. With execution time $O(n)$ and memory consumption also $O(n)$, the REPAIR method reduces aliasing efficiently, demanding low processing cost.

The tests were performed on an OpenGL application developed in C++. The primary strategy was to read the OpenGL framebuffer, retrieve the rendered image, process it with REPAIR Anti-Aliasing, and then overwrite the framebuffer with the treated image to be displayed.

For testing purposes, an alternative version of the software was created to process image files. Blender is 3D modeling software used to build more complex scenes with 3D models, textures, shadows, and other graphic elements. Since REPAIR Anti-Aliasing is a post-filtering method, its operation was not changed.

The REPAIR anti-aliasing has shown effectiveness in improving the quality of all tested images. They had their edges smoothed and their details preserved. The method works on all edges, shadows, and textures. If there is an abrupt transition of colors, the method must treat the region. Furthermore, the scene illumination is not affected by the method. Overall, the results are satisfactory.

The algorithm's efficiency comes from its selectivity to treat only regions where aliasing is expected to occur, ignoring the others. The Gaussian filter has shown better results than others filters due to its weighing system, keeping the colors closer to the center pixel. Moreover, the filter rotation process improves the smoothing quality since it is applied in the same edge

direction, preventing excessive blur. Thus, REPAIR anti-aliasing executes in the right places and the right direction.

It was impossible to compare the REPAIR method with other anti-aliasing methods because of the scarcity of documentation. Most of them are embedded in proprietary solutions. Hence, there is no access to its implementations, and there is no standard metric to evaluate them.

5.1 Future Work

The REPAIR method is currently implemented in standard C++, executing on Central Processing Unit (CPU), which is inappropriate for image processing. So, the performance can be increased by parallelizing the solution.

As future work, the algorithm can be ported to programming languages such as CUDA or OpenCL to execute on Graphics Processing Units (GPUs) since they are appropriate devices to perform image processing. Another alternative is to develop a custom solution on Field Programmable Gate Array (FPGA), a platform for digital circuits implementation. The ability to implement the algorithm on hardware also can increase its performance significantly.

Furthermore, a hybrid version of the algorithm can be developed using Hybrid Computation concepts. The algorithms tasks can be separated into different devices such as CPU, GPU, and FPGA. This solution can bring highly efficient parallelism to solve the problem fast.

Moreover, machine learning techniques can be applied to improve the smoothing filtering. A custom smoothing filter can be created for each filtered pixel, weighting its terms according to each neighborhood's intensities.

Bibliography

ALVES, R. F. M. *Aplicação de Antialiasing em Cenas Gráficas Utilizando Filtragem Espacial de Imagens*. 46 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal Rural do Semi-Árido, Mossoró, RN, 2018.

ANGEL, E.; SHREINER, D. *Interactive Computer Graphics: A Top Down Approach with WebGL*. 7. ed. Upper Saddle River, New Jersey, USA: Pearson, 2015.

AZEVEDO, E.; CONCI, A. *Computação Gráfica: Teoria e Prática*. 1. ed. Rio de Janeiro, RJ: Campus, 2003.

BARRINGER, R.; AKENINE-MOLLER, T. A4: Asynchronous adaptive anti-aliasing using shared memory. *ACM Transactions on Graphics*, v. 32, 2013. ISSN 07300301. Anti-Aliasing que utiliza memória compartilhada em CPU e GPU. Cited at page 27.

BARROSO, L. C. et al. *Cálculo Numérico (Com Aplicações)*. 2. ed. São Paulo, SP: Harbra Ltda., 1987.

Blender Foundation. *Blender*. 2020. Version 2.91. Disponível em: <https://www.blender.org/>. Cited 2 times at pages 14 and 50.

Blender Foundation. *Blender Demo Files*. 2020. Version 2.91. Disponível em: <https://www.blender.org/download/demo-files/>. Cited 2 times at pages 7 and 69.

BlenderKit. *Online BlenderKit library*. 2017. Version 1.0.32. Disponível em: <https://www.blenderkit.com/>. Cited 16 times at pages 7, 14, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, and 68.

BORJAS, S. D. M. *Cálculo Numérico*. 1. ed. Mossoró, RN: EdUFERSA, 2013.

BURGER, W.; BURGE, M. J. *Digital Image Processing: An Algorithmic Introduction Using Java*. 2. ed. London, UK: Springer, 2016.

CORMEN, T. H. et al. *Introduction to Algorithms*. 3. ed. Massachusetts, USA: The MIT Press, 2009.

CRASSIN, C. et al. Aggregate g-buffer anti-aliasing. p. 109–119, 2013. Aplicado na rasterização dos pixels.

Utiliza multi-sampling na rasterização para computar valores agregados. Cited at page 27.

ENDERTON, E. et al. Accumulative anti-aliasing. *ACM SIGGRAPH 2015 Talks, SIGGRAPH 2015*, 2015. Cited at page 27.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 3. ed. New Jersey, USA: Pearson, 2008. Cited 9 times at pages 12, 16, 17, 18, 19, 20, 21, 22, and 37.

GONZALEZ, R. C.; WOODS, R. E. *Processamento Digital de Imagens*. 3. ed. São Paulo, Brasil: Pearson, 2009. Cited 2 times at pages 12 and 71.

HEARN, D.; BAKER, M. P. *Computer Graphics: C Version*. 2. ed. New Jersey, USA: Prentice Hall, 1997. Cited at page 12.

HEARN, D. D.; BAKER, M. P.; CARITHERS, W. *Computer Graphics with OpenGL*. 4. ed. Harlow, United Kingdom: Pearson, 2014.

HUGHES, J. F. et al. *Computer Graphics: Principles and Practice*. 3. ed. Ohio, USA: Pearson, 2014. Cited 2 times at pages 12 and 13.

IOURCHA, K.; YANG, J. C.; POMIANOWSKI, A. A directionally adaptive edge anti-aliasing filter. *Proceedings of the HPG 2009: Conference on High-Performance Graphics 2009*, v. 1, p. 127–134, 2009. Cited at page 26.

JIMENEZ, J. et al. Filtering approaches for real-time anti-aliasing. *ACM SIGGRAPH 2011 Courses, SIGGRAPH'11*, 2011. This one! Cited 2 times at pages 29 and 30.

KIRK, D. B.; HWU, W. mei W. *Programming Massively Parallel Processors: A Hands-on Approach*. 2. ed. Waltham, MA, USA: Morgan Kaufman, 2013. Cited at page 12.

KVETNY, R. N.; KOSTROVA, C.; BOGATCH, I. Anti-aliasing algorithms based on self-similar multitudes. *Selected Papers from the International Conference on Optoelectronic Information Technologies*, v. 4425, p. 83, 2001. ISSN 0277786X. Cited at page 26.

LEE, J. et al. Selective multi-sample anti-aliasing for mobile vector graphics. *2016 IEEE International Conference on Consumer Electronics, ICCE 2016*, IEEE, p. 174–175, 2016. Cited at page 28.

LIU, S.; LI, L. A fast anti-aliased rendering line algorithm for real coordinate. *Proceedings - 2018 International Conference on Robots and Intelligent System, ICRIS 2018*, p. 464–467, 2018. Cited at page 28.

LUO, D.; ZHANG, J. The #-filter anti-aliasing based on sub-pixel continuous edges. *Mathematics*, v. 8, 2020. ISSN 22277390. Cited at page 29.

MATHWORKS. *rgb2ntsc: Convert RGB color values to NTSC color space*. Image Processing Toolbox Documentation. <https://www.mathworks.com/help/images/ref/rgb2ntsc.html>. Acessed: 2021-07-13. Cited at page 16.

MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. *Introduction to Linear Regression Analysis*. 5. ed. New Jersey, USA: Wiley, 2012. Cited 3 times at pages 23, 24, and 25.

MONTGOMERY, D. C.; RUNGER, G. C. *Applied Statistics and Probability for Engineers*. 6. ed. Arizona, USA: Wiley, 2014. Cited 3 times at pages 18, 22, and 23.

NAH, J. ho et al. Axaa: Adaptive approximate anti-aliasing. p. 1–2, 2009. Cited at page 26.

PHILLIPS, D. *Image Processing in C*. 2. ed. Kansas, USA: R & D Publications, 2000.

RAJARAPOLLU, P. R.; MANKAR, V. R. Design and analysis of anti-aliasing filters for rendering of graphical entities. *ACM*, 03 2016. Cited at page 27.

RUGGIERO, M. A. G.; LOPES, V. L. da R. *Cálculo Numérico: Aspectos Teóricos e Computacionais*. 2. ed. São Paulo, SP: Pearson, 1996.

SALVI, M.; VIDIMCE, K. Surface based anti-aliasing. *Proceedings - I3D 2012: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, v. 1, p. 159–164, 2012. Cited at page 27.

SCHAULAND, S.; VELTEN, J.; KUMMERT, A. A new anti-aliasing approach for improved motion-based object detection using linear filters. *IEEE Intelligent Vehicles Symposium, Proceedings*, IEEE, p. 915–920, 2010. Cited at page 26.

SCHOLLMEYER, A.; FROEHLICH, B. Efficient and anti-aliased trimming for rendering large nurbs models. *IEEE Transactions on Visualization and Computer Graphics*, IEEE, v. 25, p. 1489–1498, 2019. ISSN 19410506. Cited at page 28.

SOUZA, L. C. de. *Agrupamento e Regressão Linear de Dados Simbólicos Intervalares Baseados em Novas Representações*. Tese (Doutorado) — Universidade Federal de Pernambuco, Recife, PE, 3 2016. Cited at page 22.

Stanford University. *The Stanford 3D Scanning Repository*. 2014. Disponível em: <http://graphics.stanford.edu/data/3Dscanrep/>. Cited 5 times at pages 6, 7, 51, 52, and 54.

SUNG, M.; CHOI, S. Selective anti-aliasing for virtual reality based on saliency map. *Proceedings - 2017 International Symposium on Ubiquitous Virtual Reality, ISUVR 2017*, IEEE, p. 16–19, 2017. Cited at page 28.

Swan II, J. E. et al. An anti-aliasing technique for splatting. Cited at page 26.

WANG, Y. et al. Decoupled coverage anti-aliasing. *Proceedings - High Performance Graphics 2015*, p. 33–42, 2015. Cited at page 27.

WIERINGA, R. J. *Design Science Methodology: for Information Systems and Software Engineering*. 1. ed. Enschede, Netherlands: Springer, 2014.

XIAO-LIANGI, M. et al. Implementation of dynamic anti-aliased shadow algorithm in 3d scene. p. 1735–1737, 2007. Cited at page 26.

ZHAO, C. et al. Applications of a deep learning method for anti-aliasing and super-resolution in mri. *Magnetic Resonance Imaging*, Elsevier, v. 64, p. 132–141, 2019. ISSN 18735894. Disponível em: <https://doi.org/10.1016/j.mri.2019.05.038>. Cited at page 28.