



UNIVERSIDAD SIMÓN BOLÍVAR  
DECANATO DE ESTUDIOS PROFESIONALES  
COORDINACIÓN DE INGENIERÍA DE COMPUTACIÓN

**MUESTREO MEJORADO PARA TEMPORAL ANTI-ALIASING**

Por  
Oliveros Labrador/Christian Alexander

**TRABAJO DE GRADO**

Presentado ante la ilustre Universidad Simón Bolívar  
como requisito requisito parcial para optar al título de  
Ingeniero en Computación

Sartenejas, Julio de 2018

C. A. OLIVEROS  
LABRADOR  
2018

MUESTREO MEJORADO PARA TEMPORAL  
ANTI-ALIASING

USB  
CARRERA



UNIVERSIDAD SIMÓN BOLÍVAR  
DECANATO DE ESTUDIOS PROFESIONALES  
COORDINACIÓN DE INGENIERÍA DE COMPUTACIÓN

**MUESTREO MEJORADO PARA TEMPORAL ANTI-ALIASING**

Por  
Oliveros Labrador/Christian Alexander

Realizado con la asesoría de:

Ángela Di Serio  
Michael Doggett (Lund)

**TRABAJO DE GRADO**

Presentado ante la ilustre Universidad Simón Bolívar  
como requisito requisito parcial para optar al título de  
Ingeniero en Computación

Sartenejas, Julio de 2018

Página reservada para el acta de evaluación

## **DEDICATORIA**

Debe ser elaborada bajo las mismas normas del desarrollo del trabajo y mismo tipo de letra seleccionado (tamaño 12).

Dedicado a la Universidad Simón Bolívar y a su comunidad universidad.

## AGRADECIMIENTOS

*“El día que dejemos de explorar será el día en que nos comprometemos a vivir en un mundo estancado, desprovisto de curiosidad, vacío de sueños”. – Neil deGrasse Tyson*

Tengo muchas personas para agradecer por esta increíble aventura. Viniendo de Venezuela, a más de 8000 km de distancia de Suecia, siendo la primera vez que estoy fuera de casa por tanto tiempo, la primera vez que tomo clases sólo en inglés y ser el primer estudiante de intercambio en realizar una Tesis de Maestría en el Departamento de Ciencias de la Computación de la Universidad de Lund, ha sido una experiencia que me cambió la vida.

Me gustaría agradecer a toda mi familia, especialmente a mi papá, Wilmer Oliveros; mi mamá, Raixa Labrador; y mi hermano, Jean Pierre Oliveros, por toda su ayuda y apoyo durante este viaje. Además, me gustaría agradecer a mis amigos de todo el mundo. Gracias por ayudarme a ser lo que soy en este momento y por estar conmigo todo este tiempo.

Me gustaría agradecer a mi supervisor de la Universidad de Lund, el profesor Michael Doggett, por darme esta increíble oportunidad y por ayudarme durante su transcurso. Además, me gustaría agradecer a Pierre Moreau por ayudarme en las veces que estuve estancado. Además, me gustaría agradecer a mi supervisora de la Universidad Simón Bolívar, la profesora Angela Di Serio; y mi Coordinadora de Ingeniería de la Computación, la profesora Marlene Goncalves, por ayudarme y apoyarme en este increíble viaje.

Finalmente, me gustaría agradecer a la Universidad Lund y la Universidad Simón Bolívar, con un agradecimiento especial a la Oficina Internacional LTH; el Departamento de Ciencias de la Computación de la Universidad de Lund, la Coordinación de Ingeniería de la Computación de la Universidad Simón Bolívar; y la Dirección de Relaciones Internacionales y Cooperación de la Universidad Simón Bolívar, por permitir que los estudiantes aprovechamos estas oportunidades de desarrollo académico y de auto crecimiento.



LUND UNIVERSITY UNIVERSIDAD SIMÓN BOLÍVAR

## RESUMEN

*Anti-aliasing* es un componente clave de los sistemas modernos de computación gráfica 3D. Para la generación de imágenes en tiempo real en aplicaciones como juegos, es importante aumentar la tasa de muestreo por píxel para mejorar la calidad general de la imagen. Pero aumentar el muestreo puede ser costoso, especialmente para las arquitecturas actuales de *Shading Diferido*. Una solución innovadora a este problema es la técnica de *Temporal Anti-Aliasing* (TAA), que combina muestras de cuadros previos con las muestras del cuadro actual para aumentar efectivamente la tasa de muestreo. En esta tesis, exploraremos métodos para mejorar la calidad del TAA mediante el uso de detección de bordes, tanto de color como de profundidad, y la indexación de triángulos para asegurar que sólo las muestras pertenecientes a los píxeles del cuadro actual se combinen. Nuestro objetivo es reducir el efecto de ghosting y otros artefactos creados con las implementaciones actuales de TAA. La mejora de la calidad se evaluará comparando las imágenes generadas por TAA con las imágenes base (*ground truth*) generadas mediante el uso de tasas de muestreo mucho más altas, que no serían prácticas en tiempo real. El TAA mejorado se probó utilizando métricas de imágenes, en particular, MSE, PSNR y SSIM, para comparar con la implementación de TAA original y otras técnicas actuales de Anti-Aliasing. Los resultados obtenidos mostraron que las mejoras aplicadas a TAA enriquecieron la calidad de la imagen por encima de las técnicas existentes, con valores de PSNR en torno a 39 y valores de SSIM por encima de 0,99.

**Palabras Clave:** TAA, Sobel, Anti-Aliasing, Triangle Indexing, TRAA.

# ÍNDICE GENERAL

<b>DEDICATORIA</b>	iii
<b>AGRADECIMIENTOS</b>	iv
<b>RESUMEN</b>	v
<b>ÍNDICE GENERAL</b>	vi
<b>ÍNDICE DE FIGURAS</b>	ix
<b>ÍNDICE DE TABLAS</b>	xi
<b>LISTA DE ACRÓNIMOS</b>	xii
<b>NOTACIÓN MATEMÁTICA</b>	xiv
<b>CAPÍTULO I: INTRODUCCIÓN</b>	1
1.1. Definición del Problema . . . . .	1
1.2. Trabajos Relacionados . . . . .	2
1.3. Contribución . . . . .	3
<b>CAPÍTULO II: MARCO TECNOLÓGICO</b>	4
2.1. C++ y el Bonobo Framework . . . . .	4
2.2. OpenGL y GLSL . . . . .	4
2.3. MATLAB . . . . .	5
<b>CAPÍTULO III: MARCO TEÓRICO</b>	6
3.1. Canal de Renderizado . . . . .	6
3.2. Proceso de Rasterización . . . . .	7
3.3. El Problema del Aliasing . . . . .	7
3.4. Mapeado de Sombras y la Arquitectura de Shading Diferido . . . . .	9
3.5. Anti-Aliasing . . . . .	10
3.5.1. Super Sampling Anti-Aliasing (SSAA) . . . . .	10
3.5.2. Multi Sample Anti-Aliasing (MSAA) . . . . .	10
3.5.3. Fast Approximate Anti-Aliasing (FXAA) . . . . .	11
3.5.4. Enhanced Subpixel Morphological Antialiasing (SMAA) . . . . .	11
3.6. Temporal Anti-Aliasing . . . . .	11

3.6.1.	Camera Jittering . . . . .	12
3.6.2.	Buffer de Velocidad . . . . .	12
3.6.3.	Buffer de Historia . . . . .	13
3.6.4.	Clipping Color Box . . . . .	14
3.6.5.	Filtro de Nitidez . . . . .	15
3.6.6.	Motion Blur . . . . .	15
3.6.7.	Problemas . . . . .	16
3.7.	Buffer de Acumulación . . . . .	17
3.8.	Operador de Sobel . . . . .	17
3.9.	Métricas de Imagen . . . . .	18
3.9.1.	Error Cuadrado Medio (MSE) . . . . .	19
3.9.2.	Desviación de la Raíz Cuadrada Media (RMSD) . . . . .	19
3.9.3.	Relación Señal a Ruido de Pico (PSNR) . . . . .	19
3.9.4.	Índice de Similitud Estructural (SSIM) . . . . .	19
<b>CAPÍTULO IV: DESARROLLO</b>		<b>21</b>
4.1.	Mejoras al Proyecto de EDAN35 . . . . .	21
4.1.1.	Fast Approximate Anti-Aliasing (FXAA) . . . . .	22
4.1.2.	Enhanced Subpixel Morphological Antialiasing (SMAA) . . . . .	22
4.1.3.	Buffer de Acumulación . . . . .	23
4.2.	Implementación del Marco de Pruebas . . . . .	24
4.2.1.	Pruebas Estáticas . . . . .	24
4.2.2.	Pruebas de Ghosting . . . . .	25
4.2.3.	Métricas de Imagen de MATLAB . . . . .	25
4.3.	Modificaciones de Temporal Reprojection Anti-Aliasing . . . . .	26
4.3.1.	Implementación de Mejoras de Indexación de Triángulos . . . . .	27
4.3.2.	Pseudo-Varianza de Profundidad y Pseudo-Varianza Temporal de Profundidad . . . . .	29
4.3.3.	Implementación de las Mejoras de Sobel . . . . .	30
4.3.4.	Mezcla Final . . . . .	31
4.3.5.	Sharpen Filter Modifications . . . . .	34
<b>CAPÍTULO V: RESULTS</b>		<b>35</b>
5.1.	Evaluation Methodology . . . . .	35
5.2.	Results and Comparisons . . . . .	36
5.2.1.	Sharpen Filter . . . . .	36
5.2.2.	Pipe . . . . .	38
5.2.3.	Window with Blinds . . . . .	42
5.2.4.	Arched Window . . . . .	43
5.2.5.	Sponza Atrium . . . . .	44
5.2.6.	Sponza Atrium Flowers . . . . .	46
5.2.7.	Hard Edges . . . . .	47
5.2.8.	Sphere Ghosting . . . . .	49
5.2.9.	Hairball . . . . .	50
5.2.10.	Timing . . . . .	56

5.3. Discussion . . . . .	56
<b>CAPÍTULO VI: CONCLUSIONS AND RECOMMENDATIONS</b>	<b>59</b>
<b>REFERENCIAS</b>	<b>60</b>
<b>APÉNDICE A: GITHUB REPOSITORY</b>	<b>62</b>
<b>APÉNDICE B: CAMERA JITTERING EXPLANATION</b>	<b>63</b>

## ÍNDICE DE FIGURAS

3.1. Canal de Renderizado, basado en la 5ta clase de EDAF80. [12] . . . . .	6
3.2. Ejemplo de los resultados del Proceso de Rasterización. <i>Nota:</i> se agregaron colores para diferenciar los triángulos, pero solo se agregarían en la etapa del <i>Pixel Shader</i> . . . . .	7
3.3. Ejemplo del problema de Cobertura Parcial, basado en la segunda clase de EDAN35. [13] . . . . .	8
3.4. Imagen Base de una línea frente a su Aproximación con <i>Aliasing</i> . . . . .	9
3.5. Proceso de <i>Jittering</i> sobre la proyección de la cámara. Imagen tomada de la presentación de Fuglsand. [6] . . . . .	12
3.6. Valores usados de la <i>HaltonSequence(2, 3)</i> (Secuencia de Halton). . . . .	13
3.7. Patrón de Muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [6] . . . . .	14
3.8. Proceso de <i>Temporal Reprojection Anti-Aliasing</i> . Imagen tomada de la presentación de Fuglsand [6] . . . . .	14
3.9. <i>Color Clamping</i> versus <i>Color Clipping</i> . Imagen tomada del <i>paper</i> de Fuglsand. [6] . . . . .	15
 4.1. Los 128 de la <i>SecuendiaHalton(2, 3)</i> disponibles para usar. . . . .	22
4.2. Patrón de muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [6] . . . . .	27
4.3. Reducción de tamaño de la Color Clipping Box . . . . .	27
4.4. Imagen hecha de los valores de <i>aliasing</i> de cada píxel. . . . .	33
4.5. Imagen hecha de los valores de <i>aliasing</i> de cada píxel. . . . .	33
4.6. Imagen hecha de los valores de <i>aliasing</i> de cada píxel. . . . .	33
 5.1. Rendered Images comparison. . . . .	38
5.2. SSIM Maps comparison. . . . .	38
5.3. Pipe Regular comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	40
5.4. Pipe Regular Test ground truth. . . . .	40
5.5. Pipe with Camera Inclination comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	41
5.6. Pipe with Camera Inclination Test ground truth. . . . .	41
5.7. Window with Blinds ground truth. . . . .	42
5.8. Window with Blinds comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	43

5.9. Arched Window comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	44
5.10. Arched Window ground truth. . . . .	44
5.11. Sponza Atrium comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	45
5.12. Sponza Atrium ground truth. . . . .	45
5.13. Sponza Atrium Flowers comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	47
5.14. Sponza Atrium Flowers ground truth. . . . .	47
5.15. Hard Edges comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	48
5.16. Hard Edges ground truth. . . . .	48
5.17. Example of Ghosting. Comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 19. . . . .	50
5.18. Hairball Static Shadow comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	51
5.19. Hairball Static Shadow ground truth. . . . .	52
5.20. Hairball Static Lighted ground truth. . . . .	53
5.21. Hairball Static Lighted comparison between Master Thesis TAA, Uncharted TAA, and SMAA. . . . .	53
5.22. Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 90. . . . .	54
5.23. Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 83. . . . .	56

## ÍNDICE DE TABLAS

4.1. Comparación del comportamiento de las métricas entre el uso de 16 muestras frente a 128 para el Buffer de Acumulación. . . . .	23
5.1. Sharpen Filter Test numerical results . . . . .	37
5.2. Numerical results of the Pipe Test with regular camera inclination. . . . .	39
5.3. Numerical results of the Pipe Test with a skewed camera inclination. . . . .	41
5.4. Numerical results of the Window with Blinds Test. . . . .	42
5.5. Numerical results of the Arched Window Test. . . . .	43
5.6. Numerical results of the Sponza Atrium Test. . . . .	45
5.7. Numerical results of the Sponza Atrium Flowers Test. . . . .	46
5.8. Numerical results of the Hard Edges Test. . . . .	48
5.9. Numerical results summary of the 100 tests performed for the Sphere Ghosting Test. . . . .	49
5.10. Numerical results of the Hairball Static Shadow Test. . . . .	51
5.11. Numerical results of the Hairball Static Light Test. . . . .	52
5.12. Numerical results summary of the 100 tests performed for the Hairball Ghosting Shadow Test. . . . .	54
5.13. Numerical results summary of the 100 tests performed for the Hairball Ghosting Light Test. . . . .	55

## LISTA DE ACRÓNIMOS

**USB** Universidad Simón Bolívar

**DEP** Decanato de Estudios Profesionales

**PDF** *Portable Document Format*, Documento en Formato Portable©

**PNG** *Portable Network Graphics*, Gráficos de Red Portátiles

**2D** *Two Dimensions*, Dos Dimensiones

**3D** *Three Dimensions*, Tres Dimensiones

**TAA** *Temporal Anti-Aliasing*

**TXAA** *Temporal X Anti-Aliasing*, donde la X representa alguna versión de TAA

**TRA****A** *Temporal Reprojection Anti-Aliasing*, TAA utilizando reproyección (técnica base de TAA)

**SSAA** *Super Sampling Anti-Aliasing*, Anti-Aliasing con Super Muestreo

**MSAA** *Multi Sample Anti-Aliasing*, Anti-Aliasing Multi Muestra

**FXAA** *Fast Approximate Anti-Aliasing*, Anti-Aliasing Aproximado Rápido

**SMAA** *Enhanced Subpixel Morphological Anti-Aliasing*, Anti-Aliasing de Mejora Morfológica de Subpíxeles

**AA** *Anti-Aliasing*

**NOAA** *No Anti-Aliasing*

**API** *Application Programming Interface*, Interfaz de Programación de Aplicaciones

**OpenGL** *Open Graphics Library*, Librería Abierta de Gráficos

**GPU** *Graphics Processing Unit*, Unidad de Procesamiento de Gráficos

**GLSL** *OpenGL Shading Language*, Lenguaje de Shading de OpenGL

**MSE** *Mean Square Error*, Error Cuadrado Medio

**RMSD** *Root Mean Square Deviation*, Desviación de la Raíz Media Cuadrada

**RMSE** *Root Mean Square Error*, Raíz del Error Cuadrado Medio

**PSNR** *Peak Signal-To-Noise Ratio*, Relación Señal a Ruido de Pico

**SNR** *Signal-To-Noise Ratio*, Relación Señal a Ruido

**SSIM** *Structural Similarity Index*, Relación Señal a Ruido

## NOTACIÓN MATEMÁTICA

$\mathbb{R}$	Conjunto de números reales
$M_{m,n}$	Espacio de las matrices de tamaño $m$ por $n$ con entradas reales
$\mathcal{L}$	Operador de Laplace
$\emptyset$	Conjunto vacío

# CAPÍTULO I

## INTRODUCCIÓN

La Computación Gráfica moderna se basa en la presentación de escenas hechas de objetos representados como modelos compuestos de polígonos primitivos, siendo el triángulo el más utilizado. Esto para aprovechar su simplicidad y propiedades geométricas para crear algoritmos óptimos que manejen su renderizado (*rendering*). Los triángulos están compuestos por tres vértices, cada uno de los cuales consiste en una posición y otros parámetros asociados, por ejemplo, el color o las normales del triángulo, para la interpolación.

Cuando queremos renderizar los objetos en una escena, tomamos los vértices y los enviamos por el Canal de Renderizado (*Rendering Pipeline*). Allí, son procesados y mapeados a los píxeles de la pantalla con su color respectivo.

Este proceso tiene dos usos principales: Aplicaciones fuera de línea (*offline*), como películas; y aplicaciones en tiempo real (*online*), como los videojuegos. Cada uno de estos usos tiene sus requisitos y limitaciones, pero, para este proyecto, sólo prestaremos atención a las aplicaciones en tiempo real.

El objetivo de este proyecto es mejorar el algoritmo de *Temporal Anti-Aliasing*, técnica que aumenta la calidad de las imágenes, después del proceso de mapeo de triángulos a píxeles, al mezclar cuadros previamente renderizados con los actuales.

El principal requerimiento será lograr la representación de la escena con la mayor calidad posible, con dos restricciones principales: renderizar al menos treinta cuadros por segundo, sin causar una alta pérdida de cuadros por segundo; y trabajar con una cantidad limitada de memoria y ancho de banda, porque necesitamos poder ejecutarlo en una computadora promedio o un dispositivo móvil. [13, 3]

### 1.1. Definición del Problema

*Temporal Anti-Aliasing* (TAA) es una técnica en tiempo real, relativamente nueva, que proporciona buenos resultados sin incurrir en gran consumo de memoria o costos de procesamiento de otras técnicas. Las técnicas de detección de bordes y de indexación de

triángulos, parecen ser buenas candidatas para mejorar la calidad de TAA, al ayudar a reducir los efectos no deseados de *ghosting* y desenfoque (*blurring*) creados por las implementaciones actuales de TAA.

El objetivo de esta tesis es mejorar la técnica de *Temporal Anti-Aliasing* mediante el uso de detección de bordes, de color y profundidad, y técnicas de indexación de triángulos para reducir el efecto de desenfoque y *ghosting*, sin disminuir la calidad de la imagen renderizada o incurrir en un gran consumo de memoria o alto costo de procesamiento.

## 1.2. Trabajos Relacionados

La técnica más simple de *Anti-Aliasing* es *Super Sampling Anti-Aliasing* (SSAA, *Anti-Aliasing* con Super Muestreo), que consiste en renderizar a una resolución más alta y luego reducir la imagen a la resolución requerida. Otra técnica es *Multi Sample Anti-Aliasing* (MSAA, *Anti-Aliasing* Multi Muestra), que calcula el color para el píxel final solo una vez [13]. Podemos conocer de lo que se convertiría en la base del *Temporal Reprojection Anti-Aliasing* (TAA o TRAA, *Anti-Aliasing* de Reproyeción Temporal) en los artículos Aceleración del *Shading* en Tiempo Real con Caching de Reproducción Inversa por Nehab D., Sander PV, Lawrence J., Tatarchuk N., Isidoro JR [16], en el que describen cómo se pueden usar los pixel shaders para guardar información de los píxeles del cuadro anterior y reproyectarla en el siguiente cuadro; y Supermuestreo Amortizado por Yang L., Nehab D., Sander PV, Sitthiamorn P., Lawrence J., Hoppe H. [20] en el que describen cómo utilizar la reprojyección de cuadros previos en el actual, como método de *Anti-Aliasing* en tiempo real.

A continuación, encontramos técnicas de Post-Procesamiento como *Fast Approximate Anti-Aliasing* (FXAA, *Anti-Aliasing* Aproximado Rápido) por Timothy Lottes [11] que usa una forma de detección de bordes para corregir el *aliasing* mientras es compatible con la Arquitectura de *Shading* Diferido (*Deferred Shading Architecture*). También tenemos la implementación de Crytek del *Temporal Anti-Aliasing* (TAA o TXAA) explicada por Tiago Sousa en su presentación Métodos de *Anti-Aliasing* en CryENGINE 3 [10].

También disponemos de la técnica *Enhanced Subpixel Morphological Anti-Aliasing* (SMAA, *Anti-Aliasing* de Mejora Morfológica de Subpíxeles) de Jorge Jiménez, José I. Echeverría, Tiago Sousa y Diego Gutiérrez [9] que utiliza una técnica de reconstrucción de bordes más compleja, al mismo tiempo que puede trabajar con SSAA, MSAA y una forma básica de TAA.

Finalmente, tenemos las implementaciones TRAA de Ke Xu para Uncharted 4 y Lars Fuglsang para Inside, que implementan nuevos avances como el *Color Clipping Box*

(Caja de Recorte de Colores) y el Filtro de Nitidez (*Sharpen Filter*). Estas dos últimas implementaciones se utilizan como base de esta tesis. [6, 19]

### 1.3. Contribución

Esta tesis mejora los últimos avances de Ke Xu y Lasse Fuglsang [6, 19]. Proponemos el uso del Operador de Sobel para realizar detecciones de bordes y técnicas de indexación de triángulos para detectar píxeles que se consideran con aliasing. Una vez tenemos estos píxeles detectados, usamos su información para cambiar la forma en la que se rechazan los colores de los cuadros anteriores, a fin de reducir los artefactos de *ghosting* (*ghosting artifacts*) y desenfoque que tiene *Temporal Anti-Aliasing*.

# CAPÍTULO II

## MARCO TECNOLÓGICO

En este capítulo, explicaremos qué herramientas se utilizaron en esta tesis y por qué.

### 2.1. C++ y el Bonobo Framework

Utilizamos C ++ y el *Bonobo Framework* para implementar todas las mejoras realizadas en esta Tesis. C ++ es un lenguaje de programación de propósito general, compilado, con funciones imperativas, programación orientada a objetos y administración de memoria de bajo nivel. Lo usamos por su rendimiento, especialmente para aplicaciones de computación gráfica en tiempo real, y por su amplia base de conocimiento.

El *Bonobo Framework* es la base de los cursos de laboratorio de Computación Gráfica (EDAF80) y Computación Gráfica de Alto Rendimiento (EDAN35) de la Universidad de Lund. Fue desarrollado en C ++ y proporciona un motor de renderizado que encontramos fácil de modificar y usar, especialmente, como la base en la que desarrollamos nuestras mejoras.

### 2.2. OpenGL y GLSL

Usamos esta *Application Programming Interface* (API, Interfaz de Programación de Aplicaciones) porque es la base del sistema de renderización del Bonobo Framework, el cual modificamos para desarrollar nuestras mejoras, y por su compatibilidad multiplataforma. La *Open Graphics Library* (OpenGL, Librería Abierta de Gráficos) es una API de computación gráfica de código abierto, multiplataforma, que maneja 2D y 3D, y permite abstraer al programador de interactuar directamente con la Unidad de Procesamiento de Gráficos (GPU) para lograr una renderización acelerada por hardware. Además, proporciona al programador un Canal de Renderizado de Gráficos (*Rendering Pipeline*), que normalmente se implementa directamente en hardware.

Usamos el *OpenGL Shading Language* (GLSL, Lenguaje de *Shading* de OpenGL) para implementar TAA y nuestras mejoras, como es parte del estándar de OpenGL. GLSL es un lenguaje para *shading* de alto nivel, que permite a los programadores un mayor control del Canal de Renderizado de Gráficos, sin requerir el uso del lenguaje ensamblador de OpenGL o lenguajes específicos de cada hardware.

### 2.3. MATLAB

Elegimos MATLAB como el entorno donde desarrollamos nuestro marco de pruebas debido a la alta calidad de sus herramientas, su amplia base de conocimientos y sus capacidades de creación rápida de prototipos.

MATLAB es un entorno de cómputo numérico multi-paradigma propietario. Comúnmente utilizado para ciencia, ingeniería y economía. Es popular para aplicaciones de procesamiento de imágenes debido a su amplia biblioteca de algoritmos para este propósito, incluidas las métricas de imágenes.

# CAPÍTULO III

## MARCO TEÓRICO

En este capítulo, explicaremos toda la información teórica que es la base de esta tesis, desde la teoría de base de Computación Gráfica hasta las Métricas de Imágenes utilizadas.

### 3.1. Canal de Renderizado

El Canal de Renderizado Gráfico actual se puede simplificar en tres etapas: *Vertex Shader*, que procesa la geometría asociada con los vértices y los prepara para el siguiente paso; *Rasterizer*, que mapea los triángulos a píxeles en la pantalla, calcula su visibilidad e interpola los parámetros de los vértices para cada píxel cubierto por el triángulo; y el *Pixel Shader* (o *Fragment Shader*) que toma los píxeles visibles del *Rasterizer* y los colorea. La Figura 3.1 es un ejemplo del Canal de Renderizado simplificado.

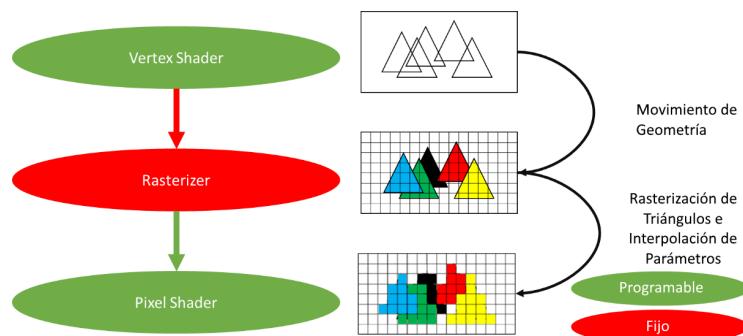


Figura 3.1: Canal de Renderizado, basado en la 5ta clase de EDAF80. [12]

Es importante tener en cuenta que las etapas de *Vertex* y *Pixel Shaders* son controlables por el programador usando programas especiales llamados *Shaders*; estos proporcionan una forma para controlar el hardware de renderizado. Por el contrario, el *Rasterizer* no está controlado por el programador y se maneja por completo mediante una función fija de hardware. [12]

Este proceso de rasterización es importante para nosotros porque de allí donde provienen algunos de los errores corregidos por *Temporal Anti-Aliasing*.

### 3.2. Proceso de Rasterización

Durante el proceso de rasterización, a cada triángulo se le realiza una prueba para establecer qué píxeles está cubriendo. Mientras se hace, cada píxel se está probando para descubrir si está cubierto por otro triángulo.

En la Figura 3.2 tenemos un ejemplo del Proceso de Rasterización. En la imagen de la izquierda, tenemos los triángulos como superficies continuas antes de enviarlos al Rasterizer y, en la imagen de la derecha, tenemos triángulos mapeados a los píxeles en la pantalla después de pasar por el *Rasterizer*.



Figura 3.2: Ejemplo de los resultados del Proceso de Rasterización. *Nota:* se agregaron colores para diferenciar los triángulos, pero solo se agregarán en la etapa del *Pixel Shader*.

Debido a que estamos mapeando un triángulo continuo a un número finito de píxeles, enfrentamos el problema de los píxeles parcialmente cubiertos y cómo determinar cuánto es suficiente para calificarlos como cubiertos, la Figura 3.3 muestra ejemplos de este problema. Esto se resuelve calculando si el centro del píxel está cubierto por la geometría del triángulo. Este proceso es susceptible a errores debido a la precisión de la representación utilizada para los vértices.

Este proceso nos muestra que lo que se renderiza en la pantalla, es una aproximación a lo que se representa en la escena, ya que los píxeles solo pueden ser cubiertos con un triángulo a la vez. [1, 13]

### 3.3. El Problema del Aliasing

Cuando mapeamos una representación continua en una finita, vamos a generar errores. Como explican Edward Angel y Dave Shreiner en su libro (página 413) [3], podemos interpretar el proceso de renderización como el muestreo de una función continua  $f(x, y)$ ,



Figura 3.3: Ejemplo del problema de Cobertura Parcial, basado en la segunda clase de EDAN35. [13]

que representa el color de la escena en ese punto, a una cuadrícula de píxeles  $n \times m$  en la que suponemos que el punto  $f_{ij}$  es el valor de  $f$  sobre un área pequeña; reconstruyendo la función  $f$ , para mostrar la imagen en la pantalla, usando solo lo que sabemos de las muestras finitas. La herramienta matemática utilizada para evaluar los problemas de este proceso es el Análisis de Fourier, que establece que una función se puede descomponer en un conjunto de sinusoides, posiblemente en un número infinito de frecuencias. Para el análisis de imágenes bidimensionales, podemos pensar en la función  $f$  como un conjunto de sinusoides en dos frecuencias espaciales.

Para esta tesis, usaremos la primera parte del Teorema de Muestreo de Nyquist, como una herramienta para ilustrar por qué aparecen problemas de aliasing y se relacionan con problemas de muestreo.

*” Teorema de Muestreo de Nyquist (Parte 1): Las muestras ideales de una función continua contienen toda la información de la función original si y solo si la función continua se muestrea a una frecuencia mayor que el doble de la frecuencia más alta en la función.*

*La Frecuencia de Nyquist se define como la mitad de la frecuencia de muestreo, que es la frecuencia más baja que no puede estar en los datos para evitar el aliasing ”.*

Tomado de la página 415 del libro de Edward Angel y Dave Shreiner. [3]

Como explican Edward Angel y Dave Shreiner, este muestreo idealizado supone que podemos tomar un número infinito de muestras por frecuencia de muestreo, que no podemos hacer en la práctica. El Problema de *Aliasing* que se experimenta en la Computación

Gráfica, proviene de no poder muestrear según lo requerido por el Teorema de Muestreo de Nyquist, creando bordes desiguales que aparecen en el proceso de rasterización (*Aliasing Espacial*), la Figura 3.4 muestra un ejemplo de este tipo de aliasing, y saltos entre objetos en movimiento (*Aliasing Temporal*), de acuerdo con Doggett y Wronski [13, 18]. Se han propuesto y utilizado muchas soluciones para resolverlo, por ejemplo la familia de soluciones *Super Sampling Anti-Aliasing* (SSAA) que trabaja en frecuencias más altas que las requeridas a costa de mayores requerimientos de espacio.



Figura 3.4: Imagen Base de una línea frente a su Aproximación con *Aliasing*.

### 3.4. Mapeado de Sombras y la Arquitectura de Shading Diferido

Como humanos, esperamos que los objetos reaccionen de cierta forma a las luces de una escena, tomando en cuenta la geometría de los objetos, porque las luces y sombras contribuyen con la información espacial a una imagen. En especial, las sombras nos dan una sensación de tamaño y distancia.

Según lo explicado por Doggett [13], bajo de la Canal de Renderizado Gráfico basado en el Rasterizer , el proceso de cálculo de la sombra es difícil de realizar. El *Rasterizer* no sabe si los objetos están cubiertos o no por una fuente de luz, por lo que debemos encontrar un método para calcular si un objeto está en la sombra. Este proceso se denomina Mapeado de Sombras, consiste en representar la escena a través de la perspectiva de cada fuente de luz y luego realizar pruebas en la perspectiva de la cámara, para establecer si el objeto se ve afectado por la luz o si está en sombras.

Pero, como es de esperar, renderizar la escena varias veces es costoso y necesitamos una forma de reducir el costo tanto como podamos. La Arquitectura de *Shading* Diferido proporciona esa solución solo para realizar el *Shading*, utilizando una operación que podría ser costosa, solo píxeles visibles y así evitar desperdiciar recursos realizando el *Shading* a píxeles de geometrías que no son visibles. Esta arquitectura funciona renderizando primero la escena, sin cálculos de *Shading*, en un buffer llamado Buffer de Geometría (*Geometry Buffer*). Allí, se guarda información sobre colores, normales, profundidades, información específica del objeto para interactuar con luces, etc. para uso futuro.

Después de llenar el Buffer de Geometría, llevamos a cabo la técnica de Mapeado de Sombras, que aprovecha la Arquitectura de *Shading* Diferido para calcular la forma en que las luces afectan sólo a los píxeles visibles; calculamos el Mapa de Sombreado de cada fuente de luz realizando cálculos de profundidad únicamente. Posteriormente, calculamos y guardamos el efecto de cada luz usando los Mapas de Sombreado.

Al final, tomamos la información de las luces, sombras y el Buffer de Geometría para renderizar la escena con iluminación.

### 3.5. Anti-Aliasing

Como hemos explicado, hay dos tipos principales de *Aliasing*, Espacial y Temporal. Las soluciones *Anti-Aliasing* proporcionan mejoras contra los artefactos creados por cualquiera de esos tipos a costa de un mayor tiempo de renderizado. Para aplicaciones en tiempo real, este aumento del tiempo de renderización limita qué soluciones Anti-Aliasing son factibles de aplicar.

Otro factor importante para decidir qué técnica *Anti-Aliasing* utilizar, es cómo se comporta con las arquitecturas actuales. Por ejemplo, las primeras soluciones Anti-Aliasing no funcionan con el *Shading* Diferido.

#### 3.5.1. Super Sampling Anti-Aliasing (SSAA)

Esta técnica consiste en renderizar la escena a 4 veces el tamaño de la pantalla y luego promediar píxeles, en un área  $4 \times 4$  alrededor de cada pixel, para calcular el resultado [13]. Proporciona buenos resultados, pero requiere más tiempo de renderizado y un gran uso de memoria.

#### 3.5.2. Multi Sample Anti-Aliasing (MSAA)

MSAA consiste en tomar varias muestras por píxel; en cada muestra, se calculan los valores de profundidad, pero solo se computa un color para el triángulo rasterizado. Esta solución proporciona buenos resultados a costa de un mayor uso de memoria para cálculos de profundidad.

El mayor problema que tiene esta técnica es que no funciona correctamente con el *Shading* Diferido [13]. Esto hace que sea complicado de utilizar con los Canales de Renderización actuales, que normalmente requieren otras correcciones para reducir los artefactos

creados cuando se aplica con *Shading Diferido*.

### 3.5.3. Fast Approximate Anti-Aliasing (FXAA)

FXAA es una técnica de *Anti-Aliasing* que se utiliza durante el posprocesamiento. Esta funciona al detectar bordes en las imágenes renderizadas, para luego suavizarlos. [11]

Es relativamente barata en comparación con MSAA y proporciona resultados relativamente buenos, sus capacidades de suavizado están limitadas por la cantidad de información que la detección de bordes puede obtener en una sola pasada y proporciona resultados relativamente buenos para el aliasing temporal.

### 3.5.4. Enhanced Subpixel Morphological Antialiasing (SMAA)

SMAA es una técnica de post-procesamiento basada en *Morphological Anti-Aliasing* (Anti-Aliasing Morfológico). Funciona mediante la reconstrucción de bordes y sus alrededores para regenerar la información de subpíxel perdida por aliasing. [9]

## 3.6. Temporal Anti-Aliasing

Como explicaron Ke Xu y Lasse Fuglsang en sus respectivas presentaciones [19, 6], el principio básico del *Temporal Anti-Aliasing* es mezclar el cuadro que actualmente está siendo renderizado con cuadros del pasado. Esto se hace para aumentar el número de muestras a través del tiempo en lugar de solo usar muestras del mismo cuadro.

Una de esas técnicas es *Temporal Reprojection Anti-Aliasing* (TRAA, *Anti-Aliasing* de Reproyección Temporal), que funciona guardando los cuadros pasados en un Buffer de Historia para luego reproyectarlos a la escena actual y mezclarlos con el cuadro actual que se está procesando. Para lograrlo, tomamos el cuadro actual y buscamos el color que debería tener en el Buffer de Historia; este paso se llama *Reprojection* (Reproyección).

Para que TRAA funcione, debemos implementar otras técnicas comunes de computación gráfica para que sirvan de base. Necesitamos de *Camera Jittering* (Agitamiento de Cámara) para poder reconstruir la información de píxeles alrededor de los bordes. Un Buffer de Velocidad para determinar las posiciones de los píxeles en el último cuadro, si se movían. Un Buffer de Historia de cuadros donde recopilar los píxeles anteriores para realizar las reproyecciones en el siguiente cuadro. Un *Color Clipping Box* (Caja de Recorte de Colores) para restringir el Buffer de Historia y evitar que aparezca ruido o colores in-

correctos. Un Filtro de Nitidez (*Sharpen Filter*) para reducir parte del desenfoque creado. Y *Motion Blur* (Desenfoque por Movimiento) para corregir los efectos de objetos que se mueven demasiado rápido para el cuadro de recorte de colores.

### 3.6.1. Camera Jittering

*Camera Jittering* (Agitamiento de Cámara) consiste en mover la cámara vertical y horizontalmente utilizando una traslación de subpíxeles, la Figura 3.5 es un ejemplo de la traslación horizontal. Se aplica en cada cuadro para conservar información de fragmentos de regiones locales de superficies en la escena. Si el cuadro actual se deja estático en relación con los pasados, es decir, es idéntico a los cuadros anteriores, el sistema está pierde información subpíxel alrededor de los bordes que podría usarse para refinarla. [6, 19]

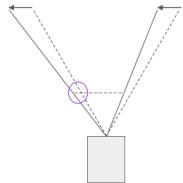


Figura 3.5: Proceso de *Jittering* sobre la proyección de la cámara. Imagen tomada de la presentación de Fuglsand. [6]

El *Jittering* se aplica como una traslación a la matriz de proyección de la cámara usando la *HaltonSequence(2, 3)* (Secuencia de Halton) como los vectores de traslación. Esta secuencia se usa porque genera un patrón irregular para las traslaciones, que ayuda a conservar más información que un patrón regular y porque la Secuencia de Halton, proporciona un generador económico de patrones pseudoaleatorios. [6, 19]

La Figura 3.6 muestra la representación de los 16 puntos utilizados para trasladar la proyección en la implementación actual de la tesis, tal como lo propusieron Fuglsand [6]. Los puntos se generaron usando MATLAB y luego se codificaron para mejorar su aleatoriedad usando reverse-radix scrambling.

### 3.6.2. Buffer de Velocidad

El algoritmo de Buffer de Velocidad (Velocity Buffer) utilizado en esta implementación es el propuesto por Chapman [5], el cual es calculado restando la posición actual del píxel de su posición en el cuadro pasado. Esto es posible ya que guardamos la matriz que



Figura 3.6: Valores usados de la *HaltonSequence(2, 3)* (Secuencia de Halton).

representa cada objeto en la escena y la utilizamos en el siguiente cuadro para calcular los píxeles del Buffer de Velocidad. Además, cancelamos matemáticamente el Cámara Jittering antes de calcular las velocidades, para evitar el ruido que crea, como lo sugiere Xu. [19]

### 3.6.3. Buffer de Historia

Para cada fragmento en el marco actual, buscamos en la vecindad de  $3 \times 3$  y en la vecindad del patrón Cruz (+) alrededor de cada píxel (Ver figura 3.7). Buscamos en ambos patrones para el mínimo y el máximo de colores del cuadro actual, luego los promediamos y los usamos para construir parte del Color Clipping Box restringiendo los píxeles del Buffer de Historia. [6]

En la vecindad  $3 \times 3$  buscamos la velocidad del píxel con la profundidad más cercana a la cámara, esto es para obtener mejores bordes en movimiento para los píxeles que están ocultos detrás de otros [6]. Usamos esta velocidad para reproyectar la posición del píxel del cuadro actual en el Buffer de Historia. [6, 19]

Después de tener los píxeles en el Buffer de Historia, lo restringimos (explicación en la siguiente subsección) y lo interpolamos linealmente con el cuadro actual (Ver la Figura 3.8 para obtener una representación visual del proceso completo). Interpolamos linealmente los píxeles del Buffer de Historia y el cuadro actual usando un valor de retroalimentación que se calcula por la diferencia de luminancia entre los colores del Buffer de Historia restringido y el cuadro actual. Este valor de retroalimentación es sesgado a favor de mantener el color del Buffer de Historia del píxel sobre el color del cuadro actual, esto se hace para agregar información del cuadro actual mientras se mantiene la mayor parte de la historia del píxel.

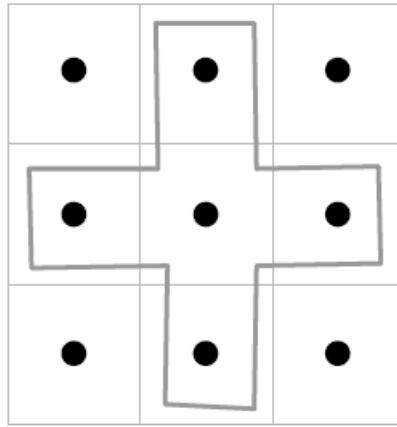


Figura 3.7: Patrón de Muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [6]

Esta interpolación lineal estabiliza la imagen, eliminando el *Jittering* y suavizando los bordes [6, 19]. Dado que el historial de cada píxel se acumula en el Buffer de Historia, obtenemos el efecto que el historial de píxeles de los cuadros anteriores pese menos cuanto más tiempo sea mantenido el historial del píxel dentro Buffer de Historia. [6]



Figura 3.8: Proceso de *Temporal Reprojection Anti-Aliasing*. Imagen tomada de la presentación de Fuglsand [6]

### 3.6.4. Clipping Color Box

Una *Clipping Color Box* (Caja de Recorte de Colores) a es una caja 3D construida utilizando el color de píxel actual como el centro y el color mínimo y máximo calculado en la última subsección como los límites. Se utiliza para manejar el rechazo de la historia

cuando el color del píxel en el Buffer de Historia está demasiado alejado del color actual. Tomamos el color de la historia del como un vector y luego lo proyectamos contra los límites de la caja; si se encuentra fuera del borde de la caja, entonces mantenemos la proyección; de lo contrario, el color de la historia quedará intacto. El uso del Clipping Color Box evita la agrupación de colores en las esquinas que ocurriría si se aplicara *clamping* (Ver Figura 3.9). [6].

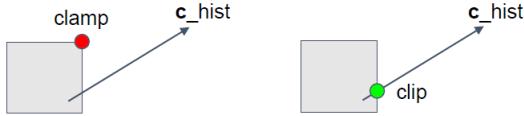


Figura 3.9: *Color Clamping* versus *Color Clipping*. Imagen tomada del *paper* de Fuglsand. [6]

### 3.6.5. Filtro de Nitidez

Como el Proceso de Reproyección y el *Clipping* de Colores crean imágenes borrosas, se requiere un Filtro de Nitidez. Usamos el propuesto por Xu. [19]

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (3.1)$$

La Ecuación 3.1 es la matriz de convolución utilizada como Filtro de Nitidez en el *paper* de Xu. [19]

### 3.6.6. Motion Blur

Debido a la naturaleza del Buffer de Historia, algo de *ghosting* es creado por fragmentos de objetos que se mueven más rápido que el tiempo que tarda el *Clipping Color Box* para rechazar el color de píxeles antiguos, esto es especialmente notable bajo condiciones especiales de luz y fondo. Fuglsand y Xu [6, 19] propusieron usar soluciones como *Motion Blur* para ocultar estos artefactos.

El *Motion Blur* utilizado es el propuesto por Chapman [5]. Intenta comportarse como una cámara real, escalando la velocidad de cada píxel mediante la división de los cuadros actuales por segundo (FPS) a la velocidad deseada, simulando así la velocidad de obturación.

ción. Luego, mezcla los colores de los píxeles que se muestran al seguir la dirección del vector del Buffer de Velocidad.

### 3.6.7. Problemas

Temporal Anti-Aliasing tiene dos inconvenientes principales, el efecto de Ghosting de los objetos en movimiento y el Desenfoque proveniente de la forma en que funciona el Clipping Color Box. Esta tesis pretende ayudar a reducir los efectos de estos dos inconvenientes utilizando nuevos enfoques, pero, por completitud, presentamos algunas de las soluciones actuales disponibles.

#### Desenfoque

Las implementaciones actuales de TAA generan un desenfoque muy agresivo debido a la forma en que mezclan los colores del cuadro actual y el historial; el uso de áreas más grandes que el píxel aumenta los errores generados, por lo tanto, se requiere un Filtro de Nitidez. El filtro aplicado en la implementación es el utilizado por Xu [19], resuelve el desenfoque razonablemente bien, pero no puede eliminar algunos artefactos.

#### Ghosting

Algo de *Ghosting* se crean cuando los objetos se mueven, especialmente bajo condiciones particulares de luz y fondo que hacen que el primer plano y el fondo se vean de manera similar. Esto se corrige parcialmente con *Motion Blur*, sin embargo, parte de él permanece cerca de objetos que se mueven lo suficientemente rápido como para crear algo de *Ghosting* pero lo suficientemente lentos como para evitar el *Motion Blur*. Xu propone el uso de *Motion Blur* y aumentar el tamaño de todo utilizando una técnica *Stencil* y el etiquetado manual de objetos [19], pero nuestro objetivo es evitar que los artistas etiqueten manualmente y prueben objetos por sus comportamientos bajo *Ghosting*. La implementación de Pederson permite el *Jittering* en los cálculos del Buffer de Velocidad para evitar *Ghosting* en las imágenes, pero crea desenfoque no deseado. [6].

### 3.7. Buffer de Acumulación

El Buffer de Acumulación es una técnica de *Anti-Aliasing* que consiste, según Paul Haeberli y Kurt Akeley [8], en renderizar la escena varias veces aplicando *Jittering* a la cámara y luego realizar una suma ponderada a escala de las renderizaciones para generar el cuadro actual.

Este es un proceso que aumenta el muestreo por píxel y reduce los efectos de *aliasing*, lo que produce una imagen de alta calidad a costa de procesar todo varias veces por cuadro.

### 3.8. Operador de Sobel

El Operador Sobel es un operador de gradiente isotrópico  $3 \times 3$  eficientemente computable, como lo explica Irwin Sobel [17]. Usamos este operador para detectar bordes en las imágenes renderizadas y marcarlos como posibles lugares con aliasing. Esto es porque son un lugar común para que aparezcan los artefactos de aliasing.

Funciona al tomar las cuatro estimaciones de gradiente simple central posibles en un vecindario de  $3 \times 3$  y sumarlas. La función de imagen se toma como una función de densidad / intensidad y las cuatro posibles estimaciones como vectores ortogonales que son derivadas direccionales multiplicadas por un vector unitario que especifica la dirección de la derivada. La suma de las cuatro estimaciones posibles de gradiente simple central es equivalente a la suma vectorial de los ocho vectores derivativos direccionales.

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

Sea la Matriz 3.2 sea Vecindario  $3 \times 3$  y  $|G|$  la magnitud de la estimación derivada direccional del vecindario.

La dirección de  $G$  estará dada por el vector unitario asociado al vecino apropiado. La suma vectorial hace que se cancelen todos los valores de  $e$  (centro del Vecindario  $3 \times 3$ ) dejando solo la siguiente expresión 3.3:

$$\begin{aligned} G &= \frac{c-g}{4} * \begin{bmatrix} 1 & 1 \end{bmatrix} + \frac{a-i}{4} * \begin{bmatrix} -1 & 1 \end{bmatrix} + \frac{b-h}{2} * \begin{bmatrix} 0 & 1 \end{bmatrix} + \frac{f-d}{2} * \begin{bmatrix} 1 & 0 \end{bmatrix} \\ &= \left[ \frac{c-g-a+i}{4} + \frac{f-d}{2} \quad \frac{c-g+a-i}{4} + \frac{b-h}{2} \right] \end{aligned} \quad (3.3)$$

Luego multiplicamos por 4 para aproximar el valor y asegurarnos de que no perdamos

precisión si lo realizamos con enteros pequeños de punto fijo. La magnitud recién calculada es dieciséis veces más grande que el gradiente promedio original.

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

La Ecuación 3.4 se puede expresar en dos matrices de ponderación. Usamos la Matriz 3.5 para el componente  $x$  y la Matriz 3.6 para el componente  $y$ .

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad (3.5)$$

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3.6)$$

Para la detección de bordes, lo que se hace comúnmente es comparar la magnitud de  $G$  contra un umbral numérico para marcar píxeles como bordes.

### 3.9. Métricas de Imagen

El proceso de medir la calidad de una imagen es complicado. Como explican Al-Najjar y Soong [2], podemos seguir dos tipos métodos principales: subjetivo u objetivo. Los métodos subjetivos se basan en opiniones recogidas de humanos y, como cabría esperar, se consideran costosos, difíciles de implementar y llevan mucho tiempo. El segundo tipo de métodos, los objetivos, se basan en fórmulas matemáticas y algoritmos para medir la calidad de la imagen sin intervención humana. Para esta tesis, usamos métodos objetivos.

Los métodos de objetivos se pueden categorizar en tres grupos, como describen Al-Najjar y Soong [2]:

- **Sin-Referencia:** En el cual no tenemos una imagen de referencia para comparar.
- **Referencia-Reducida:** Donde tenemos parte de una imagen de referencia.
- **Referencia-Completa:** Tenemos la imagen de referencia completa.

Para la computación gráfica, los métodos preferidos son los de Referencia-Completa porque las imágenes de referencia se pueden generar utilizando algoritmos de mayor calidad pero con mal rendimiento para representarlos. Las métricas más comunes utilizadas,

y las utilizadas en esta tesis, son: Error Cuadrado Medio (MSE, *Mean Square Error*); Relación Señal a Ruido de Pico (PSNR, *Peak Signal-to-Noise Ratio*); y el Índice de Similitud Estructural (SSIM, *Structural Similarity Index*).

### 3.9.1. Error Cuadrado Medio (MSE)

Basada en el promedio del error cuadrado entre los píxeles de la imagen y la referencia.

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

Donde  $N, M$  son el ancho y largo de las imágenes y  $Im, Ref$  son el píxel de la Imagen y la Referencia.

### 3.9.2. Desviación de la Raíz Cuadrada Media (RMSD)

Es la desviación estándar de MSE (*Root Mean Square Deviation*). También llamada *Root Mean Square Error* (RMSE).

$$RMSE = \sqrt{MSE} \quad (3.8)$$

### 3.9.3. Relación Señal a Ruido de Pico (PSNR)

Se basa en el concepto matemático de Relación de Señal a Ruido (SNR, *Signal-To-Noise Ratio*) que mide la señal de la imagen, que se almacena como los colores de los píxeles para nuestros propósitos, contra su error en comparación con la referencia. [2]

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

Donde  $S$  es el valor máximo que puede alcanzar la señal. En nuestro caso es 255 porque usamos canales de color de 8 bits.

### 3.9.4. Índice de Similitud Estructural (SSIM)

SSIM es una métrica de imagen ampliamente utilizada, que coincide con la subjetividad humana y es muy sensible a las degradaciones en la estructura espacial de la luminancia de la imagen, como explican Malpica y Bovik. [14]

Requiere dos imágenes para comparar,  $X$  y  $Y$ , y se calcula en base a tres funciones de similitud en una ventana ponderada gaussiana de  $N \times N$  (típicamente  $11 \times 11$ ).

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

Donde

$$\begin{aligned} \mu_X(x, y) &= \sum_{p=-P}^P \sum_{q=-Q}^Q w(p, q) * X(x + p, y + q) \\ \sigma_X^2(x, y) &= \sum_{p=-P}^P \sum_{q=-Q}^Q w(p, q) * [X(x + p, y + q) - \mu_X(x, y)]^2 \\ \sigma_{XY}(x, y) &= \sum_{p=-P}^P \sum_{q=-Q}^Q w(p, q) * [X(x + p, y + q) - \mu_X(x, y)] \\ &\quad * [Y(x + p, y + q) - \mu_Y(x, y)] \end{aligned}$$

Donde  $w(p, q)$  es una función de ponderación gaussiana tal que  $\sum_{p=-P}^P \sum_{q=-Q}^Q w(p, q) = 1$  y  $C_1, C_2, C_3$  son pequeñas constantes que proporcionan estabilidad cuando el denominador se aproxima a cero. Por lo general, se establecen de la siguiente manera:

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

Donde  $L$  es el rango dinámico de la imagen y  $K_1, K_2 \ll 1$  son pequeñas constantes. Al final, las tres funciones de similitud se combinan en la forma general:

$$SSIM(x, y) = l(x, y) * c(x, y) * s(x, y) \quad (3.11)$$

# CAPÍTULO IV

## DESARROLLO

En este capítulo, se presenta el trabajo principal realizado en esta tesis.

### 4.1. Mejoras al Proyecto de EDAN35

Para esta tesis, decidimos utilizar nuestro proyecto de *High Performance Computer Graphics* (EDAN35) como base. Durante el curso implementamos una técnica de *Temporal Reprojection Anti-Aliasing* (TRAAs, Anti-Aliasing de Reproyección Temporal) basada en las presentaciones de Ke Xu y Lasse Fuglsang [19, 6], de los juegos *Inside* y *Uncharted 4*. Esta resultó ser confiable y bien documentada; y nos permitió poner en práctica los fundamentos de la técnica en un entorno académico, proporcionando la base para las mejoras realizadas para esta tesis.

La implementación del proyecto EDAN35 tenía errores en el Proceso de *Jittering* que se corrigieron al expandir correctamente en lo que ambas implementaciones se referían por *Jittering* de Cámara, consulte el Apéndice B para obtener una explicación completa. El manejo de los puntos de *Halton* se rehizo para lograr la mejora del movimiento de la cámara, se incluyó el soporte de hasta 128 puntos para que funcione como el *Jittering* del Buffer de Acumulación (Ver Figura 4.1). Sin embargo, hay que tener en cuenta que para el *Temporal Anti-Aliasing*, solo los primeros 16 puntos se usan según lo sugerido por Xu y Fuglsang. [19, 6]

El *Anti-Aliasing* de Iluminación Especular, como superficies metálicas, es un problema complejo en sí mismo que requiere soluciones especializadas que funcionen directamente con los reflejos de la luz. Las técnicas *Anti-Aliasing* no lo corrigen este problema por sí mismas, generalmente funcionan en conjunto con soluciones adecuadas ya hechas. Por esto, para evitar problemas con la iluminación especular, se decidió apagarla.

Además, se agregaron modelos de una esfera, pared, tubería, *Hairball* (Bola de Pelo), una ventana con persianas y una ventana arqueada para probar las mejoras realizadas al *Temporal Anti-Aliasing*. Todos los modelos, excepto la pared, se agregaron con una textura



Figura 4.1: Los 128 de la *SecuendiaHalton(2,3)* disponibles para usar.

de color sólido para evitar la introducción de errores de iluminación en los cálculos de las métricas de imagen, ya que estas son usadas para las comparaciones entre la implementación de Uncharted 4 y la desarrollada en esta tesis. El modelo de la pared usa una textura blanca con letras negras, porque las letras usan bordes duros para definir su forma y debe permanecer así después de aplicar cualquier técnica de *Anti-Aliasing*.

#### 4.1.1. Fast Approximate Anti-Aliasing (FXAA)

Para esta tesis, se usó la versión del *white paper* (paper inicial) de *Fast Approximate Anti-Aliasing* (FXAA, *Anti-Aliasing* Aproximado Rápido) de Lottes [11] para compararla con *Temporal Anti-Aliasing*, basándose en que ambas técnicas son *Anti-Aliasing* y post-procesamiento, y que FXAA es una técnica popular utilizada en la industria. Esta técnica se implementó bajo el preajuste de mayor calidad según el *white paper* sin tener en cuenta el impacto en el rendimiento porque queríamos comparar la mejora cruda que ambas técnicas pueden proporcionar.

#### 4.1.2. Enhanced Subpixel Morphological Antialiasing (SMAA)

Con el fin de probar una técnica de post-procesado más compleja y más nueva, se implementó SMAA siguiendo las instrucciones proporcionadas por Jiménez et al [9]. Se utilizó el valor preestablecido más alto que funciona con la Arquitectura de *Shading* Diferido.

#### 4.1.3. Buffer de Acumulación

Usamos un Buffer de Acumulación para proporcionar una imagen de referencia de la escena. Fue implementado siguiendo a Haeberli y Akeley [8]. Los puntos usados para realizar *jittering* a la cámara son los mismos puntos de la Secuencia de Halton que usa *Temporal Anti-Aliasing*, aunque el Buffer de Acumulación puede usar hasta 128. Las razones para usar la Secuencia de Halton son las siguientes: cumple con los requisitos establecidos por Haeberli y Akeley; es fácil ampliar el sistema de cámara actual para admitir más puntos Halton; e, igual que para *Temporal Anti-Aliasing*, proporciona puntos pseudoaleatorios que no siguen un patrón para ayudar a reunir tanta información de la escena como sea posible.

#### Número de Muestras Seleccionadas

El número de muestras seleccionadas para el Buffer de Acumulación es 128. Esto se debe a que proporciona la mejor representación posible de la escena a pesar de que causa una pérdida sustancial de rendimiento.

Para mostrar la diferencia entre el uso de 16 y 128 muestras, realizamos cuatro pruebas, denominadas de A a D, para observar cómo se comportan las medidas bajo diferentes arreglos de objetos e iluminación en la escena. En la tabla 4.1 vemos los resultados de una de esas pruebas:

Tabla 4.1: Comparación del comportamiento de las métricas entre el uso de 16 muestras frente a 128 para el Buffer de Acumulación.

Prueba D				
Muestras	16	128	Diferencia	Diferencia Relativa (%)
Pruebas				
MSE de Temporal	40.647863	38.947297	-1.700566	4.183654132 %
RMSD de Temporal	6.375568	6.240777	-0.134791	2.114180258 %
MSE de No AA	24.872104	24.524992	-0.347112	1.395587603 %
RMSD de No AA	4.987194	4.952271	-0.034923	0.700253489 %
Peak-SNR de Temporal	32.040426	32.22603	0.185604	0.575944353 %
SNR de Temporal	30.30596	30.492374	0.186414	0.611346299 %
Peak-SNR de No AA	34.173678	34.234715	0.061037	0.178289786 %
SNR de No AA	32.439212	32.501058	0.061846	0.190289190 %
SSIM de Temporal	0.993302	0.993451	0.000149	0.014998223 %
SSIM de No AA	0.996826	0.996891	6.5E - 05	0.006520272 %

Los cambios son lo suficientemente grandes en algunas métricas para ser perceptibles, especialmente en MSE y SSIM de Temporal Anti-Aliasing para nuestras comparaciones entre técnicas *Anti-Aliasing*. Vamos a notar los efectos del uso de 128 muestras cuando lleguemos a las comparaciones entre las técnicas *Anti-Aliasing*.

## 4.2. Implementación del Marco de Pruebas

Para medir cualquier mejora lograda, desarrollamos un marco de prueba que nos permite guardar la información importante cuando se realiza una prueba. El marco nos permite seleccionar qué técnica utilizar como el renderizador principal: la implementación de *Temporal Anti-Aliasing* de la Tesis, *Temporal Anti-Aliasing* de *Uncharted 4*, Enhanced Subpixel Morphological Anti-Aliasing (SMAA) o *Fast Approximate Anti-Aliasing* (FXAA). También nos permite hacer zoom en cualquier parte de la pantalla y luego realizar todos los cálculos de métricas de imágenes usando MATLAB.

Cuando se realiza una prueba, las imágenes renderizadas seleccionadas se guardan como PNG's con 4 canales de color y sin compresión. Además, información básica sobre la fecha en que se realizó la prueba, la información de la cámara y los datos de a los valores utilizados para el *Temporal Anti-Aliasing* se guardan en un archivo de texto sin plano.

Para cuantificar si se lograron los objetivos propuestos de esta tesis, sobre *ghosting* y desenfoque, desarrollamos dos tipos diferentes de pruebas: Pruebas Estáticas y Pruebas de *Ghosting*.

### 4.2.1. Pruebas Estáticas

Este tipo de prueba consiste en dejar que el Buffer de Historia se llene de 16 cuadros de la escena, sin ningún objeto en movimiento, utilizando la técnica *Temporal Anti-Aliasing* seleccionada para luego guardar el último cuadro renderizado. Inmediatamente después de guardar el último cuadro de TAA, renderizamos el último cuadro nuevamente pero ahora utilizando el Buffer de Acumulación, para generar la imagen base de la escena. También, renderizamos el último cuadro usando SMAA, FXAA y *No Anti-Aliasing* (NOAA, No AA), para propósitos de comparación.

#### 4.2.2. Pruebas de Ghosting

Este tipo de prueba se realiza solo con los modelos de la Esfera y *HairBall*; el primero se mueve a través del pasillo de la escena, simulando un objeto en movimiento en una aplicación, y el segundo gira en una posición estática, simulando muchos bordes en movimiento. La prueba consiste en renderizar la escena por un número seleccionado de cuadros, con la implementación de la Tesis de *Temporal Anti-Aliasing* y la implementación de Uncharted 4 de *Temporal Anti-Aliasing* al mismo tiempo. Después de renderizar cada cuadro, se guardan cada imagen, la posición de la esfera y la rotación de *Hairball*.

Una vez que el número seleccionado de cuadros se ha renderizado, la esfera vuelve a su posición original y el movimiento se repite utilizando las posiciones guardadas anteriormente. *Hairball* también vuelve a su rotación original y el movimiento también se repite. La diferencia esta vez es que cada cuadro se procesa utilizando el Buffer de Acumulación y luego se guarda, se hace para evitar la gran pérdida de rendimiento causada, que afecta las técnicas de *Temporal Anti-Aliasing*.

Después de guardar todas las imágenes renderizadas, comparamos las producidas por el *Temporal Anti-Aliasing* de la Tesis y el *Temporal Anti-Aliasing* de Uncharted 4 contra la imagen base para calcular las métricas de la imagen de ellos. Estas métricas muestran cuánto error se generó por efecto de *ghosting* en ambas implementaciones, lo que nos permite comparar cómo se comportan ambas técnicas.

#### 4.2.3. Métricas de Imagen de MATLAB

Las métricas de las imágenes para cada prueba representan, numéricamente, la calidad de cada imagen. Los calculamos para comparar cómo se comportó cada técnica en cada prueba para identificar si el *Temporal Anti-Aliasing* de la Tesis genera imágenes de mayor calidad que las otras técnicas probadas.

Una vez que se guardan todos los resultados de la prueba, una secuencia de comandos toma todas las imágenes y las organiza en carpetas por nombre y tipo de prueba. Después, todos los resultados de la prueba se procesan utilizando MATLAB para obtener los resultados de las métricas de las imágenes. Con los resultados de las métricas de imagen, comparamos cómo se comportó el *Temporal Anti-Aliasing* de la Tesis con el *Temporal Anti-Aliasing* de Uncharted 4 y las otras técnicas *Anti-Aliasing*, para detectar si las mejoras funcionaban y observar cómo se comportó nuestra implementación mejorada frente a las demás.

Para las Pruebas Estáticas, realizamos mediciones de MSE, RMSD, PSNR, SNR y SSIM en las imágenes renderizadas de TAA, FXAA, SMAA y No AA, utilizando las imágenes renderizadas del Buffer de Acumulación como referencia para comparar. Asimismo, generamos un mapa local de SSIM para cada imagen renderizada, además del renderizado con el Buffer de acumulación, que se guardan como PNGs y FIGs de MATLAB. Todos los resultados de las mediciones se almacenan en la carpeta con los resultados de la prueba como texto plano.

Para Pruebas de Ghosting, realizamos mediciones de MSE, RMSD, PSNR, SNR y SSIM en cada cuadro procesado con la técnica de Temporal Anti-Aliasing de Uncharted 4 y el Temporal Anti-Aliasing de la Tesis, utilizando el cuadro renderizado del Buffer de Acumulación correspondiente. Se genera un mapa local de SSIM para cada imagen renderizada, exceptuando las del Buffer de Acumulación, y se guardan como PNG y FIGs de MATLAB. Todos los resultados de todas las imágenes se almacenan en un archivo de texto plano. En los mapas locales de SSIM, las diferencias son representadas como colores, el blanco simbolizando la similitud.

#### 4.3. Modificaciones de Temporal Reprojection Anti-Aliasing

Para esta tesis, la técnica del *Color Clipping Box* se modificó para que se viera afectada por los valores calculados a partir de las nuevas técnicas aplicadas en esta tesis. Estos cambios siguen la lógica de que queremos aplicar toda la fuerza de la técnica de *Temporal Anti-Aliasing* cuando sea necesario y minimizar la aplicación de la técnica en otras partes, para minimizar los efectos de la *ghosting* y el desenfoque.

El primer cambio consiste en que los colores que se calcularon a partir del promedio entre las vecindades  $3 \times 3$  y Cruz, de los patrones de muestreo (ver Figura 4.2), ahora se mezclan en una cantidad variable. Esto sigue la idea de que preferimos la vecindad de Cruz si el píxel que estamos calculando actualmente no se considera con *aliasing*, porque es menos probable que la vecindad de Cruz introduzca ruido en los cálculos de la *Color Clipping Box*. Pero, si se considera que el píxel con *aliasing*, preferimos el vecindario  $3 \times 3$  porque proporciona más información sobre el entorno del píxel para crear la imagen sin *aliasing*.

El segundo cambio consiste en que el tamaño de la *Color Clipping Box* depende de cuánto *aliasing* se considera que el píxel tiene. Al utilizar *Color Clipping Box* más pequeñas (Ver Figura 4.3) que la técnica original en píxeles sin *aliasing*, aumentamos la eliminación de colores no deseados del historial, reduciendo los efectos del efecto *ghosting* ya que rechazamos colores más rápido en píxeles que sabemos que no se consideran con *aliasing*.

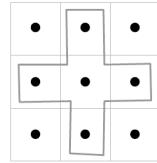


Figura 4.2: Patrón de muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [6]

Esto se implementa mediante la interpolación lineal del color de píxel actual y los colores mínimo y máximo calculados para construir la *Color Clipping Box*.

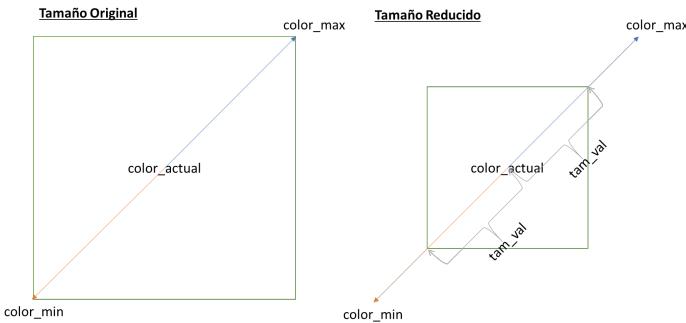


Figura 4.3: Reducción de tamaño de la Color Clipping Box

En las siguientes subsecciones, explicaremos cómo se calculan todos los valores que contribuyen a decidir si un píxel se considera con *aliasing* o no. Todos estos valores se calculan al mismo tiempo que se aplica TAA, con la excepción del Operador de Sobel, que ocurre antes de la técnica TAA principal. Después, mostramos cómo usamos ese valor para reducir el tamaño del *Color Clipping Box* y la preferencia del patrón de muestreo.

#### 4.3.1. Implementación de Mejoras de Indexación de Triángulos

La idea principal detrás de la aplicación de esta técnica es detectar los píxeles que consideramos con *aliasing* utilizando la cantidad de modelos diferentes que rodean a un píxel. Queremos detectar los bordes entre diferentes modelos porque el *aliasing* normalmente ocurre allí. Una vez que tenemos esta información, procedemos a modificar la *Color Clipping Box* que controla la aplicación de TAA.

Para implementar esta técnica, todos los modelos en la escena reciben un índice único. Luego, en el pase de Renderización de Geometría, todos los triángulos que pertenecen a un mismo modelo reciben el índice del modelo como su ID. Posteriormente, se usa en el

*píxel shader* para generar una textura en la que cada píxel contiene el ID del modelo al que pertenece.

En el pase de Reproyección Temporal, se calcula el promedio del número de píxeles pertenecientes a diferentes modelos en la vecindad  $3 \times 3$  del píxel siendo revisado. Con este promedio, procedemos a sesgar la interpolación lineal de color entre el mínimo, máximo y promedio entre las vecindades Cruz y  $3 \times 3$  del píxel y procedemos a cambiar el tamaño del *Color Clipping Box*.

Si el promedio es cercano a cero, lo que significa que el píxel está rodeado por píxeles de su mismo modelo, interpolamos hacia los colores de la vecindad Cruz y reducimos el tamaño de la *Color Clipping Box*. Pero, si el promedio es cercano a uno, lo que significa que el píxel está rodeado por muchos píxeles de otros modelos, interpolamos hacia los colores de la vecindad  $3 \times 3$  y dejamos que la *Color Clipping Box* permanezca en su tamaño original.

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

Donde

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

A continuación, vamos a mostrar algunos ejemplos de cómo funciona esta técnica. Para cada matriz, los números representan los ID's de los triángulos, siendo la posición central el píxel que actualmente se está calculando y el resto siendo su vecindad. La Matriz 4.2 muestra un ejemplo de un píxel no considerado con *aliasing* porque la mayoría de sus vecinos tienen el mismo ID. Por otro lado, la Matriz 4.3 muestra un pixel considerado con *aliasing* debido a la gran cantidad de ID's diferentes alrededor.

$$\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 5 & 23 & 82 \end{bmatrix} \quad (4.2)$$

$$\begin{bmatrix} 3 & 3 & 3 \\ 3 & 5 & 3 \\ 5 & 23 & 82 \end{bmatrix} \quad (4.3)$$

#### 4.3.2. Pseudo-Varianza de Profundidad y Pseudo-Varianza Temporal de Profundidad

La razón detrás de esta técnica es que queremos detectar píxeles que están en una vecindad de píxeles separados por profundidades relativamente largas. También, queremos detectar píxeles cuya profundidad en el último cuadro cambió, relativamente, una distancia larga en contraste con su vecindad actual. Todos los píxeles que detectamos los consideramos con *aliasing*.

Primero, calculamos la profundidad lineal mínima, máxima y su promedio del vecindario  $3 \times 3$ . Luego procedemos a usar la siguiente fórmula para calcular el valor que vamos a usar para normalizar los resultados:

$$\begin{aligned} maxDepthDistance = \min ( |depthMin - depthAvg|, \\ |depthMax - depthAvg| ) \end{aligned} \quad (4.4)$$

Calculamos el valor de normalización utilizando el mínimo para evitar la interferencia de valores atípicos. Si  $maxDepthDistance$  está por debajo de 0.002, todo lo que sigue se establece en cero porque los píxeles están tan cerca que probablemente no tengan aliasing. Este umbral se definió experimentando qué valores no generan ruido dentro de los cálculos, pero dejan entrar los píxeles interesantes.

Luego calculamos la Pseudo-Varianza de Profundidad como:

$$depthPseudoVariance = \left( \frac{|currentDepth - depthAvg|}{maxDepthDistance} \right)^2 \quad (4.5)$$

Esto nos proporciona una Pseudo-Varianza que mide la distancia entre la profundidad promedio de la vecindad y la profundidad actual del píxel. Hay que tener en cuenta que normalmente el valor va a ser entre 0 y 1 pero, si la profundidad del píxel es un valor atípico en la vecindad, este valor va a ser superior a 1.

Finalmente, calculamos cómo la profundidad del píxel en el último cuadro se relaciona con la vecindad actual, usando la Pseudo-Varianza Temporal de Profundidad. Hay que tener en cuenta que usamos la cuarta potencia para reducir el ruido en los cálculos; dado que el valor normalizado está entre 0 y 1, esto lo hace converger a 0 si el valor era cercano a 0 o a 1 si es cercano a 1.

$$depthTemporalPseudoVariance = \left( \frac{|pastDepth - depthAvg|}{maxDepthDistance} \right)^4 \quad (4.6)$$

A continuación, vamos a mostrar ejemplos de cómo esta técnica decide si un píxel se

considera con aliasing o no. Para cada matriz, los números representan las profundidades de los triángulos, siendo la posición central el píxel actual que se analiza y el resto es su vecindad. La Matriz 4.7 nos muestra un ejemplo de un pixel no considerado con aliasing porque está relativamente cerca de la mayoría de sus vecinos. Por el contrario, la Matriz 4.8 muestra un ejemplo de un píxel que se consideraría con aliasing porque tiene una distancia relativamente grande en comparación con su vecindad. Un ejemplo de un píxel considerado con aliasing por la Pseudo-Varianza Temporal de Profundidad, sería si la profundidad del píxel de la Matriz 4.7 en el último cuadro fuera de 4.0.

$$\begin{bmatrix} 9.0 & 9.3 & 8.7 \\ 9.2 & 9.0 & 9.3 \\ 8.8 & 8.9 & 8.7 \end{bmatrix} \quad (4.7)$$

$$\begin{bmatrix} 9.0 & 9.3 & 8.7 \\ 9.2 & 4.0 & 9.3 \\ 8.8 & 8.9 & 8.7 \end{bmatrix} \quad (4.8)$$

#### 4.3.3. Implementación de las Mejoras de Sobel

La idea principal detrás de la aplicación de la técnica de detección de bordes de Sobel es concentrar los efectos TAA en píxeles en los bordes para corregir el *aliasing*, si es necesario. Es importante señalar que esta es la única técnica de las mejoras de la tesis que se ejecuta antes del algoritmo TAA principal.

Aplicamos el Operador Sobel a la luminancia de los colores de la escena iluminada producida por la Arquitectura de *Shading* Diferido, la luminancia de los colores de la escena no iluminada del Buffer de Geometría y la profundidad lineal actual. Utilizamos la luminancia porque el ojo humano reconoce mejor los cambios repentinos y usamos la escena, iluminada y no iluminada, para evitar problemas al detectar bordes debido a luces o sombras.

Cada operador de Sobel se realiza por separado, y sus magnitudes se mezclan al final de la siguiente manera:

$$g = (u * 0.3 + l * 0.7) + d \quad (4.9)$$

Donde  $u$  es la magnitud del Operador Sobel de la luminancia de la escena no iluminada,  $l$  es la magnitud del Operador Sobel de la luminancia de la escena iluminada y  $d$  es la magnitud del Operador Sobel de la profundidad lineal actual.

Luego, restringimos(*clamp*)  $g$  entre 0.0 y 1.0 para finalmente aplicar el polinomio de paso suave (*smoothstep*) de la siguiente manera:

$$sobel = \sqrt{g^2 * (3.0 - 2.0 * g)} \quad (4.10)$$

Después de aplicar todos los Operadores de Sobel, guardamos los resultados en una textura y realizamos una versión simplificada de TRAA para mantener los resultados estables a lo largo del tiempo. Esta versión simplificada es casi la misma que la que usamos como base de esta tesis, las diferencias provienen del uso de *Clamping* (Restricción) en lugar de una *Color Clipping Box*, porque los valores de textura son unidimensionales; y que no aplicamos un Filtro de Nitidez.

La salida de este TRAA se usa para calcular el valor de Sobel del píxel actual, el valor promedio de Sobel en la Vecindad Cruz y el valor promedio de Sobel en la Vecindad  $3 \times 3$ .

#### 4.3.4. Mezcla Final

Finalmente, modificamos cómo se calcula la *Color Clipping Box* utilizando los valores que calculamos previamente. Llamamos a este valor mezclado final *aliasedValue*, el cual representa la cantidad de un pixel que se considera con *aliasing*. Después de calcularlo, lo usamos para cambiar el patrón de muestreo a partir del cual se construye la *Color Clipping Box* y su tamaño.

Primero, definimos la función de mezcla de la siguiente manera:

$$Mix(x, y, t) = x * (1 - t) + y * t \quad \text{with } 0 \leq t \leq 1 \quad (4.11)$$

Luego, la mezcla final se aplica de la siguiente manera:

$$sobelAvgMixVal = Clamp01(modelAverage_i + sobel) \quad (4.12)$$

Donde  $sobel$  es el valor Sobel del píxel actual, de la Ecuación 4.10,  $modelAverage_i$  proviene de la Ecuación 4.1 y *Clamp01* es la función de Clamping (Restricción) entre 0 y 1.

$$sobelAvg = Mix(sobelAvgCross, sobelAvg3x3, sobelAvgMixVal) \quad (4.13)$$

Donde  $sobelAvgCross$  es el valor promedio de Sobel de la Vecindad Cruz y  $sobelAvg3x3$

es el valor promedio de Sobel de la Vecindad  $3 \times 3$  alrededor del píxel actual.

Y, utilizamos las Ecuaciones 4.13, 4.5, 4.6 y 4.1 para calcular cuánto *aliasing* se considera este píxel tiene:

$$\begin{aligned} \text{aliasedValue} = & \text{Clamp01}(\text{sobelAvg} + \text{depthPseudoVariance} \\ & + \text{depthTemporalPseudoVariance} \\ & + \text{modelAverage}_i) \end{aligned} \quad (4.14)$$

Con este *aliasedValue* (valor de *aliasing*) procedemos a modificar la *Color Clipping Box*. Primero, seleccionamos cuánto de cada patrón de muestreo queremos que sea parte del *Color Clipping Box*:

$$\begin{aligned} \text{colorMin} = & \text{Mix}(\text{colorMinCross}, \text{colorMin3x3}, \text{aliasedValue}) \\ \text{colorMax} = & \text{Mix}(\text{colorMaxCross}, \text{colorMax3x3}, \text{aliasedValue}) \\ \text{colorAvg} = & \text{Mix}(\text{colorAvgCross}, \text{colorAvg3x3}, \text{aliasedValue}) \end{aligned} \quad (4.15)$$

Luego modificamos el tamaño del *Color Clipping Box* mediante la interpolación hacia el tamaño normal si el *aliasedValue* está cerca de uno; en caso contrario, usamos el color actual, que es el centro del cuadro, lo cual disminuye el tamaño.

$$\begin{aligned} \text{clipColorMin} = & \text{Mix}(\text{colorCurrent}, \text{colorMin}, \text{aliasedValue}) \\ \text{clipColorMax} = & \text{Mix}(\text{colorCurrent}, \text{colorMax}, \text{aliasedValue}) \end{aligned} \quad (4.16)$$

Las Figuras 4.4, 4.5 y 4.6 son ejemplos de los valores de *aliasing* calculados, cada píxel representa el *aliasedValue* actual de esa imagen. El color blanco representa un *aliasedValue* de 1 y el color negro de 0. Como esperamos, la mayoría de los bordes están marcados como probablemente con *aliasing* por las técnicas que utilizamos.



Figura 4.4: Imagen hecha de los valores de *aliasing* de cada píxel.

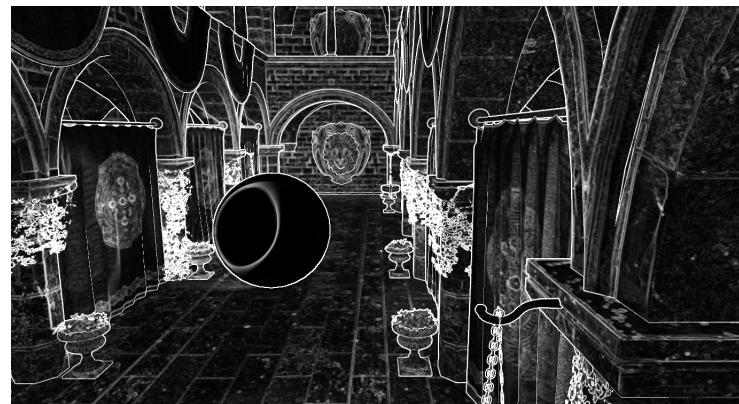


Figura 4.5: Imagen hecha de los valores de *aliasing* de cada píxel.

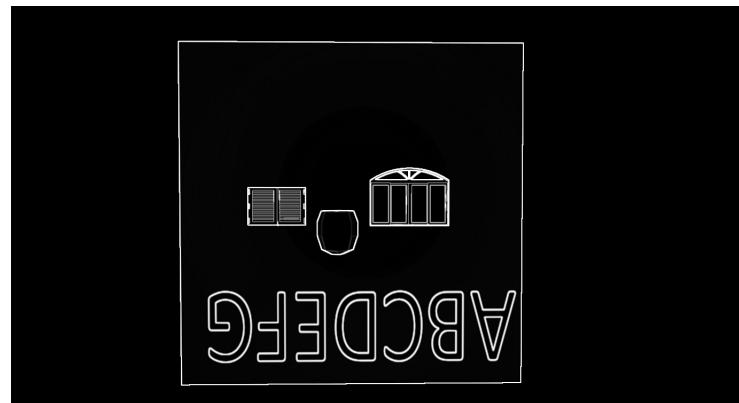


Figura 4.6: Imagen hecha de los valores de *aliasing* de cada píxel.

#### 4.3.5. Modificaciones del Filtro de Nitidez

Con el Filtro de Nitidez actual, se crea un pico de color cuando el píxel central es brillante y los píxeles vecinos están oscuros, esto ocurre porque los colores oscuros son cercanos a cero y no son capaces de cancelar el aumento al píxel central, el cual se multiplica por 5. Un ejemplo del peor caso posible es tener el píxel central con un color brillante y los píxeles del vecindario como negro puro, que se representan como cero. Por lo tanto, hemos normalizado el Filtro de Nitidez para evitar crear ese pico de color y trabajar mejor con píxeles brillantes con vecindarios oscuros. El filtro se cambió de 3.1 a:

$$\begin{bmatrix} 0 & -0.25 & 0 \\ -0.25 & 2 & -0.25 \\ 0 & -0.25 & 0 \end{bmatrix} \quad (4.17)$$

La Ecuación 4.17 muestra la nueva matriz de convolución del Filtro de Nitidez utilizada.

# CAPÍTULO V

## RESULTS

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

### 5.1. Evaluation Methodology

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

The models used on the tests were:

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

It is important to note that in Computer Graphics there are only common models used to test techniques, there is no standard per se. On this Master Thesis, we used two common models: the Sponza Atrium, which has many variations and it is commonly used to test many Computer Graphics techniques; and Hairball, which is sometimes used to test Ray and Path Tracing techniques. As well, an explanation why each model was selected is available under each test subsection.

Note that Sponza Atrium and Hairball models were downloaded from Morgan McGuire's Computer Graphics Archive [15]; the Pipe model is from Spencer Arts [4]; the Arched Window model is from Isabela H. [7]; and the Window with Blinds is from Channa Yim [21].

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

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

## 5.2. Results and Comparisons

First, we need to know that the best possible value for MSE and RMSD is zero, meaning that there is no error; that having a high PSNR and SNR value means it is better because the noise, which is the denominator in the equation of this metric, is close to zero; and, finally, that having an SSIM of 1 means that the image is structurally the same as the ground truth, while having a value of 0 means that it is structurally different. For the SSIM maps, each pixel represents its SSIM value; having a white color means that is structurally the same while having a darker color means that there are structural differences.

### 5.2.1. Sharpen Filter

For this test, we rendered the Sponza Atrium Model and a Static Sphere Model to evaluate the effects of the Normalized Sharpen Filter to the general quality of the rendered image, especially, regarding the blurring normally caused by TAA. Everything was rendered using both implementations of TAA with and without the Normalized Sharpen

Filter to see if this change was the only improvement that was increasing the quality of the rendered image. The scene was selected for the test because it provided an example of a general scene that should not generate any blur. As we see from the Table 5.1, even if the Master Thesis TAA and the Uncharted TAA used the new Sharpen Filter, our other improvements still rendered a higher quality image. If we zoom on Figure 5.1 From Figure 5.2 we can see that the Normalized Sharpen Filter contributes reducing the artifacts in the rendered image, as there are almost no dark areas on the SSIM maps of the TAA's using it.

Tabla 5.1: Sharpen Filter Test numerical results

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

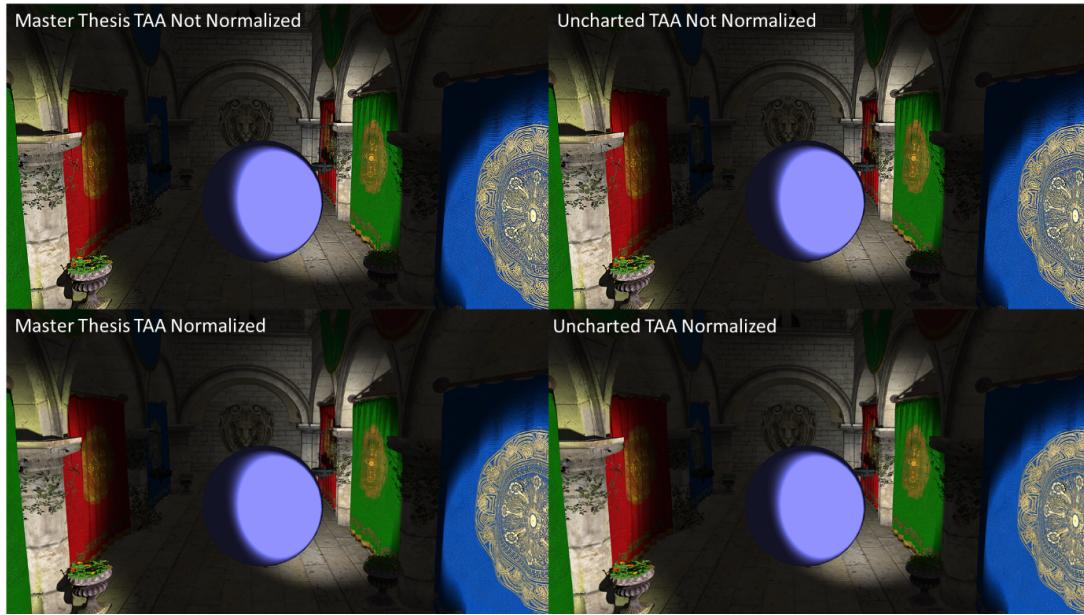


Figura 5.1: Rendered Images comparison.



Figura 5.2: SSIM Maps comparison.

### 5.2.2. Pipe

For this test, we rendered the Pipe Model in order to test how the improvements behaved when rendering a model with hard edges. We wanted to test if our improvements

reduced the amount of blur around those hard edges while still fixing the aliasing problem. We rendered the Pipe twice, the first time using a regular camera angle, for normal aliasing around the edges, and a skewed camera angle, for increased aliasing effects. As we see from the results, TAA with our improvements is at the same quality level as SMAA.

## Regular

When we zoom and compare Figure 5.4 with the rendered images in Figure 5.3, especially around the edges, we observe that there is a reduction of blurring in the Master Thesis TAA in comparison with the Uncharted TAA. As well, we observe that the Uncharted TAA generates bright colors around the edges which should not be there; this is easier to observe as the dark edges on the SSIM map of the Uncharted TAA on Figure 5.3. Furthermore, Table 5.2 confirm us that the Master Thesis TAA reaches almost the same quality as SMAA.

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

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

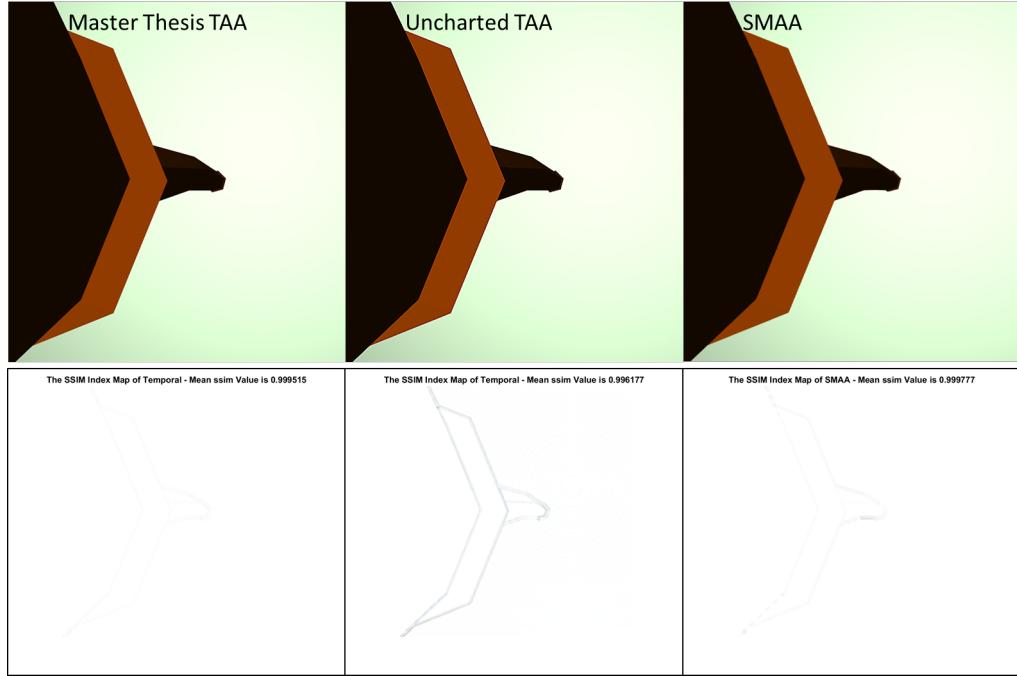


Figura 5.3: Pipe Regular comparison between Master Thesis TAA, Uncharted TAA, and SMAA.



Figura 5.4: Pipe Regular Test ground truth.

### With Camera Inclination

Zooming and comparing the Figures 5.5 and 5.6, we can observe that the Master TAA is the most similar to the Ground Truth. From the SSIM map of the Uncharted TAA, we notice that blurring is being generated around the edges.

Finally, from Figure 5.5 and Table 5.6 we note that SMAA is not properly detecting the upper edge of the pipe, when zoomed we observe a small staircase forming around the

edge. This is due to the fact that the camera was set up with a skewed inclination which pushed to the limit the edge detection techniques used in SMAA.

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

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

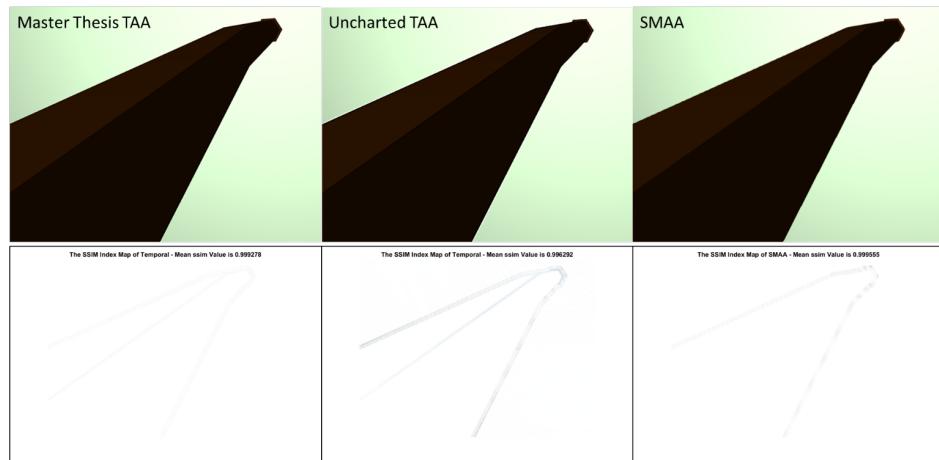


Figura 5.5: Pipe with Camera Inclination comparison between Master Thesis TAA, Uncharted TAA, and SMAA.



Figura 5.6: Pipe with Camera Inclination Test ground truth.

### 5.2.3. Window with Blinds

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

From Figures 5.8 and 5.7 we can observe it is complicated for the techniques to handle the small gaps between the blinds. From Figure 5.8 and Table 5.4, we notice that the Master Thesis TAA and SMAA are able to reconstruct more details than the Uncharted TAA but, aesthetically and visually, we believe it is better the Uncharted TAA than the other techniques because those small gaps between the blinds flicker less when moving.

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

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

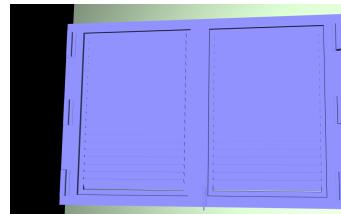


Figura 5.7: Window with Blinds ground truth.

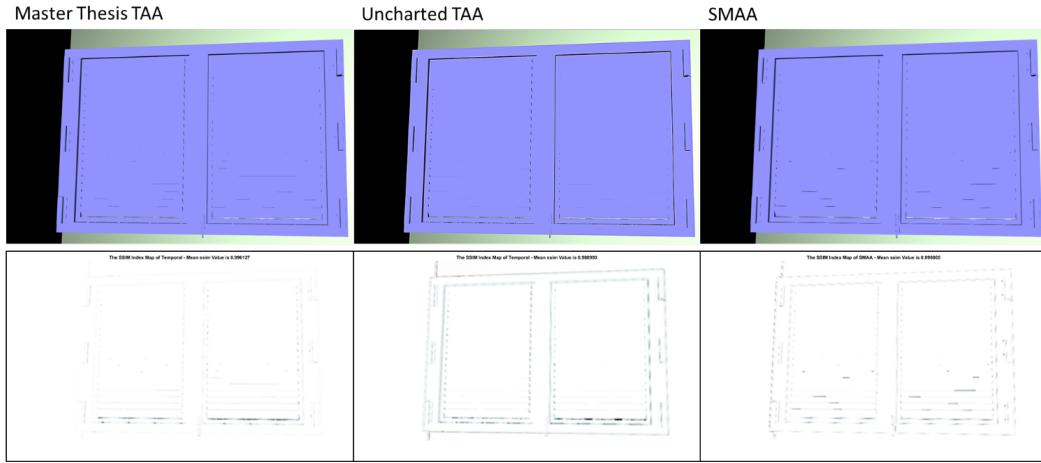


Figura 5.8: Window with Blinds comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

#### 5.2.4. Arched Window

We used the Arched Window Model to test how our improved implementation with the small details of the window's door and the aliasing from the arch. We can observe from Figures 5.9 and 5.10 that for all the techniques, the small gaps around the window's door are hard to render. On some parts small gaps, we see that the techniques could not render it completely. Even though on Table 5.5 SMAA appears to be the best, but we believe that the Uncharted TAA has the best visual quality because those incomplete gaps generate flickering when there is movement.

Tabla 5.5: Numerical results of the Arched Window Test.

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



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

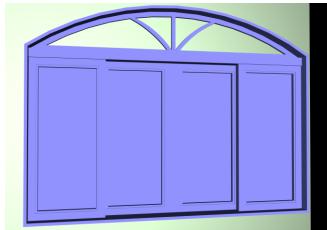


Figura 5.10: Arched Window ground truth.

### 5.2.5. Sponza Atrium

For this test, we wanted to analyze how the Master TAA implementation behaved with a general scene with lights and shadows, we used the Sponza Atrium Model with the Sphere Model being static in the center.

If we compare Figures 5.11 and 5.12, we can observe that the Uncharted TAA had blurring problems around all the edges, which is visible on its SSIM map; SMAA had problems with all the flowers, we can observe the flower shape on its SSIM map; and that the Master Thesis TAA only had minor problems with the flowers, as seen on the gray areas on its SSIM map. Furthermore, Table 5.6 confirm what we are observing on the visual results, as the Master Thesis TAA got the best scores on most of the test; the Uncharted TAA got the worst scores due to the edges problems; and SMAA got worse than normally scores due to the flowers problem.

Tabla 5.6: Numerical results of the Sponza Atrium Test.

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

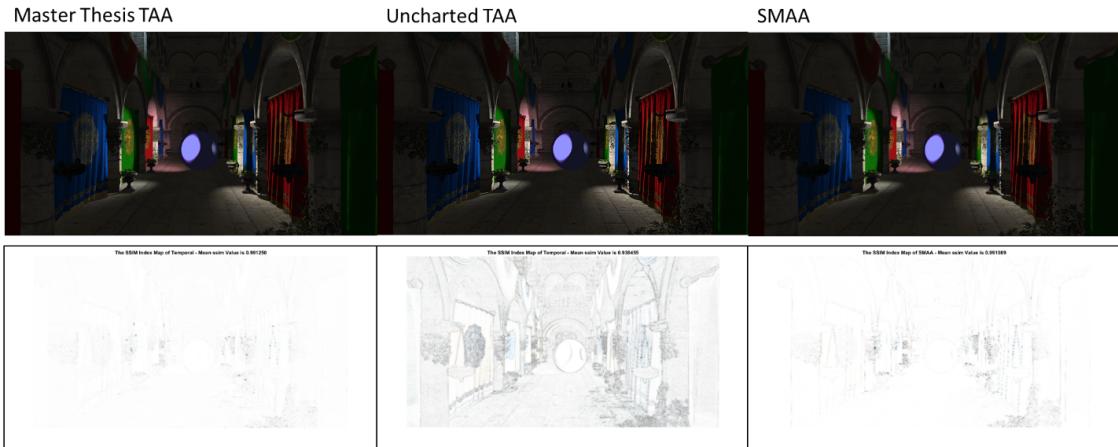


Figura 5.11: Sponza Atrium comparison between Master Thesis TAA, Uncharted TAA, and SMAA.



Figura 5.12: Sponza Atrium ground truth.

### 5.2.6. Sponza Atrium Flowers

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

From Figures 5.13 and 5.14, we observe the Uncharted TAA has many problems handling this type of model, especially if we look at its SSIM map; we notice that SMAA could not correct all the aliasing artifacts from the edges of the flowers, we see the edges of the flowers appear on its SSIM map; and, finally, we observe that the Master Thesis TAA corrected the most aliasing artifacts, some are still left as seen on its SSIM map on the gray areas. Furthermore, Table 5.7 confirm what we observe visually, as the Master TAA got the best scores and SMAA scored worse than average.

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

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



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



Figura 5.14: Sponza Atrium Flowers ground truth.

### 5.2.7. Hard Edges

For this test, we explored how the Master Thesis Implementation with many small details at a far distance and how it behaved with hard edges. We used both Windows Models for the small details and the Pipe and Wall Models for the hard edges.

From Figures 5.15 and 5.16, we observe that the Master TAA is the best technique for handling all the edges from all the models, its SSIM map barely has any dark area and on Table 5.8 it surpasses any other technique; the Uncharted TAA creates blurring around all the edges, this appears on its SSIM maps as all the edges are visible; and SMAA fail to correct some aliasing artifacts which we can observe on its SSIM map.

Tabla 5.8: Numerical results of the Hard Edges Test.

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

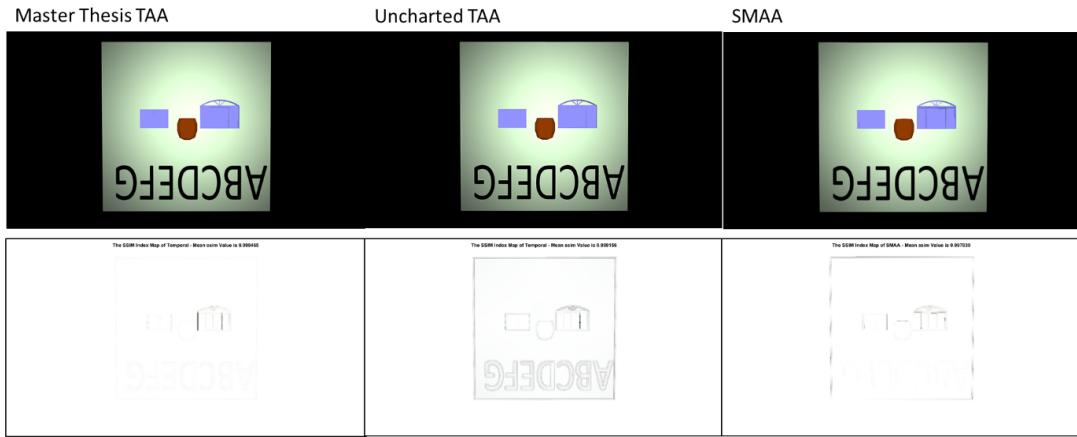


Figura 5.15: Hard Edges comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

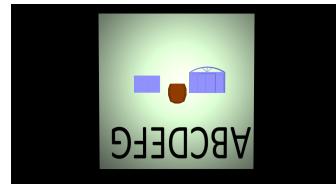


Figura 5.16: Hard Edges ground truth.

### 5.2.8. Sphere Ghosting

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

As we see from the visual results on Figure 5.17, ghosting effects were diminished in our implementation as the stripes visible on the Uncharted TAA are almost invisible on the Master Thesis TAA. Furthermore, in Table 5.9 the average results on the Master TAA image metrics are better than the average from the Uncharted TAA.

It is important to note that some metrics got an infinite result as they were exactly the same as the ground truth. This happens on some images in which the spheres cover the whole frame with a dark blue color. As well, the Test Index marks which test the associated result belongs to; on the averages we use Not Applicable (N/A) because those results come from the average of all the tests.

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

Sphere Ghosting Test Summary				
AA Tests	Uncharted TAA	Test Index	Master TAA	Test Index
Best MSE	0.000	63	0.000	63
Worst MSE	100.871	19	91.766	29
Average MSE	28.634	N/A	19.501	N/A
Best Peak-SNR	Inf	63	Inf	63
Worst Peak-SNR	28.093	19	28.504	29
Average Peak-SNR	Inf	N/A	Inf	N/A
Best SNR	Inf	63	Inf	63
Worst SNR	16.818	11	18.524	29
Average SNR	Inf	N/A	Inf	N/A
Best SSIM	1.000	63	1.000	63
Worst SSIM	0.964	99	0.972	29
Average SSIM	0.965	N/A	0.990	N/A

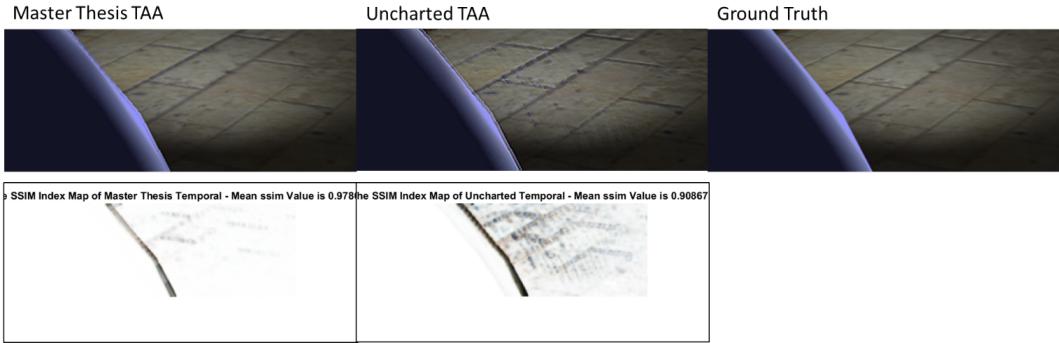


Figura 5.17: Example of Ghosting. Comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 19.

### 5.2.9. Hairball

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

#### Static Shadow

From Figures 5.18 and 5.19, we can observe that the Uncharted TAA has problems on most of the fibers, on its SSIM map this is visible as the big dark edge around Hairball; SMAA fails to correct aliasing on the smaller fibers, we observe this as the gray areas around the Hairball model on the SSIM map; and, finally, we notice that the Master TAA is the best at handling the fibers, they look smooth as in the original model, and its SSIM map has the least amount of dark areas. Also, from Table 5.10 we can confirm that the Master TAA is the best at rendering the static shadow Hairball by a relatively big margin compared to the other techniques.

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

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

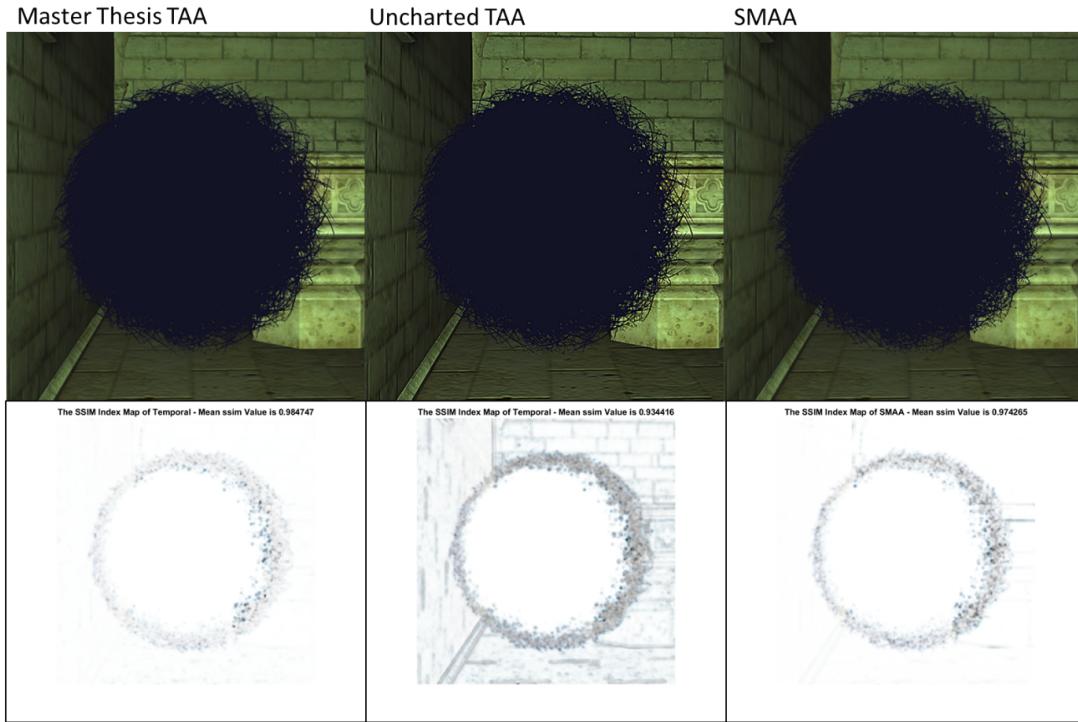


Figura 5.18: Hairball Static Shadow comparison between Master Thesis TAA, Uncharted TAA, and SMAA.



Figura 5.19: Hairball Static Shadow ground truth.

### Static Light

From Figures 5.11 and 5.20, we observe that the Uncharted TAA generated wrong bright colors around all the fibers, on its SSIM this appears as the complete model is dark; SMAA fails to correct a high amount of fibers, they appear aliased on the rendered image and its SSIM map contains many dark zones; and, finally, we can observe that the Master TAA has the smoothest edges of all the rendered images, on its SSIM map and on Table 5.11 we notice that there still errors but they are smaller by a large margin compared to any other technique.

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

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

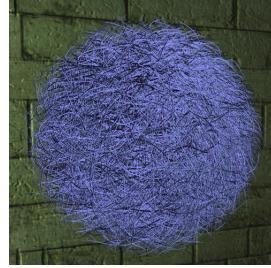


Figura 5.20: Hairball Static Lighted ground truth.

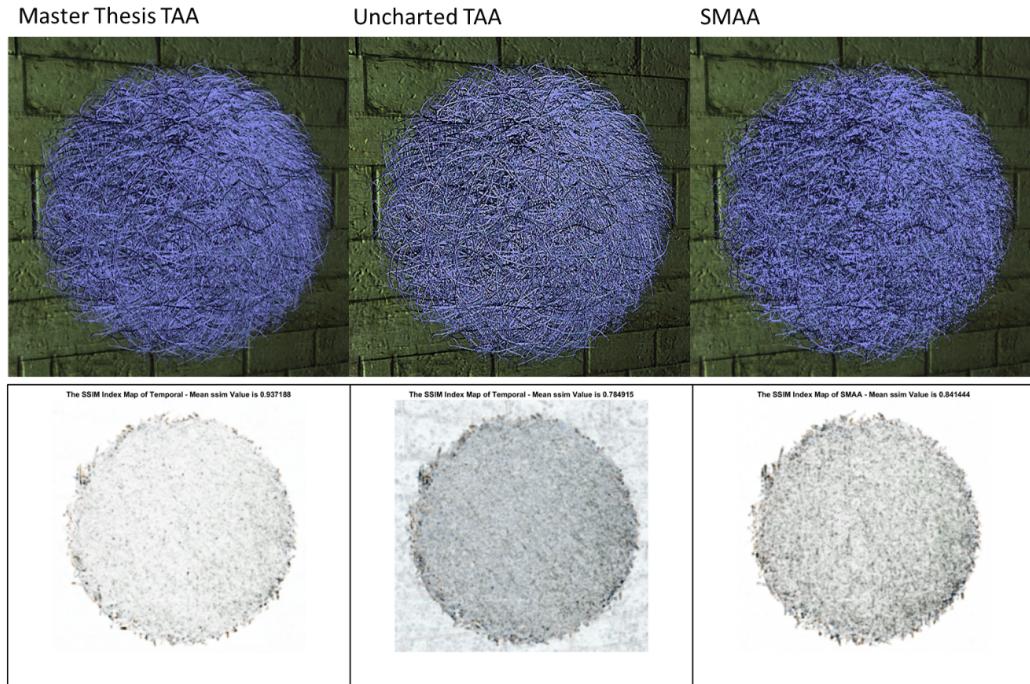


Figura 5.21: Hairball Static Lighted comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

### Ghosting Shadow

From Figure 5.22, we can observe that both implementations generate blurriness around the fibers edges, this is visible on both SSIM maps as the dark ring around the model. On Table 5.12 we observe that the Master Thesis TAA is numerically better than the Uncharted TAA. As well, the Test Index on Table 5.12 marks which test the associated result belongs to; on the averages we use Not Applicable (N A) because those results come from the average of all the tests.

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

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

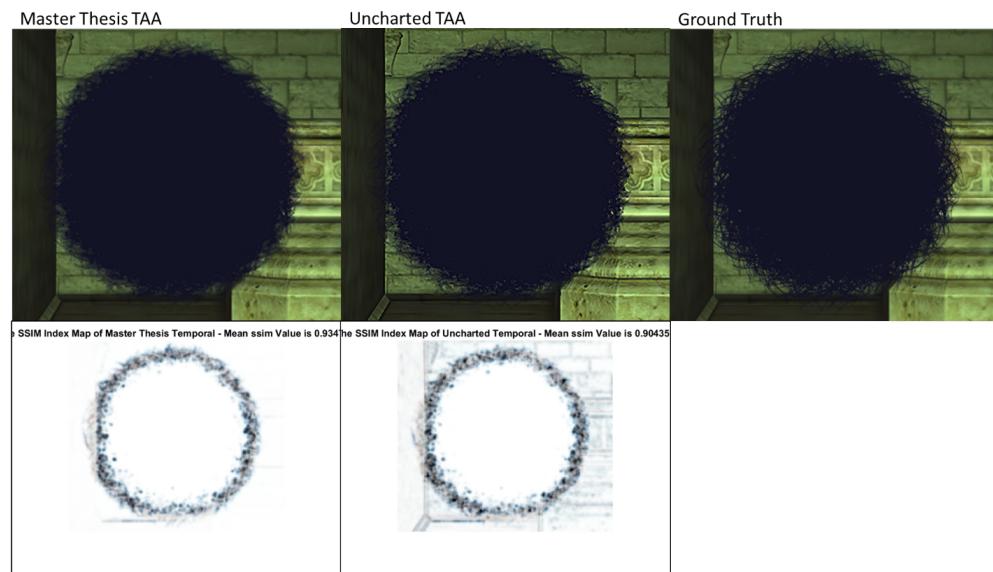


Figura 5.22: Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 90.

## Ghosting Light

From Figure 5.23 and Table 5.13, we observe that both techniques generate a high amount of blurring around the fibers. On both SSIM maps the blurring is visible as Hairball is seen as a big dark area. As well, the Test Index on Table 5.13 marks which test the associated result belongs to; on the averages we use Not Applicable (N/A) because those results come from the average of all the tests.

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

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

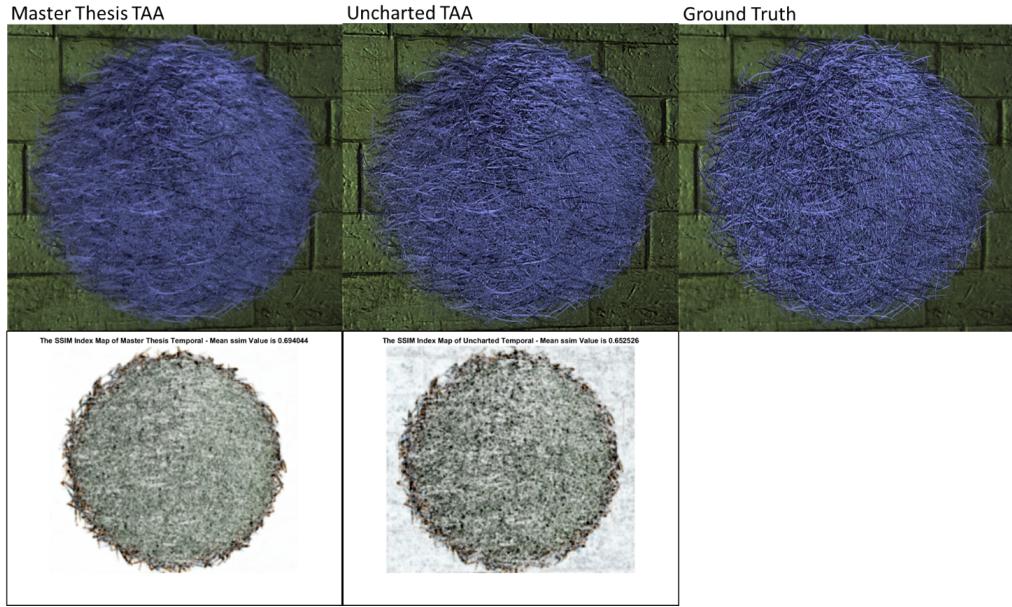


Figura 5.23: Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 83.

### 5.2.10. Timing

It is important to note that all the tested techniques are of Post-Processing nature, that means that they receive the output image from the Deferred Shading Architecture as input, making them not dependent on the complexity of the scene.

The measured time the Master Thesis TAA technique took to run was between 0.5 and 0.6 *ms* on average. The Sobel pass took between 0.2 *ms* and 0.3 *ms*, and the reprojection was around 0.3 *ms*. The measured time the Uncharted TAA technique took was between 0.2 *ms* and 0.3 *ms*; FXAA took between 0.1 *ms* and 0.2 *ms*; and SMAA took 0.2 *ms* and 0.3 *ms*.

## 5.3. Discussion

As we appreciate the results of the Sharpen Filter Test, see Table 5.1, the improvements achieved in this master thesis go beyond modifying the filter from the Uncharted implementation, because changing this filter only avoids generating those wrong bright pixels around edges. We believe that their use of that specific filter is of artistic nature. It tends to pronounce the edges at the cost of creating bright pixels around the edges

that flicker, especially, when the camera moves while the foreground is illuminated but the background is in shadows or vice versa.

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

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

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

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

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

In the Sphere Ghosting Test we see a clear example of the improvements accomplished

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

Finally, we have the four Hairball Tests, the most complex tests performed. The Hairball model has many gaps and small details that react to lighting and shadows. We anticipated a high amount of errors due to them because all Anti-Aliasing techniques have difficulties reconstructing this high density of fine details.

On the Shadow and Light Static Test (See Tables 5.10 and 5.11) we can observe that the Master Thesis Implementation is the best at handling the hair fibers. It is able to reconstruct smoother edges than the Uncharted TAA and reconstructs more details than SMAA technique. Especially on the light version (See Figure 5.21), we can appreciate how smooth the result is; it looks almost like the ground truth. This test far exceeded the expectation of our improvements, the visual and numerical (See Table 5.11) results shows a big increase in quality compared to the other Anti-Aliasing solutions.

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

From our timing results (See 5.2.10), we can see that our improvements fit the time requirements to run on real-time applications as it is below the 1 ms common limit.

# CAPÍTULO VI

## CONCLUSIONS AND RECOMMENDATIONS

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

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

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

## REFERENCIAS

- [1] T. Akenine-Möller. [mobile] graphics hardware. Pages 7-8.
- [2] Y. A. Y. Al-Najjar and D. C. Soong. Comparison of image quality assessment: PSNR, HVS, SSIM, UIQI. *International Journal of Scientific & Engineering Research*, August 2012. Volume 3, Issue 8.
- [3] E. Angel and D. Shreiner. *Computer Graphics A Top-Down Approach with Shader-Based OpenGL 6th Edition*. Addison-Wesley, 2011.
- [4] Specter Arts. Pipe model, March 2010. Accessed: 2018-02-15, <https://www.turbosquid.com/3d-models/pipe-stone-bowl-3d-model/522479>.
- [5] J. Chapman. Per-Object Motion Blur, September 2012. Accessed: 2017-11-30, <http://john-chapman-graphics.blogspot.se/2013/01/per-object-motion-blur.html>.
- [6] L. J. 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>.
- [7] Isabela H. Arched window model, January 2016. Accessed: 2018-02-15, <https://www.cgtrader.com/free-3d-models/architectural/window/window-arch>.
- [8] P. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. *ACM Computer Graphics*, August 1990. Volume 24, Number 4, August 1990.
- [9] J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez. SMAA: Enhanced subpixel morphological antialiasing. *EUROGRAPHICS 2012 / P. Cignoni, T. Ertl Volume 31 (2012), Number 2*, 2012.
- [10] J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Pertuis, H. Yu, M. McGuire, T. Lottes, H. Malan, E. Persson, D. Andreev, and T. Sousa. Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses*, 2011.

- [11] T. Lottes. FXAA. *NVIDIA Docs*, 2009. The Document was last edited in 2011.
- [12] M. Doggett. EDAF80 Computer Graphics, September 2017.
- [13] M. Doggett. EDAN35 High Performance Computer Graphics, November 2017.
- [14] W. S. Malpica and A. C. Bovik. Range image quality assessment by structural similarity. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1149–1152, April 2009.
- [15] M. McGuire. Computer graphics archive, July 2017. Accessed: 2018-02-15, <https://casual-effects.com/data>.
- [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] I. Sobel. An isotropic 3x3 image gradient operator. *Irwin Sobel Correspondence*, 02 2014.
- [18] B. Wronski. Temporal Supersampling and Antialiasing. <https://bartwronski.com/2014/03/15/temporal-supersampling-and-antialiasing/>, March 2014. Accessed: 2017-12-01.
- [19] K. XU. Temporal Antialiasing In Uncharted 4. *SIGGRAPH 2016*, 2016.
- [20] L. Yang, D. Nehab, P. V. Sander, P. Sitthiamorn, J. Lawrence, and H. Hoppe. Amortized supersampling. *ACM Trans. Graph.* 28 (2009), 135:1–135:12, 2009.
- [21] C. Yim. Window with blinds model, October 2015. Accessed: 2018-02-15, <https://www.cgtrader.com/free-3d-models/architectural/window/window-brown>.

# **APÉNDICE A**

## **GITHUB REPOSITORY**

The main link to the repository is <https://github.com/maniatic0/Christian-TRAAs>. From there, the next links can be accessed for the printed version of this report.

- Full Repository: <https://github.com/maniatic0/Christian-TRAAs>.
- Complete Computer Specification:  
<https://github.com/maniatic0/Christian-TRAAs/tree/master/PC%20Specification>.
- All Tests:  
<https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests>.
- Accumulation Buffer Tests:  
<https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests/Accumulation>.
- Most of the Master Thesis Tests:  
<https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests/Master%20Th>.
- Sharpening Tests:  
[https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests/Sharpening%](https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests/Sharpening%2).
- Ghosting Tests:  
<https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests/Ghosting>.
- HairBall Tests:  
<https://github.com/maniatic0/Christian-TRAAs/tree/master/Important%20Tests/HairBall>.
- Code:  
[https://github.com/maniatic0/Christian-TRAAs/tree/master/CG\\_Labs](https://github.com/maniatic0/Christian-TRAAs/tree/master/CG_Labs).
- L<sup>A</sup>T<sub>E</sub>XReport:  
[https://github.com/maniatic0/Christian-TRAAs/tree/master/LaTeX/Master\\_Thesis](https://github.com/maniatic0/Christian-TRAAs/tree/master/LaTeX/Master_Thesis).

# APÉNDICE B

## CAMERA JITTERING EXPLANATION

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

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

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

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

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

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

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

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

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

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

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

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