



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

**MUESTREO MEJORADO PARA TEMPORAL ANTI-ALIASING**

Por  
Christian Alexander Oliveros Labrador

**TRABAJO DE GRADO**

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

Sartenejas, Septiembre 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  
Christian Alexander Oliveros Labrador

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, Septiembre de 2018

Página reservada para el acta de evaluación

## **DEDICATORIA**

Dedicado a mi papá, mi mamá, mi hermano, toda mi familia, mis amigos de Venezuela, mis amigos de todo el mundo y a todos los profesores que me han estado para mí durante el camino.

Dedicado a la Universidad Simón Bolívar, la Universidad de Lund y a ambas comunidades universitarias.

## 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 este proyecto de grado, se explora 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. El 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.

**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 . . . . .	2
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 . . . . .	8
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 . . . . .	13
3.6.3.	Buffer de Historia . . . . .	13
3.6.4.	Clipping Color Box . . . . .	15
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 . . . . .	24
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.	Modificaciones del Filtro de Nitidez . . . . .	34
<b>CAPÍTULO V: RESULTADOS</b>		<b>35</b>
5.1.	Metodología de Evaluación . . . . .	35
5.2.	Resultados y Comparaciones . . . . .	36
5.2.1.	Filtro de Nitidez . . . . .	36
5.2.2.	Tubería . . . . .	38
5.2.3.	Ventana con Persianas . . . . .	42
5.2.4.	Ventana Arqueada . . . . .	43
5.2.5.	Atrium de Sponza . . . . .	44
5.2.6.	Flores del Atrium de Sponza . . . . .	45
5.2.7.	Bordes Duros . . . . .	47
5.2.8.	Prueba de Ghosting de Esfera . . . . .	48
5.2.9.	Hairball . . . . .	50
5.2.10.	Comparación de Tiempos . . . . .	56

5.3. Discusión . . . . .	56
<b>CAPÍTULO VI: CONCLUSIONES Y RECOMENDACIONES</b>	<b>59</b>
<b>REFERENCIAS</b>	<b>60</b>
<b>APÉNDICE A: REPOSITORIO DE GITHUB</b>	<b>62</b>
<b>APÉNDICE B: EXPLICACIÓN DEL CAMERA JITTERING</b>	<b>64</b>

# ÍNDICE DE FIGURAS

3.1. Canal de Renderizado, basado en la 5ta clase de EDAF80. [13] . . . . .	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. [14] . . . . .	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. [7] . . . . .	12
3.6. Valores usados de la <i>Halton Sequence</i> (2, 3) (Secuencia de Halton). . . . .	13
3.7. Patrón de Muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [7] . . . . .	14
3.8. Proceso de <i>Temporal Reprojection Anti-Aliasing</i> . Imagen tomada de la presentación de Fuglsand [7] . . . . .	14
3.9. <i>Color Clamping</i> versus <i>Color Clipping</i> . Imagen tomada del <i>paper</i> de Fuglsand. [7] . . . . .	15
 4.1. Los 128 de la <i>SecuendiaHalton</i> (2, 3) disponibles para usar. . . . .	22
4.2. Patrón de muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [7] . . . . .	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. Comparación de las Imágenes Renderizadas. . . . .	38
5.2. Comparación de Mapas de SSIM. . . . .	38
5.3. Imagen Base de la Prueba de Tubería Regular. . . . .	40
5.4. Comparación de Tubería Regular entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	40
5.5. Comparación de Prueba de Tubería con Inclinación de Cámara entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	41
5.6. Imagen Base de Prueba de Tubería con Inclinación de Cámara. . . . .	41
5.7. Imagen Base de Ventana con Persianas. . . . .	42
5.8. Comparación de Prueba de Ventana con Persianas entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	43

5.9. Comparación de Prueba de Ventana Arqueada entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	44
5.10. Imagen Base de Ventana Arqueada. . . . .	44
5.11. Comparación de Prueba del Atrium de Sponza entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	45
5.12. Imagen Base del Atrium de Sponza. . . . .	45
5.13. Imagen Base de las Flores del Atrium de Sponza. . . . .	46
5.14. Comparación de Prueba de las Flores del Atrium de Sponza entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	47
5.15. Comparación de la Prueba de Bordes Duros entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	48
5.16. Imagen Base de los Bordes Duros. . . . .	48
5.17. Ejemplo de <i>Ghosting</i> . Comparación entre TAA del Proyecto de Grado y TAA de Uncharted en la Prueba Número 19. . . . .	49
5.18. Comparación de la Prueba Estática de Hairball Sin Luz entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	51
5.19. Imagen Base de Hairball Estático Sin Luz. . . . .	51
5.20. Comparación de la Prueba Estática de Hairball Con Luz entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA. . . . .	53
5.21. Imagen Base de Hairball Estático Con Luz. . . . .	53
5.22. Comparación de <i>Ghosting</i> entre TAA del Proyecto de Grado, TAA de Uncharted y la Imagen Base en la Prueba Número 90. . . . .	54
5.23. Comparación de <i>Ghosting</i> entre TAA del Proyecto de Grado, TAA de Uncharted y la Imagen Base en la Prueba Número 83. . . . .	55

## 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. Resultados numéricos de la prueba del Filtro de Nitidez . . . . .	37
5.2. Resultados numéricos de la prueba de Tubería con un ángulo de cámara regular. . . . .	39
5.3. Resultados numéricos de la prueba de Tubería con una inclinación de cámara sesgada. . . . .	41
5.4. Resultados numéricos de la prueba de Ventana con Persianas. . . . .	42
5.5. Resultados numéricos de la prueba de Ventana Arqueada. . . . .	43
5.6. Resultados numéricos de la Prueba del Atrium de Sponza. . . . .	45
5.7. Resultados numéricos de la Prueba de las Flores del Atrium de Sponza. . .	46
5.8. Resultados numéricos de la Prueba de Bordes Duros. . . . .	47
5.9. Resumen de resultados numéricos de las 100 pruebas realizadas para la Prueba de <i>Ghosting</i> de Esfera. . . . .	49
5.10. Resultados numéricos de la Prueba Estática de Hairball Sin Luz. . . . .	51
5.11. Resultados numéricos de la Prueba Estática de Hairball Con Luz. . . . .	52
5.12. Resumen de resultados numéricos de las 100 pruebas realizadas para la Prueba de <i>Ghosting</i> de Hairball Sin Luz. . . . .	54
5.13. Resumen de resultados numéricos de las 100 pruebas realizadas para la Prueba de <i>Ghosting</i> de Hairball Con Luz. . . . .	55

## LISTA DE ACRÓNIMOS

**USB** Universidad Simón Bolívar

**ms** Milisegundo

**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

$\ll$  Mucho menos que

$\sum_{i=a}^b$  Sumatoria desde  $a$  hasta  $b$  indexada por  $i$

# 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 se quiere renderizar los objetos en una escena, se toman los vértices y se envían 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 se presta 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 pérdida mayor a 5 cuadros por segundo durante el transcurso de la ejecución; y trabajar con una cantidad limitada de memoria y ancho de banda, porque necesitamos poder ejecutarlo en una computadora personal promedio (se toma la definición provista por la encuesta de hardware de una de las mayores tiendas de videojuegos en PC, Steam) [6] o un dispositivo móvil. [14, 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 [14]. 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 este proyecto de grado 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 [14]. Otra técnica es *Multi Sample Anti-Aliasing* (MSAA, *Anti-Aliasing* Multi Muestra), que calcula el color para el píxel final solo una vez [14]. Se puede ver la base de lo que se conoce en la actualidad como *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. et al. [17], 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. et al. [21] en el que describen cómo utilizar la reprojeción de cuadros previos en el actual, como método de *Anti-Aliasing* en tiempo real.

A continuación, se encuentran técnicas de Post-Procesamiento como *Fast Approximate Anti-Aliasing* (FXAA, *Anti-Aliasing* Aproximado Rápido) por Timothy Lottes [12] que usa una forma de detección de bordes para corregir el *aliasing* mientras es compatible con la Arquitectura de *Shading* Diferido (*Deferred Shading Architecture*) [14]. También se tiene 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 [11].

También se dispone de la técnica *Enhanced Subpixel Morphological Anti-Aliasing* (SMAA,

*Anti-Aliasing de Mejora Morfológica de Subpíxeles) de Jorge Jiménez et al.* [10] que utiliza una técnica de reconstrucción de bordes más compleja que FXAA, ya que utiliza una máscara de bordes comunes calculados previamente para realizar la detección de bordes , al mismo tiempo que puede trabajar con SSAA, MSAA y una forma básica de TAA.

Finalmente, se tiene las implementaciones TRAA de Ke Xu para Uncharted 4 y Lasse 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 este proyecto de grado [7, 20].

### 1.3. Contribución

este proyecto de grado mejora los últimos avances de Ke Xu y Lasse Fuglsang [7, 20]. 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 este proyecto de grado y las razones de su uso.

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

Utilizamos C ++ y el *Bonobo Framework* para implementar todas las mejoras realizadas en este proyecto de grado. 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 Interfaz de Programación de Aplicaciones (API, *Application Programming Interface*) porque es la base del sistema de renderización del *Bonobo Framework*, el cual modificamos para desarrollar nuestras mejoras, y por su compatibilidad multiplataforma. La Librería Abierta de Gráficos (OpenGL, *Open Graphics Library*) 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 los conceptos teóricos que constituyen la base de este trabajo de grado, desde la teoría de base de Computación Gráfica hasta las Métricas de Imágenes utilizadas.

### 3.1. Canal de Renderizado

El Canal de Renderizado Gráfico son las etapas que se realizan para transformar la información de lo que queremos graficar a lo que finalmente vemos en la pantalla. 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.

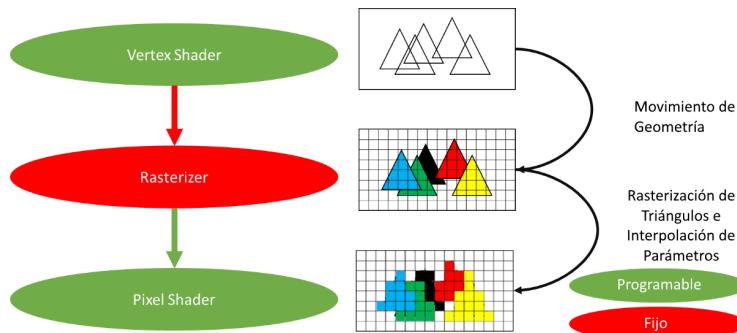


Figura 3.1: Canal de Renderizado, basado en la 5ta clase de EDAF80. [13]

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. [13]

Este proceso de rasterización es importante para este trabajo 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, 14]



Figura 3.3: Ejemplo del problema de Cobertura Parcial, basado en la segunda clase de EDAN35. [14]

### 3.3. El Problema del Aliasing

Cuando mapeamos una representación continua en una finita, se generan 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  $f(x, y)$ , 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 este proyecto de grado, 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 sólo 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 tomar muestras según lo requerido por el Teorema de Muestreo de Nyquist, creando bordes desiguales que aparecen en el proceso de rasterización (*Aliasing Espacial*) y saltos entre objetos en movimiento (*Aliasing Temporal*), de acuerdo con Doggett y Wronski [14, 19]. La Figura 3.4 muestra un ejemplo de *Aliasing Espacial*. Se han propuesto y utilizado muchas soluciones para resolver el problema de *Aliasing*, como por ejemplo la familia de soluciones *Super Sampling Anti-Aliasing* (SSAA), que trabaja con 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 [14], bajo el 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 y consiste en la representación de 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 para realizar el *Shading*, utilizando una operación que podría ser

costosa, a solo píxeles visibles y así evitar desperdiciar recursos realizando el *Shading* a píxeles de geometrías que están cubiertas por píxeles de otros objetos. 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 el área  $4 \times 4$  alrededor de cada pixel para calcular el resultado [14]. 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* [14]. Esto hace que sea complicado de utilizar con los Canales de Rendereación 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 post-procesamiento. Esta funciona al detectar bordes en las imágenes renderizadas, para luego suavizarlos. [12]

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*. [10]

## 3.6. Temporal Anti-Aliasing

Como explicaron Ke Xu y Lasse Fuglsang en sus respectivas presentaciones [20, 7], 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 incorrectos. 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, se pierde información subpíxel alrededor de los bordes que podría usarse para refinar la imagen final. [7, 20]

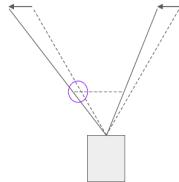


Figura 3.5: Proceso de *Jittering* sobre la proyección de la cámara. Imagen tomada de la presentación de Fuglsand. [7]

El *Jittering* se aplica como una traslación a la matriz de proyección de la cámara usando la *Halton Sequence* (2, 3) (Secuencia de Halton 2D utilizando 3 como base) 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. [7, 20]

La Figura 3.6 muestra la representación de los 16 puntos utilizados para trasladar la proyección en la implementación actual del Proyecto de Grado, tal como lo propusieron Fuglsand [7]. Los puntos se generaron usando MATLAB y luego se codificaron para mejorar su aleatoriedad usando reverse-radix scrambling.

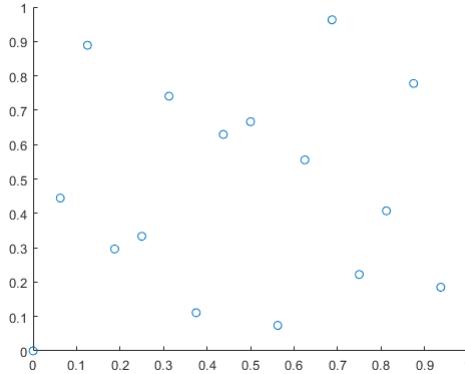


Figura 3.6: Valores usados de la *Halton Sequence* (2, 3) (Secuencia de Halton).

### 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 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. [20]

### 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. [7]

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 [7]. Usamos esta velocidad para reproyectar la posición del píxel del cuadro actual en el Buffer de Historia. [7, 20]

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

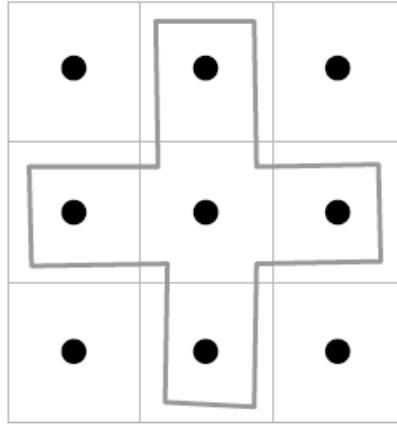


Figura 3.7: Patrón de Muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [7]

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. Esta interpolación lineal estabiliza la imagen, eliminando el *Jittering* y suavizando los bordes [7, 20]. 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. [7]

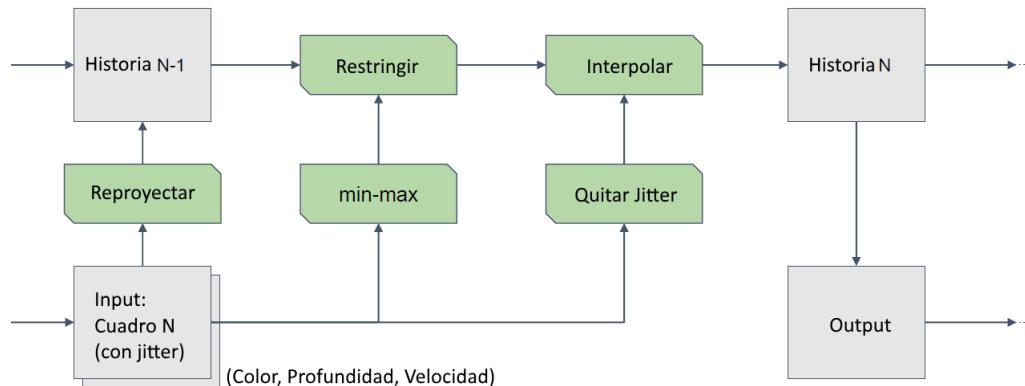


Figura 3.8: Proceso de *Temporal Reprojection Anti-Aliasing*. Imagen tomada de la presentación de Fuglsand [7]

### 3.6.4. Clipping Color Box

La *Clipping Color Box* (Caja de Recorte de Colores) 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 muy alejado del color actual. Tomamos el color de la historia como un vector y luego lo proyectamos contra los bordes de la caja; si se encuentra fuera del borde de la caja, 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) [7].



Figura 3.9: *Color Clamping* versus *Color Clipping*. Imagen tomada del *paper* de Fuglsand. [7]

### 3.6.5. Filtro de Nitidez

El Proceso de Reproyección y el *Clipping* de Colores crean imágenes borrosas, por lo que se requiere un Filtro de Nitidez. Usamos el filtro propuesto por Xu. [20]

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

La Ecuación 3.1 representa la matriz de convolución utilizada como Filtro de Nitidez en el *paper* de Xu. [20]

### 3.6.6. Motion Blur

Debido a la naturaleza del Buffer de Historia, algo de *ghosting* es creado por algunos 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 [7, 20] 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. 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*. este proyecto de grado pretende ayudar a reducir los efectos de estos dos inconvenientes utilizando nuevos enfoques. 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 [20], que resuelve el desenfoque razonablemente bien, pero no logra eliminar algunos artefactos.

#### Ghosting

Algunos artefactos 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 del *Ghosting* permanece cerca de objetos que se mueven lo suficientemente rápido como para crearlo, pero son 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 [20], 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. [7].

### 3.7. Buffer de Acumulación

El Buffer de Acumulación es una técnica de *Anti-Aliasing* que consiste, según Paul Haeberli y Kurt Akeley [9], 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  $3 \times 3$  computable eficientemente, como lo explica Irwin Sobel [18]. Usamos este operador para detectar bordes en las imágenes renderizadas y marcarlos como posibles lugares con aliasing. Esto es porque los bordes son lugar común para que aparezcan los artefactos de aliasing.

Funciona al tomar las cuatro estimaciones posibles del gradiente simple central en un vecindario de  $3 \times 3$  y sumarlas. La función de imagen se toma como una función de densidad / intensidad y las cuatro estimaciones posibles 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.

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

Siendo la Matriz 3.2 el 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 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 que no perdemos precisión si lo realizamos con enteros pequeños de punto fijo. La magnitud recién calculada es diecisésis veces más grande que el gradiente promedio original.

$$G' = 4 * G = \left[ (c - g - a + i) + (f - d) * 2 \quad (c - g + a - i) * 4 + (b - h) * 2 \right] \quad (3.4)$$

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 métodos principales: subjetivos u objetivos. 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 este proyecto de grado, usamos métodos objetivos.

Los métodos 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 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 utilizadas en este proyecto de grado, 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. [15]

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 este proyecto de grado.

### 4.1. Mejoras al Proyecto de EDAN35

Durante el curso de *High Performance Computer Graphics* (EDAN35) implementamos una técnica de *Temporal Reprojection Anti-Aliasing* (TRAA, Anti-Aliasing de Reproyección Temporal) basada en las presentaciones de Ke Xu y Lasse Fuglsang [20, 7], de los juegos *Inside* y *Uncharted 4*, como proyecto final. 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 este proyecto de grado.

La implementación del proyecto EDAN35 tenía errores en el Proceso de *Jittering*, que se corrigieron al expandir correctamente lo que las implementaciones de Inside y Uncharted 4 se referían por *Jittering* de Cámara, consulte el Apéndice B para obtener la explicación completa de la corrección realizada. El manejo de los puntos de *Halton* se rehízo 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*, sólo se usan los primeros 16 puntos según lo sugerido por Xu y Fuglsang. [20, 7]

El *Anti-Aliasing* de Iluminación Especular, como en 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 corrigen este problema por sí mismas, generalmente funcionan en conjunto con otras soluciones ya creadas. Para evitar problemas con la iluminación especular, se decidió apagarla.

Para probar las mejoras realizadas al *Temporal Anti-Aliasing*, se agregaron los modelos de una esfera, pared, tubería, *Hairball* (Bola de Pelo), una ventana con persianas y una ventana arqueada. Todos los modelos, excepto la pared, se agregaron con una textura de color sólido para evitar la introducción de errores de iluminación en los cálculos de las mé-



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

tricas de imagen, ya que estas son usadas para las comparaciones entre la implementación de Uncharted 4 y la desarrollada en este proyecto de grado. 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 este proyecto de grado, se usó la versión propuesta en el *white paper* de *Fast Approximate Anti-Aliasing* (FXAA, *Anti-Aliasing Aproximado Rápido*) de Lottes [12], 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 con el ajuste de mayor calidad según el *white paper*, sin considerar su impacto en el rendimiento, porque queríamos comparar la mejora cruda que ambas técnicas pueden proporcionar.

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

Con la finalidad de probar una técnica de post-procesado más nueva y compleja, se implementó SMAA siguiendo las instrucciones proporcionadas por Jiménez et al [10]. 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 [9]. 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 fueron: 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 como sea posible de la escena.

#### 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, identificadas 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.

Pruebas \ Muestras	Prueba D			
	16	128	Diferencia	Diferencia Relativa (%)
MSE de Temporal	40.6479	38.9473	-1.7006	4.1837 %
RMSD de Temporal	6.3756	6.2408	-0.1348	2.1142 %
MSE de No AA	24.8721	24.5250	-0.3471	1.3956 %
RMSD de No AA	4.9872	4.9523	-0.0349	0.7003 %
Peak-SNR de Temporal	32.0404	32.2260	0.1856	0.5759 %
SNR de Temporal	30.3060	30.4924	0.1864	0.6113 %
Peak-SNR de No AA	34.1737	34.2347	0.0610	0.1783 %
SNR de No AA	32.4392	32.5011	0.0618	0.1903 %
SSIM de Temporal	0.9933	0.9935	0.0001	0.0150 %
SSIM de No AA	0.9968	0.9969	0.0001	0.0065 %

En algunas métricas, los cambios son lo suficientemente grandes para ser perceptibles,

especialmente en MSE y SSIM de Temporal Anti-Aliasing para nuestras comparaciones entre técnicas *Anti-Aliasing*. Se pueden apreciar los efectos del uso de 128 muestras en el capítulo de comparaciones entre las técnicas *Anti-Aliasing*.

## 4.2. Implementación del Marco de Pruebas

Para medir las mejoras logradas, desarrollamos un Marco de Pruebas que nos permite guardar la información importante cuando se realizan las pruebas. El marco nos permite seleccionar qué técnica utilizar como renderizador principal: *Temporal Anti-Aliasing* implementado en el proyecto de grado, *Temporal Anti-Aliasing* de *Uncharted 4*, Enhanced Sub-pixel Morphological Anti-Aliasing (SMAA) o *Fast Approximate Anti-Aliasing* (FXAA). También nos permite hacer *zoom* en cualquier parte de la pantalla y luego realizar los cálculos de métricas de imágenes usando MATLAB.

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

Para cuantificar si se lograron los objetivos propuestos de este proyecto de grado, 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 con 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 ejecutó únicamente con los modelos de la Esfera y *HairBall*, dado que son los únicos que nos interesa ver en movimiento y medir el *Ghosting* que

presentan. En el primero, la Esfera se mueve a través del pasillo de la escena, simulando un objeto en movimiento en una aplicación. En el segundo, el modelo de *Hairball* gira en una posición estática, simulando muchos bordes en movimiento. La prueba consiste en renderizar la escena para un número seleccionado de cuadros, con la implementación del Proyecto de Grado 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 se regresa 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 es que cada cuadro se procesa utilizando el Buffer de Acumulación y luego se guarda, se hace de esta forma para evitar la gran pérdida de rendimiento, que afecta las técnicas de *Temporal Anti-Aliasing*.

Después de guardar todas las imágenes renderizadas, comparamos las producidas por el *Temporal Anti-Aliasing* del Proyecto de Grado y el *Temporal Anti-Aliasing* de Uncharted 4 contra la imagen base, generada con el Buffer de Acumulación, para calcular las métricas de imagen de ambos TAA. Estas métricas muestran cuánto error se generó por efecto de *ghosting* en ambas implementaciones, lo que nos permite comparar el comportamiento de 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. Se calcularon para comparar cómo se comportó cada técnica en cada prueba, para identificar si el *Temporal Anti-Aliasing* del Proyecto de Grado genera imágenes de mayor calidad que las otras técnicas probadas.

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

Para las Pruebas Estáticas, realizamos mediciones de MSE, RMSD, PSNR, SNR y SSIM en las imágenes renderizadas con TAA, FXAA, SMAA y No AA, utilizando las imágenes renderizadas del Buffer de Acumulación como referencia de comparación. Asimismo,

generamos un mapa local de SSIM para cada imagen renderizada que es guardada, junto con renderizado con el Buffer de Acumulación (Imagen Base), como archivos PNG y FIG de MATLAB. Todos los resultados de las mediciones se almacenan en la carpeta con los resultados de la prueba como texto plano.

Para las 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 la del Temporal Anti-Aliasing del Proyecto de Grado, utilizando el cuadro renderizado del Buffer de Acumulación correspondiente. Se generó un mapa local de SSIM para cada imagen renderizada, exceptuando las del Buffer de Acumulación, y se guardaron como archivos PNG y FIG de MATLAB. Los resultados de todas las imágenes se almacenaron en un archivo de texto plano. En los mapas locales de SSIM, las diferencias son representadas como colores, donde el blanco simboliza la similitud.

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

Para este proyecto de grado, la técnica del *Color Clipping Box* se modificó para que se viera afectada por los valores calculados a partir de las nuevas técnicas aplicadas. Estos cambios siguen la lógica que queremos aplicar toda la fuerza de la técnica de *Temporal Anti-Aliasing* únicamente cuando sea necesario y minimizar la aplicación de la técnica en otras partes, para minimizar los efectos de la *ghosting* y 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), se mezclan en una cantidad variable; a diferencia de antes donde solo se tomaba la el promedio entre ambos. El valor que utilizamos para mezclar se calcula en función de cuanto *aliasing* consideramos que tiene el píxel siendo inspeccionado. La idea es 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 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 tiene el píxel. 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 *Ghosting* ya que rechazamos colores más rápido en píxeles que sabemos que no se consideran con *aliasing*. Esto se implementa mediante la interpolación lineal del color de píxel actual y los colores mínimo

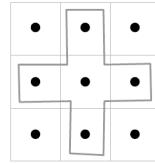


Figura 4.2: Patrón de muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [7]

y máximo calculados para construir la *Color Clipping Box*.

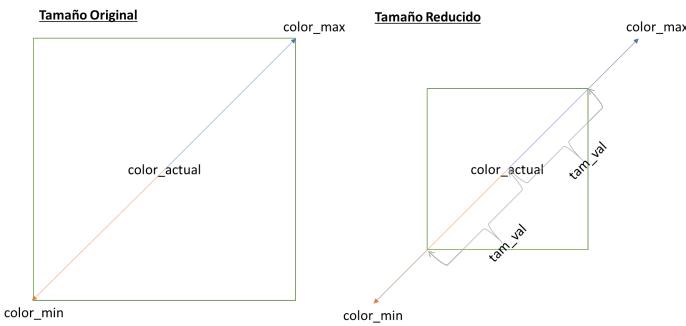


Figura 4.3: Reducción de tamaño de la Color Clipping Box

En las siguientes secciones, explicaremos cómo se calculan los valores que contribuyen a decidir si un píxel se considera con *aliasing* o no. 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. Más adelante mostraremos cómo usamos esos valores 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 identificación (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, mostraremos algunos ejemplos de cómo funciona esta técnica. Para cada matriz, los números representan los ID de los triángulos, siendo la posición central el píxel que actualmente se está calculando y el resto 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 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 para utilizar 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 utilizaremos 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 estar entre 0 y 1 pero, si la profundidad del píxel es un valor atípico en la vecindad, este valor será 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, el cálculo lo hace converger a 0 si el valor es cercano a 0 o a 1 si es cercano a 1.

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

A continuación, mostraremos ejemplos de cómo esta técnica decide si un píxel se con-

sidera 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 otro lado, 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 de aplicar TAA en los píxeles situados en los bordes, para corregir el *aliasing*, en caso de ser necesario. Es importante señalar que esta es la única técnica de las mejoras del Proyecto de Grado 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 ejecuta 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 similar a la que usamos como base de este proyecto de grado, las diferencias provienen del uso de *Clamping* (Restricción) en lugar de una *Color Clipping Box*, porque los valores de textura son unidimensionales. Otra diferencia es 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 en que un píxel 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 como:

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

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.

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

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

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 utilizadas.



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.

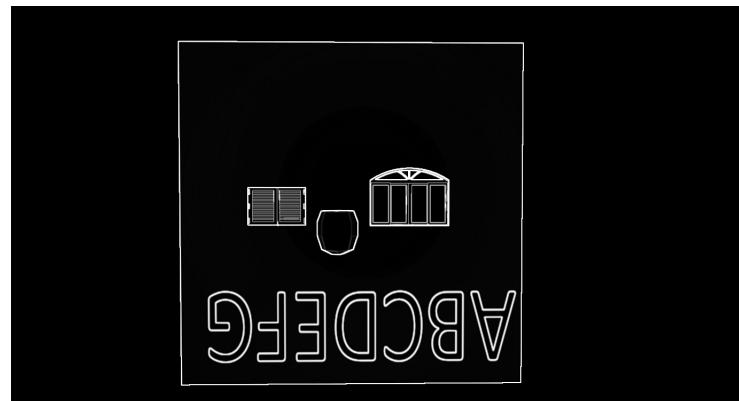


Figura 4.6: Imagen hecha de los valores de *aliasing* de cada píxel.

#### 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, normalizamos el Filtro de Nitidez para evitar crear ese pico de color y trabajar mejor con píxeles brillantes en 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

## RESULTADOS

En este capítulo, explicamos cómo evaluamos las mejoras realizadas al *Temporal Anti-Aliasing*. Mostramos los resultados numéricos y visuales obtenidos. Finalmente, explicamos los resultados y su significado en comparación con la implementación anterior de TAA y otras soluciones de *Anti-Aliasing*.

### 5.1. Metodología de Evaluación

Para evaluar las mejoras logradas en la técnica de *Temporal Anti-Aliasing*, seleccionamos modelos y ángulos de cámara que colocan a la técnica bajo presión. Luego, utilizando el marco de prueba que desarrollamos, procedemos a renderizar y guardar imágenes de cada uno de esos modelos para comparar cómo se comporta la técnica en comparación con la implementación TAA original y otras técnicas *Anti-Aliasing*, para determinar si las técnicas propuestas en este proyecto de grado proporcionan imágenes con mejor calidad sin incurrir en un uso excesivo de memoria o consumo de tiempo. Además, se realizó una prueba especial para medir si el cambio del Filtro de Nitidez era el único mecanismo que proporcionaba una mejora.

Los modelos utilizados en las pruebas fueron:

- Tubería: una tubería marrón con bordes duros.
- Ventana con Persianas: una ventana azul con persianas cerradas.
- Ventana Arqueada: una ventana azul con un arco en la parte superior.
- Pared: una pared blanca con texto negro.
- Atrium de Sponza: ejemplifica una escena general.
- Flores del Atrium de Sponza: ejemplifica un modelo con recortes.

- *Hairball*: contiene muchos detalles finos por sus numerosas fibras.

Es importante tener en cuenta que en Computación Gráfica hay modelos comunes que se usan para probar técnicas, pero no existe un estándar per se. En este proyecto de grado, utilizamos dos modelos comunes: el Atrium de Sponza, que tiene muchas variaciones y se usa comúnmente para probar muchas técnicas de Computación Gráfica; y *Hairball*, que a veces se usa para probar las técnicas *Ray* y *Path Tracing*. Para el resto de los modelos, presentamos una explicación del por qué se seleccionó cada uno en la sección correspondiente a cada una de las pruebas.

Los modelos del Atrium de Sponza y *Hairball* se descargaron del *Computer Graphics Archive* [16] de Morgan McGuire. El modelo de la Tubería es de Spencer Arts [4]. El modelo de Ventana Arqueada es de Isabela H. [8] y la Ventana con Persianas es de Channa Yim [22].

Las pruebas se realizaron en la computadora proporcionada por la Universidad de Lund, que tiene la siguiente especificación:

- CPU: Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz, 3601 Mhz, 4 Cores, 8 Procesadores Lógicos
- RAM: 64.0 GB
- GPU: NVIDIA GeForce GTX 1080 con 8 GB de VRAM
- Resolución de Renderizado cuando no se utiliza el zoom: 1600 x 900

## 5.2. Resultados y Comparaciones

Para tener una base para analizar los datos, necesitamos observar que el mejor valor posible para MSE y RMSD es cero, lo que significa que no hay ningún error. Además, tener un alto valor de PSNR y SNR es mejor porque el ruido, que es el denominador en la ecuación de esta métrica, es cercano a cero. Finalmente, tener un SSIM de 1 significa que la imagen es estructuralmente la misma que la imagen base, mientras que tener un valor de 0 significa que es estructuralmente diferente. Para los mapas SSIM, cada píxel representa su valor SSIM. Tener un color blanco significa que es estructuralmente el mismo, mientras que tener un color más oscuro significa que hay diferencias estructurales.

### 5.2.1. Filtro de Nitidez

Para esta prueba, usamos el modelo del Atrium de Sponza y un modelo de una Esfera estática para evaluar los efectos del Filtro de Nitidez Normalizado en la calidad general

de la imagen renderizada, especialmente con respecto al desenfoque causado normalmente por TAA. La escena fue seleccionada para esta prueba porque proporciona un ejemplo de una escena general, que no debería generar ningún efecto de desenfoque. Todo se procesó utilizando ambas implementaciones de TAA con y sin el Filtro de Nitidez Normalizado para observar si este cambio era la única mejora que aumentaba la calidad de la imagen procesada. Como se puede observar en la tabla 5.1, incluso cuando el TAA Mejorado del Proyecto de Grado y el TAA de Uncharted utilizan el Filtro de Nitidez Normalizado, las otras mejoras que hemos realizado con el TAA renderiza una imagen de mayor calidad. Si hacemos zoom en la figura 5.1 de la Figura 5.2 podemos ver que el Filtro de Nitidez Normalizado contribuye a reducir los artefactos en la imagen renderizada, ya que casi no hay áreas oscuras en los mapas de SSIM de los TAA que lo está usando.

Tabla 5.1: Resultados numéricos de la prueba del Filtro de Nitidez

Pruebas	Prueba del Filtro de Nitidez					TAA Mejorado Normalizado
	AA Uncharted TAA No Normalizado	AA Uncharted TAA Normalizado	TAA Mejorado No Normalizado	TAA Mejorado Normalizado	Mejor	
MSE	149.271	8.835	148.036	8.224		
RMSD	12.218	2.972	12.167	2.868		
Peak-SNR	26.391	38.669	26.427	38.980		
SNR	16.245	28.522	16.281	28.833		
SSIM	0.932	0.992	0.933	0.992		

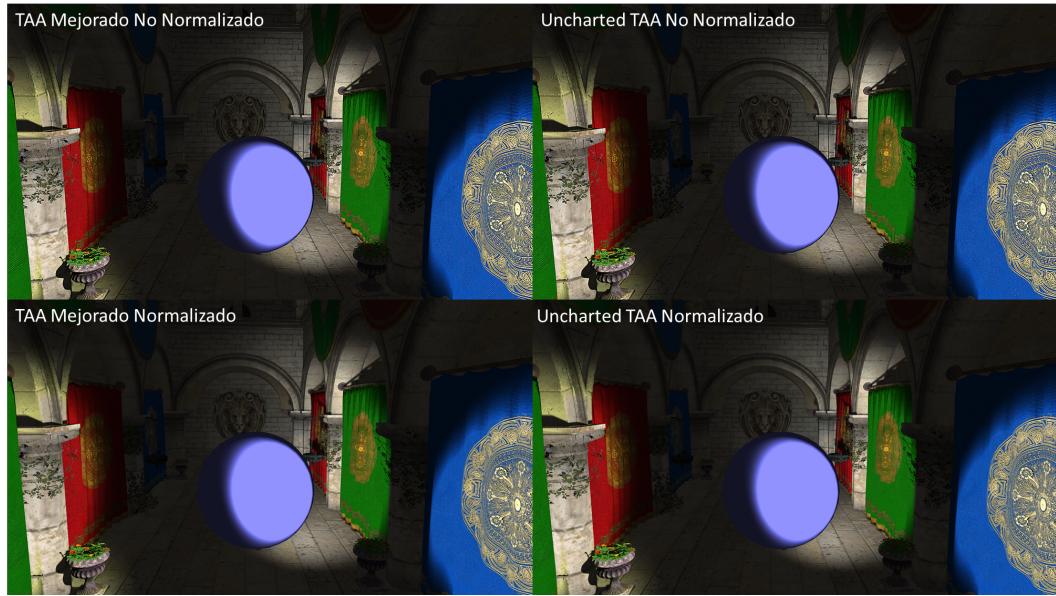


Figura 5.1: Comparación de las Imágenes Renderizadas.

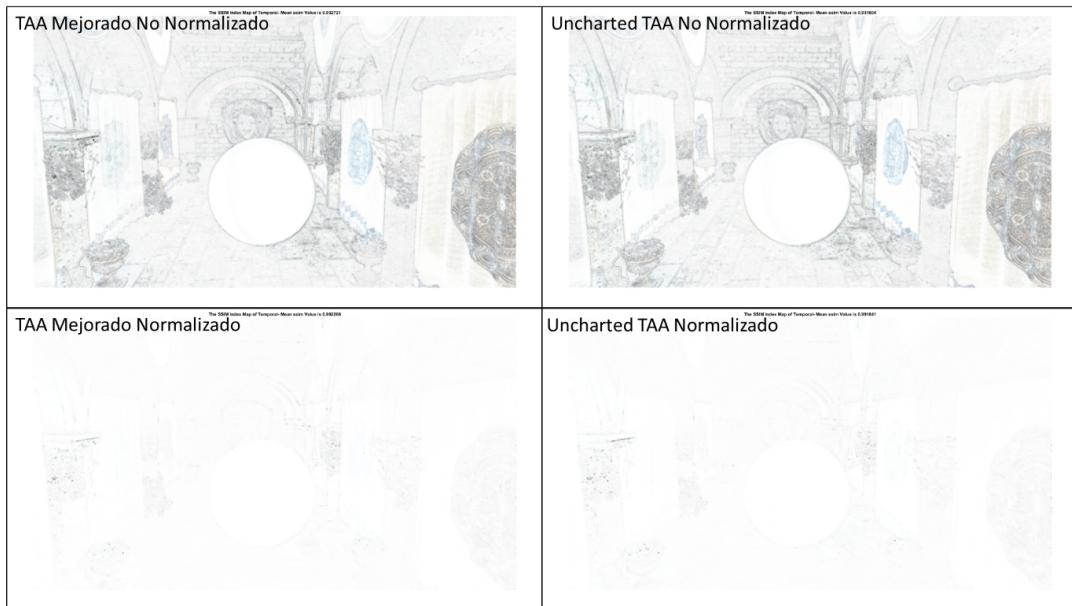


Figura 5.2: Comparación de Mapas de SSIM.

### 5.2.2. Tubería

Utilizamos el modelo de Tubería para probar cómo se comportan las mejoras al renderizar un modelo con bordes duros (*hard edges*). Queríamos probar si nuestras mejoras reducen la cantidad de desenfoque alrededor de esos bordes duros, mientras se disminuyen

los problemas causados por el *aliasing*. Renderizamos el tubo dos veces, la primera vez con un ángulo de cámara normal, para *aliasing* normal alrededor de los bordes, y un ángulo de cámara inclinada, para aumentar los efectos de *aliasing*. Como puede advertirse en los resultados, TAA con nuestras mejoras se encuentra en el mismo nivel de calidad que SMAA.

## Regular

Cuando hacemos zoom y comparamos la Figura 5.3 con las imágenes renderizadas en la Figura 5.4, especialmente alrededor de los bordes, observamos que hay una reducción del desenfoque en el TAA del Proyecto de Grado en comparación con la TAA de Uncharted. Además, observamos que el TAA Uncharted genera colores brillantes alrededor de los bordes que no deberían estar; esto es más fácil de observar que los bordes oscuros en el Mapa de SSIM del TAA de Uncharted en la Figura 5.4. La Tabla 5.2 nos confirma que el TAA del Proyecto de Grado alcanza casi la misma calidad que SMAA.

Tabla 5.2: Resultados numéricos de la prueba de Tubería con un ángulo de cámara regular.

		Prueba de Tubería Regular						
Pruebas	AA	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	TAA Mejorado Contra El Mejor
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

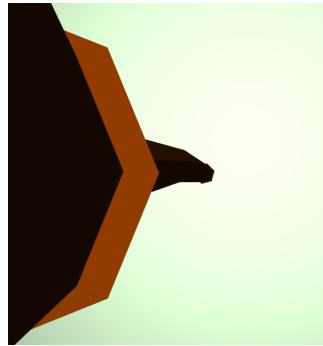


Figura 5.3: Imagen Base de la Prueba de Tubería Regular.

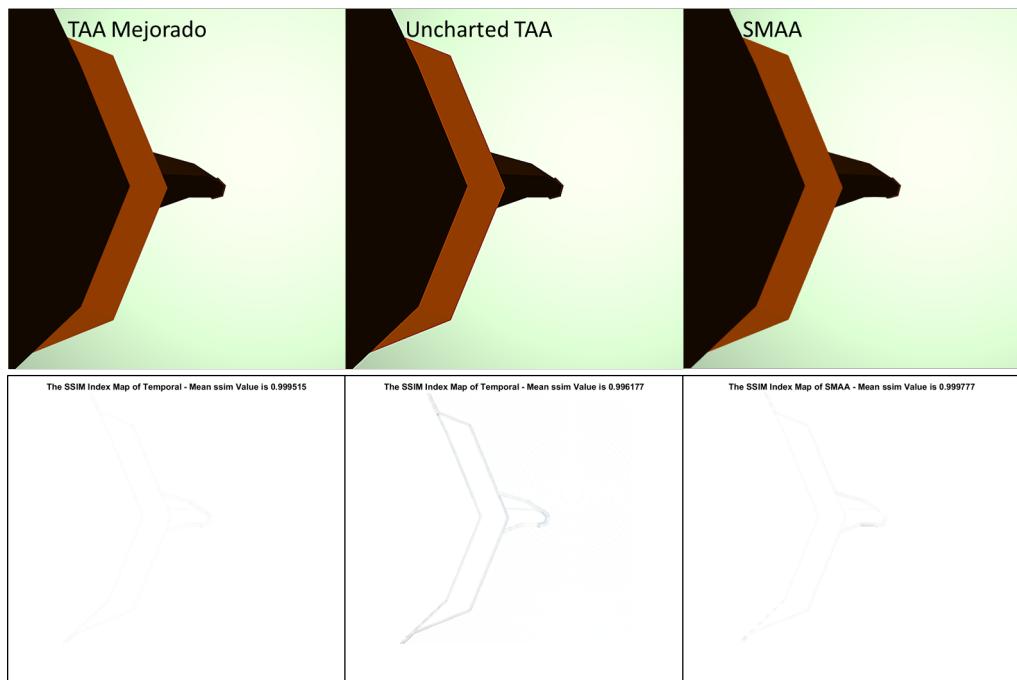


Figura 5.4: Comparación de Tubería Regular entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.

### Con Ángulo de Cámara Inclinada

Al acercar y comparar las Figuras 5.5 y 5.6, podemos observar que el TAA Mejorado es el más semejante a la imagen base. En el mapa de SSIM del TAA de Uncharted, notamos que se está generando desenfoque alrededor de los bordes.

Finalmente, en la Figura 5.5 y la Tabla 5.6 observamos que SMAA no está detectando correctamente el borde superior de la tubería. Cuando hacemos zoom, notamos una pequeña escalera que se forma alrededor del borde. Esto se debe al hecho de que la cámara

se configuró con una inclinación sesgada que llevó al límite las técnicas de detección de bordes utilizadas en SMAA.

Tabla 5.3: Resultados numéricos de la prueba de Tubería con una inclinación de cámara sesgada.

Prueba de Tubería con Inclinación de Cámara							
Pruebas \ AA	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	TAA Mejorado Contra El Mejor
MSE	16.112	6.470	2.810	14.349	2.664	TAA Mejorado	0.000
RMSD	4.014	2.544	1.676	3.788	1.632	TAA Mejorado	0.000
Peak-SNR	36.059	40.022	43.644	36.563	43.876	TAA Mejorado	0.000
SNR	32.474	36.437	40.059	32.978	40.291	TAA Mejorado	0.000
SSIM	0.998	0.999	1.000	0.996	0.999	SMAA	0.000

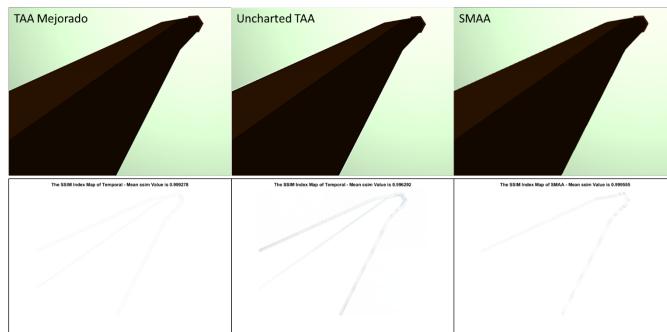


Figura 5.5: Comparación de Prueba de Tubería con Inclinación de Cámara entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.



Figura 5.6: Imagen Base de Prueba de Tubería con Inclinación de Cámara.

### 5.2.3. Ventana con Persianas

Usamos el modelo Ventana con Persianas por sus pequeños detalles en las persianas. Probamos cómo se comportan nuestras mejoras con este tipo de detalles y hallamos que visualmente no reacciona de manera adecuada, incluso si los resultados numéricos muestran lo contrario.

En las Figuras 5.8 y 5.7 podemos observar lo complicado que es para las técnicas manejar las pequeñas brechas entre las persianas. De la Figura 5.8 y la Tabla 5.4, notamos que el TAA Mejorado y SMAA pueden reconstruir más detalles que el TAA de Uncharted pero, estética y visualmente, creemos que es mejor el TAA de Uncharted porque esos pequeños espacios entre las persianas parpadean menos, en comparación a las otras técnicas, cuando se mueve la cámara.

Tabla 5.4: Resultados numéricos de la prueba de Ventana con Persianas.

Prueba de Ventana con Persianas								
Pruebas \ AA	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor		TAA Mejorado Contra El Mejor
MSE	96.044	70.486	35.134	170.229	32.115	TAA Mejorado	0.000	
RMSD	9.800	8.396	5.927	13.047	5.667	TAA Mejorado	0.000	
Peak-SNR	28.306	29.650	32.674	25.820	33.064	TAA Mejorado	0.000	
SNR	25.467	26.810	29.834	22.981	30.224	TAA Mejorado	0.000	
SSIM	0.986	0.990	0.995	0.976	0.995	TAA Mejorado	0.000	

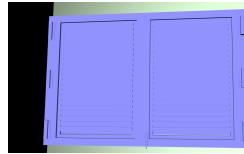


Figura 5.7: Imagen Base de Ventana con Persianas.

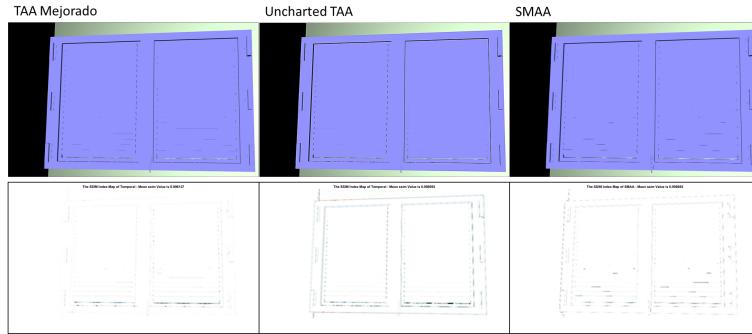


Figura 5.8: Comparación de Prueba de Ventana con Persianas entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.

#### 5.2.4. Ventana Arqueada

Usamos el modelo de Ventana Arqueada para probar cómo nuestra implementación mejorada se comporta con los pequeños detalles de la puerta de la ventana y el *aliasing* del arco. Podemos observar a partir de las Figuras 5.9 y 5.10 que, para todas las técnicas, las pequeñas brechas alrededor de la ventana son difíciles de renderizar. En algunas partes de las brechas pequeñas, notamos que las técnicas no pueden renderizar la brecha por completo. Aunque en la Tabla 5.5 SMAA parece ser el mejor, creemos que el TAA de Uncharted tiene la mejor calidad visual porque esos espacios incompletos generan menos parpadeos de píxeles cuando hay movimiento.

Tabla 5.5: Resultados numéricos de la prueba de Ventana Arqueada.

		Prueba de Ventana Arqueada						TAA
Pruebas \ AA		No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	Mejorado Contra El Mejor
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

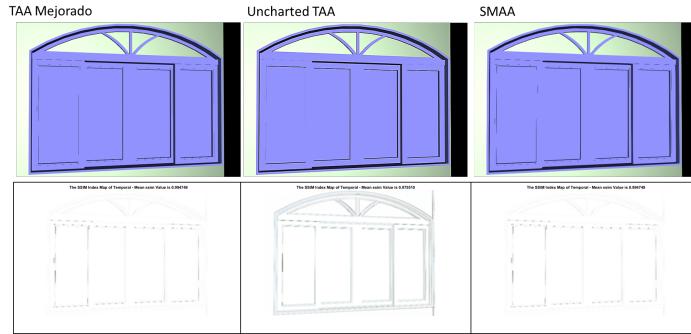


Figura 5.9: Comparación de Prueba de Ventana Arqueada entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.

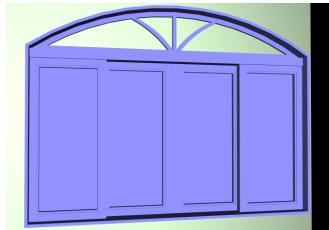


Figura 5.10: Imagen Base de Ventana Arqueada.

### 5.2.5. Atrium de Sponza

Con esta prueba queremos analizar cómo se comporta la implementación de TAA del Proyecto de Grado con una escena general con luces y sombras. Utilizamos el modelo del Atrium de Sponza con el modelo de Esfera estática en el centro.

Si comparamos las Figuras 5.11 y 5.12, podemos observar que el TAA de Uncharted tiene problemas de desenfoque alrededor de todos los bordes, esto es visible en su Mapa de SSIM. SMAA tuvo problemas con todas las flores, podemos observar la forma de las flores en su Mapa de SSIM. El TAA Mejorado solo tiene problemas menores con las flores, como se ve en las áreas grises en su Mapa de SSIM. La Tabla 5.6 confirma lo que estamos observando en los resultados visuales, ya que el TAA Mejorado del Proyecto de Grado obtuvo los mejores resultados en la mayor parte de la prueba; mientras que el TAA de Uncharted obtuvo las peores resultados debido a los problemas de bordes; y SMAA empeoró debido al problema de las flores.

Tabla 5.6: Resultados numéricos de la Prueba del Atrium de Sponza.

Pruebas	Prueba del Atrium de Sponza							TAA Mejorado Contra El Mejor
	AA	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	
MSE	13.458	8.290	8.610	42.728	3.972	TAA Mejorado	0.000	
RMSD	3.669	2.879	2.934	6.537	1.993	TAA Mejorado	0.000	
Peak-SNR	36.841	38.945	38.781	31.824	42.141	TAA Mejorado	0.000	
SNR	20.056	22.160	21.996	15.038	25.356	TAA Mejorado	0.000	
SSIM	0.988	0.991	0.991	0.938	0.991	TAA Mejorado	0.000	



Figura 5.11: Comparación de Prueba del Atrium de Sponza entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.



Figura 5.12: Imagen Base del Atrium de Sponza.

### 5.2.6. Flores del Atrium de Sponza

En esta prueba vimos cómo nuestras mejoras manejan los detalles de un modelo con partes transparentes, como son las flores del modelo del Atrium de Sponza. El *aliasing* que

presentan este tipo de modelos se considera difícil de detectar y corregir correctamente.

En las Figuras 5.14 y 5.13, notamos que el TAA de Uncharted tiene problemas para manejar este tipo de modelo, especialmente si observamos su Mapa de SSIM. Notamos que SMAA no pudo corregir todos los artefactos de aliasing de los bordes de las flores, vemos que la silueta de las flores aparece en su Mapa de SSIM. Finalmente, observamos que el TAA del Proyecto de Grado corrigió la mayoría de los artefactos de aliasing, aunque algunos todavía aparecen como áreas grises en su Mapa de SSIM. Además, la Tabla 5.7 confirma lo que percibimos visualmente, ya que el TAA del Proyecto de Grado obtuvo los mejores puntajes y SMAA obtuvo una puntuación peor que el promedio.

Tabla 5.7: Resultados numéricos de la Prueba de las Flores del Atrium de Sponza.

Pruebas \ AA	Prueba de las Flores del Atrium de Sponza							TAA Mejorado Contra El Mejor
	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor		
MSE	122.795	66.062	72.279	490.281	36.162	TAA Mejorado	0.000	
RMSD	11.081	8.128	8.502	22.142	6.013	TAA Mejorado	0.000	
Peak-SNR	27.239	29.931	29.541	21.226	32.548	TAA Mejorado	0.000	
SNR	19.590	22.282	21.891	13.577	24.899	TAA Mejorado	0.000	
SSIM	0.959	0.975	0.972	0.863	0.985	TAA Mejorado	0.000	



Figura 5.13: Imagen Base de las Flores del Atrium de Sponza.

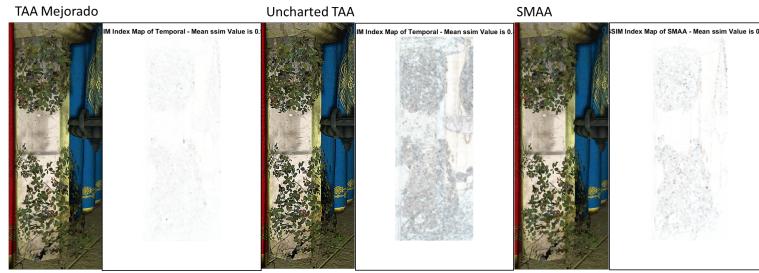


Figura 5.14: Comparación de Prueba de las Flores del Atrium de Sponza entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.

### 5.2.7. Bordes Duros

En esta prueba exploramos cómo la implementación del Proyecto de Grado maneja muchos detalles pequeños a gran distancia y cómo se comporta con bordes duros. Utilizamos los modelos de las Ventanas para los detalles pequeños y los modelos de Tubería y Pared para los bordes duros.

En las Figuras 5.15 y 5.16, observamos que el TAA del Proyecto de Grado es la mejor técnica para manejar los bordes de todos los modelos, su Mapa de SSIM apenas tiene un área oscura y en la Tabla 5.8 sobrepasa cualquier otra técnica. El TAA de Uncharted crea artefactos de desenfoque alrededor de todos los bordes, esto aparece en su Mapa de SSIM ya que todos los bordes son visibles. Finalmente, SMAA no puede corregir algunos artefactos de *aliasing* que podemos observar en su Mapa de SSIM.

Tabla 5.8: Resultados numéricos de la Prueba de Bordes Duros.

		Prueba de Bordes Duros						
Pruebas \ AA		No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	TAA Mejorado Contra El Mejor
MSE		12.463	9.342	8.019	31.385	5.012	TAA Mejorado	0.000
RMSD		3.530	3.057	2.832	5.602	2.239	TAA Mejorado	0.000
Peak-SNR		37.174	38.426	39.090	33.164	41.131	TAA Mejorado	0.000
SNR		30.327	31.579	32.242	26.316	34.283	TAA Mejorado	0.000
SSIM		0.997	0.997	0.998	0.989	0.998	TAA Mejorado	0.000

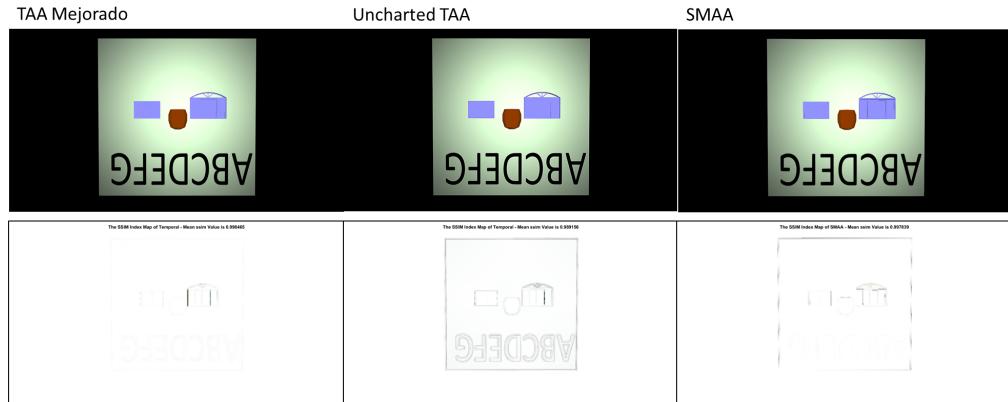


Figura 5.15: Comparación de la Prueba de Bordes Duros entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.

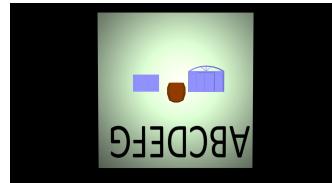


Figura 5.16: Imagen Base de los Bordes Duros.

### 5.2.8. Prueba de Ghosting de Esfera

Con esta prueba queremos medir cómo las mejoras que realizamos disminuyen los efectos de *Ghosting* en las imágenes renderizadas por la implementación de Uncharted. Por esta razón, seleccionamos el Modelo Esfera para que se moviera en la sala del Atrium de Sponza y lo renderizamos bajo un ángulo de cámara que nos mostraba efectos de *Ghosting*.

Como observamos en los resultados visuales de la Figura 5.17, los efectos de *Ghosting* disminuyeron en nuestra implementación ya que las franjas visibles en el TAA de Uncharted son casi invisibles en el TAA del Proyecto de Grado. Además, en la Tabla 5.9 los resultados promedio en las métricas de imágenes del TAA del Proyecto de Grado son mejores que el promedio del TAA de Uncharted.

Es importante notar que algunas métricas obtuvieron un resultado infinito, ya que eran exactamente iguales a la imagen base; esto sucede en algunas imágenes en las que la esfera cubre todo el cuadro con un color azul oscuro. Es importante señalar que el Índice

de Prueba marca a que prueba está asociado el resultado. En los promedios utilizamos No Aplicable (N)

A) en los resultados para indicar promedio proviene de todas las pruebas, ya que no tiene sentido que se asocie el índice de una prueba específica a ellos.

Tabla 5.9: Resumen de resultados numéricos de las 100 pruebas realizadas para la Prueba de *Ghosting* de Esfera.

Pruebas	Resumen de la Prueba de Ghosting de Esfera				
	AA	Uncharted	Índice de Prueba	TAA Mejorado	Índice de Prueba
	TAA				
Mejor MSE	0.000	63	0.000	63	
Peor MSE	100.871	19	91.766	29	
Promedio de MSE	28.634	N/A	19.501	N/A	
Mejor Peak-SNR	Inf	63	Inf	63	
Peor Peak-SNR	28.093	19	28.504	29	
Promedio de Peak-SNR	Inf	N/A	Inf	N/A	
Mejor SNR	Inf	63	Inf	63	
Peor SNR	16.818	11	18.524	29	
Promedio de SNR	Inf	N/A	Inf	N/A	
Mejor SSIM	1.000	63	1.000	63	
Peor SSIM	0.964	99	0.972	29	
Promedio de SSIM	0.965	N/A	0.990	N/A	

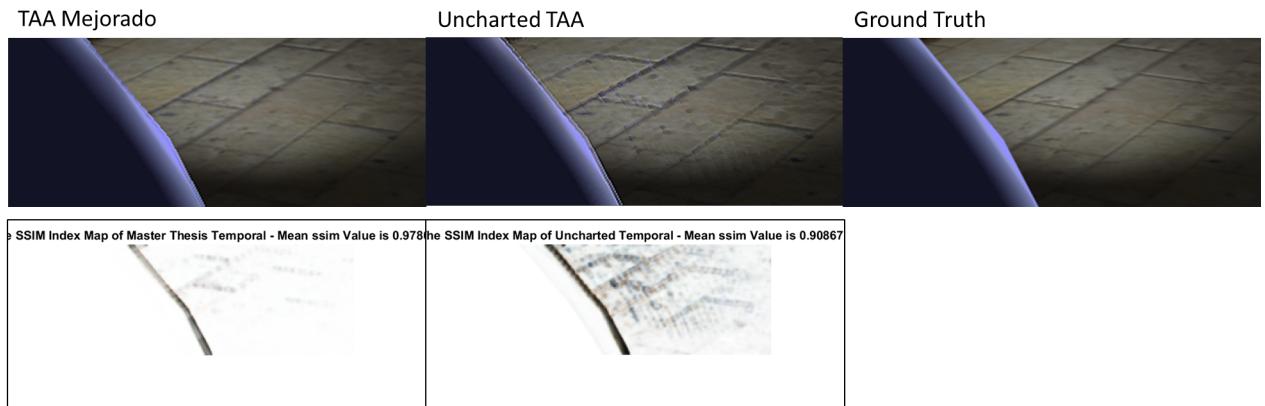


Figura 5.17: Ejemplo de *Ghosting*. Comparación entre TAA del Proyecto de Grado y TAA de Uncharted en la Prueba Número 19.

### 5.2.9. Hairball

Estas pruebas fueron las más complicadas para todas las técnicas que probamos. El modelo *Hairball* contiene muchas fibras con muchos detalles, que proporcionan una prueba compleja para cualquier solución de *Anti-Aliasing*. Realizamos pruebas sin luz para ver la silueta y con la luz para ver cómo su reacción a las fibras afectaba a todas las soluciones Anti-Aliasing. Hicimos dos series de pruebas, en la primera, *Hairball* está estático, para mostrarnos el comportamiento de la corrección del desenfoque y *aliasing*. En la segunda serie, *Hairball* gira sobre su propio eje, para mostrarnos cómo se comporta el efecto de *Ghosting* en las fibras. Los resultados de las pruebas estáticas fueron sorprendentes, ya que no esperábamos que la implementación de TAA del Proyecto de Grado fuera el mejor manejo de las fibras debido a los resultados previos en ambas pruebas con las Ventanas, donde los detalles pequeños de las ventanas no se manejan bien.

#### Prueba Estática Sin Luz

En las Figuras 5.18 y 5.19, podemos observar que el TAA de Uncharted tiene problemas en la mayoría de las fibras, en su Mapa de SSIM esto es visible como el gran borde oscuro alrededor de Hairball. SMAA no puede corregir el *aliasing* en las fibras más pequeñas, observamos esto como las áreas grises alrededor del modelo Hairball en su Mapa de SSIM. Finalmente, notamos que el TAA del Proyecto de Grado es la mejor técnica manejando las fibras, las cuales se ven lisas como en el modelo original y en su Mapa de SSIM tiene la menor cantidad de áreas oscuras. Además, en la Tabla 5.10 podemos confirmar que el TAA del Proyecto de Grado es el mejor para renderizar Hairball estático sin luz por un margen relativamente grande, en comparación con las otras técnicas.

Tabla 5.10: Resultados numéricos de la Prueba Estática de Hairball Sin Luz.

Prueba Estática de Hairball Sin Luz								TAA
Pruebas \ AA	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	Mejorado Contra El Mejor	
MSE	44.367	21.101	25.379	88.976	10.293	TAA Mejorado	0.000	
RMSD	6.661	4.594	5.038	9.433	3.208	TAA Mejorado	0.000	
Peak-SNR	31.660	34.888	34.086	28.638	38.005	TAA Mejorado	0.000	
SNR	18.808	22.036	21.234	15.786	25.154	TAA Mejorado	0.000	
SSIM	0.962	0.978	0.974	0.934	0.985	TAA Mejorado	0.000	

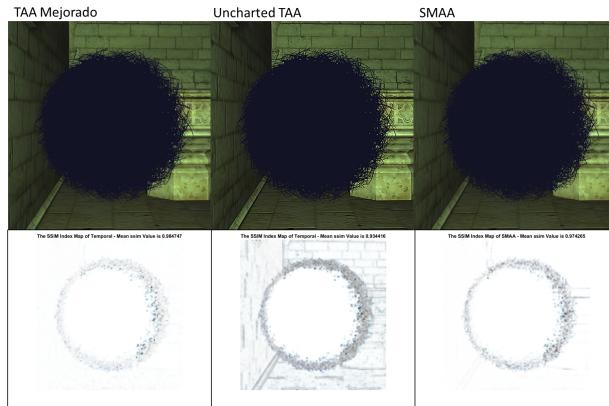


Figura 5.18: Comparación de la Prueba Estática de Hairball Sin Luz entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.



Figura 5.19: Imagen Base de Hairball Estático Sin Luz.

### Prueba Estática Con Luz

En las Figuras 5.11 y 5.21, observamos que el TAA de Uncharted generó colores brillantes incorrectos alrededor de todas las fibras, en su Mapa de SSIM esto aparece como el

modelo completo oscuro. SMAA no puede corregir una gran cantidad de fibras, aparecen con *aliasing* en la imagen renderizada y su Mapa de SSIM contiene muchas zonas oscuras. Finalmente, podemos observar que el TAA del Proyecto de Grado tiene los bordes más suaves de todas las imágenes renderizadas, en su Mapa SSIM y en la Tabla 5.11 notamos que todavía hay errores pero son más pequeños, por un gran margen, comparado a cualquier otra técnica.

Tabla 5.11: Resultados numéricos de la Prueba Estática de Hairball Con Luz.

<b>Prueba Estática de Hairball Con Luz</b>							
<b>Pruebas \ AA</b>	No AA	FXAA	SMAA	Uncharted TAA	TAA Mejorado	Mejor	TAA Mejorado Contra El Mejor
MSE	1294.649	649.940	847.702	1444.095	226.567	TAA Mejorado	0.000
RMSD	35.981	25.494	29.115	38.001	15.052	TAA Mejorado	0.000
Peak-SNR	17.009	20.002	18.848	16.535	24.579	TAA Mejorado	0.000
SNR	8.446	11.439	10.285	7.971	16.015	TAA Mejorado	0.000
SSIM	0.801	0.865	0.841	0.785	0.937	TAA Mejorado	0.000

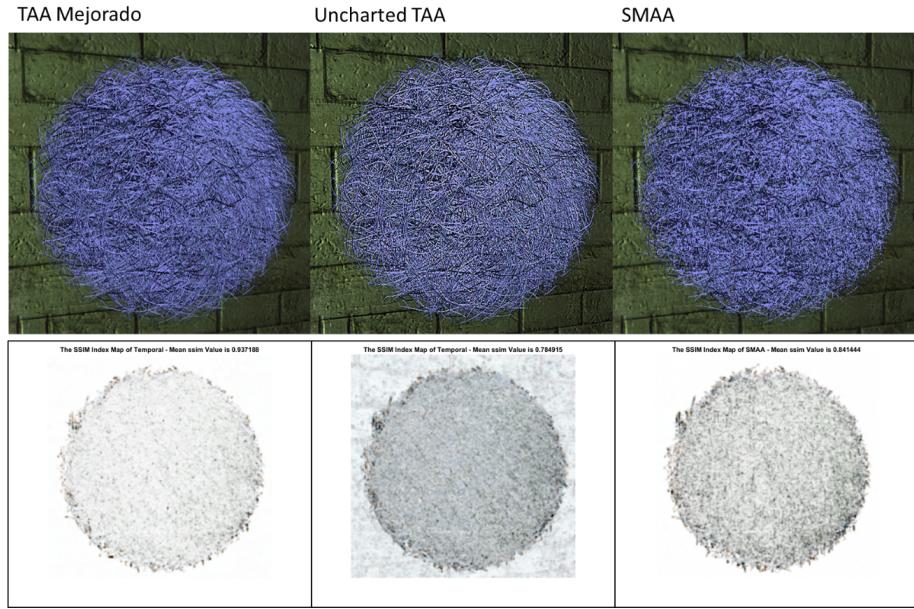


Figura 5.20: Comparación de la Prueba Estática de Hairball Con Luz entre TAA del Proyecto de Grado, TAA de Uncharted y SMAA.

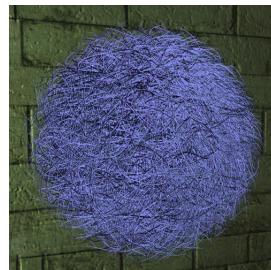


Figura 5.21: Imagen Base de Hairball Estático Con Luz.

### Prueba de Ghosting Sin Luz

En la Figura 5.22, podemos observar que ambas implementaciones de TAA generan desenfoque alrededor de los bordes de las fibras, esto es visible en ambos Mapas de SSIM como el anillo oscuro alrededor del modelo. En la Tabla 5.12 observamos que el TAA del Proyecto de Grado es numéricamente mejor que el TAA de Uncharted. Además, es importante acotar que el Índice de Prueba en la Tabla 5.12 marca a qué prueba pertenece el resultado asociado; en los promedios que utilizamos No Aplicable (N / A) porque esos resultados provienen del promedio de todas las pruebas y no hace sentido que sea asociado al índice de una prueba específica.

Tabla 5.12: Resumen de resultados numéricos de las 100 pruebas realizadas para la Prueba de *Ghosting* de Hairball Sin Luz.

Resumen de Prueba de Ghosting de Hairball Sin Luz				
Pruebas \ AA	Uncharted	Índice de Prueba	TAA Mejorado	Índice de Prueba
Pruebas	TAA	de Prueba	Mejorado	
Mejor MSE	70.261	17	32.692	0
Peor MSE	93.024	90	42.962	90
Promedio de MSE	81.887	N/A	38.534	N/A
Mejor Peak-SNR	29.664	17	32.986	0
Peor Peak-SNR	28.445	90	31.800	90
Promedio de Peak-SNR	29.009	N/A	32.278	N/A
Mejor SNR	16.940	17	20.278	0
Peor SNR	15.846	90	19.201	90
Promedio de SNR	16.333	N/A	19.602	N/A
Mejor SSIM	0.925	17	0.947	17
Peor SSIM	0.911	99	0.934	68
Promedio de SSIM	0.913	N/A	0.940	N/A

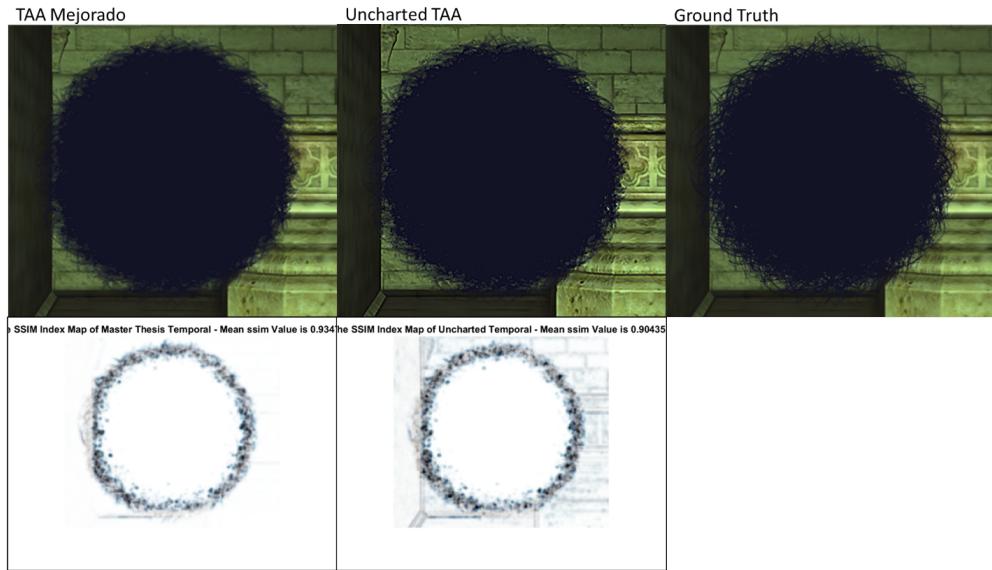


Figura 5.22: Comparación de *Ghosting* entre TAA del Proyecto de Grado, TAA de Uncharted y la Imagen Base en la Prueba Número 90.

### Prueba de Ghosting Con Luz

En la Figura 5.23 y la Tabla 5.13, observamos que ambas técnicas de TAA generan una gran cantidad de desenfoque alrededor de las fibras. En ambos Mapas de SSIM, el desenfoque es visible ya que Hairball se ve como una gran área oscura. Es importante

acotar que el Índice de Prueba en la Tabla 5.13 marca a las que prueba el resultado asociado pertenece; en los promedios que utilizamos No Aplicable (N / A) porque esos resultados provienen del promedio de todas las pruebas.

Tabla 5.13: Resumen de resultados numéricos de las 100 pruebas realizadas para la Prueba de *Ghosting* de Hairball Con Luz.

Resumen de Prueba de Ghosting de Hairball Con Luz				
Pruebas \ AA	Uncharted	Índice de Prueba	TAA Mejorado	Índice de Prueba
Pruebas	TAA	de Prueba	TAA Mejorado	Índice de Prueba
Mejor MSE	714.811	62	603.190	1
Peor MSE	980.701	83	749.516	99
Promedio de MSE	875.687	N/A	671.202	N/A
Mejor Peak-SNR	19.589	62	20.326	1
Peor Peak-SNR	18.215	83	19.383	99
Promedio de Peak-SNR	18.737	N/A	19.867	N/A
Mejor SNR	10.037	62	10.913	1
Peor SNR	8.666	83	9.902	99
Promedio SNR	9.261	N/A	10.390	N/A
Mejor SSIM	0.738	62	0.766	1
Peor SSIM	0.662	99	0.694	83
Promedio de SSIM	0.691	N/A	0.722	N/A

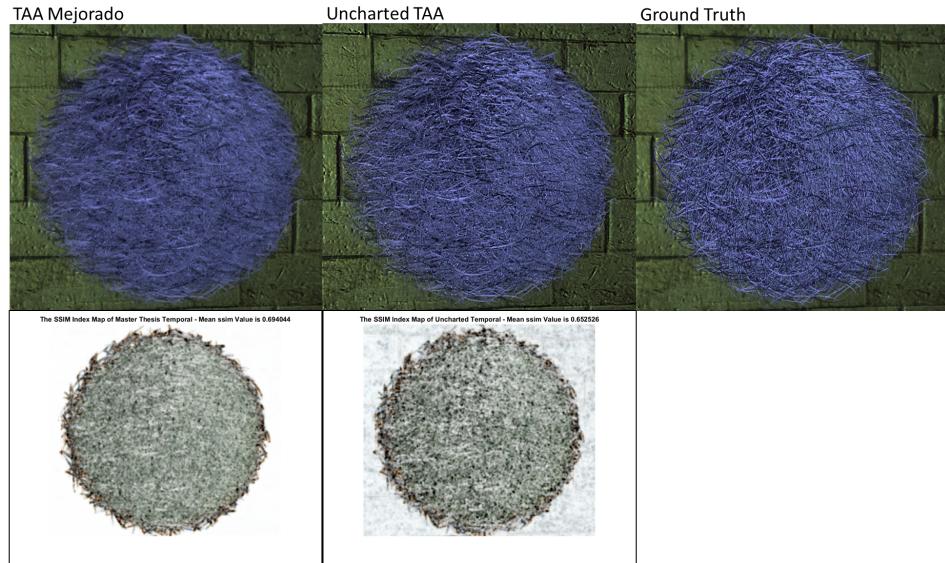


Figura 5.23: Comparación de *Ghosting* entre TAA del Proyecto de Grado, TAA de Uncharted y la Imagen Base en la Prueba Número 83.

### 5.2.10. Comparación de Tiempos

Es importante tener en cuenta que todas las técnicas probadas son de naturaleza de Post-Procesamiento, lo que significa que reciben la imagen de salida de la Arquitectura de *Shading Diferido* como entrada y no dependen de la complejidad de la escena.

El tiempo medido que tomó la técnica de TAA del Proyecto de Grado fue entre 0.5 y 0.6 *ms* en promedio. El paso de Sobel tomó entre 0.2 *ms* y 0.3 *ms*, y la reproyección fue de alrededor de 0.3 *ms*. El tiempo medido que tomó la técnica TAA de Uncharted fue de entre 0.2 *ms* y 0.3 *ms*; FXAA tomó entre 0.1 *ms* y 0.2 *ms*; y SMAA tomó 0.2 *ms* y 0.3 *ms*.

## 5.3. Discusión

Como apreciamos los resultados de la prueba del Filtro de Nitidez, ver Tabla 5.1, las mejoras logradas en este proyecto de grado van más allá de modificar el filtro de la implementación de Uncharted, ya que al cambiar este filtro sólo se evita generar esos píxeles brillantes incorrectos alrededor bordes. Creemos que el uso de ese filtro específico es de naturaleza artística, ya que tiende a pronunciar los bordes a costa de crear píxeles brillantes, que parpadean al moverse, alrededor de los bordes. Esto es especialmente notable cuando la cámara se mueve, mientras el primer plano está iluminado pero el fondo está en sombras o viceversa.

De ambas Pruebas de Tubería, podemos observar que los resultados del TAA del Proyecto de Grado están cerca de los resultados de SMAA al dibujar bordes duros. Podemos cuantificar la reducción del problema del desenfoque al comparar con la implementación de TAA de Uncharted en los resultados de las Tablas 5.2 y 5.3. Cuando comparamos los resultados de los Mapas de SSIM (Figuras 5.4 y 5.5), observamos una gruesa línea de error alrededor de los bordes en la implementación de Uncharted, la cual no está presente en nuestra implementación. Pero la reducción del desenfoque no es perfecta, como vemos en los puntajes de las pruebas, aun aparece alrededor de los bordes de nuestra implementación y le permite a SMAA obtener el puntaje más alto en algunas pruebas.

De las pruebas de Ventanas con Persianas y Ventana Arqueada, podemos apreciar cómo reaccionan las técnicas ante características pequeñas, casi de tamaño de píxel, como las persianas y las puertas de las ventanas de las pruebas. Aunque el TAA del Proyecto de Grado y SMAA aparecen numéricamente mejores que el TAA de Uncharted en las Tablas 5.4 y 5.5, no pueden reconstruir todos los pequeños detalles y dejan rayas finas

de píxeles que parpadean cuando la cámara se mueve. Creemos que, en este caso, admitir el desenfoque del TAA de Uncharted beneficia su aplicación final porque es mejor perder algunos pequeños detalles que tener muchos píxeles parpadeando cada vez que se mueve la cámara.

La Prueba del Atrium de Sponza nos muestra que el TAA del Proyecto de Grado es más que capaz de manejar una escena general con iluminación y sombras. Como se ve en la Tabla 5.6, nuestra implementación demostró ser mejor que las otras técnicas *Anti-Aliasing* por un margen justo en casi todas las pruebas.

Consideramos que la prueba de las Flores del Atrium de Sponza un experimento distinto al resto, pues las flores son una superficie plana 2D con muchos agujeros transparentes complejos que giran alrededor de la columna. Como observamos a partir de los resultados numéricos de la Tabla 5.7 y los Mapas de SSIM en la Figura 5.14, todas las técnicas tienen problemas con esos agujeros transparentes, pero nuestro implementación de TAA demuestra ser el mejor en el manejo ellos. Vemos en esta prueba que nuestra implementación es buena para manejar este tipo de pequeños detalles que, en comparación con las pruebas de las Ventanas, son más grandes que solo un píxel.

A partir de la prueba de Bordes Duros, seguimos observando que la técnica del Proyecto de Grado maneja mejor el desenfoque, en comparación al TAA de Uncharted (ver Tabla 5.8), esto es especialmente notable en las letras, la tubería y el cuadrado. También notamos que la técnica del Proyecto de Grado todavía tiene dificultades para manejar los detalles súper finos de las ventanas, a esta distancia todavía creemos que el desenfoque del TAA Uncharted ayuda a ocultar las rayas de píxeles no deseados, que parpadean al ocurrir movimiento(Ver Figura 5.15).

En la prueba de *Ghosting* de Esfera, vemos un claro ejemplo de las mejoras logradas en este proyecto de grado. La Figura 5.17 muestra un ejemplo del efecto de *Ghosting* creado por la implementación del TAA de Uncharted, podemos percibir claramente las franjas que deja la esfera mientras se mueve, mientras que en nuestra implementación de TAA son apenas visibles.

Finalmente, tenemos las cuatro pruebas de Hairball, que fueron las más complejas realizadas en este proyecto de grado. El modelo Hairball tiene muchos huecos y pequeños detalles que reaccionan a la iluminación y las sombras. Anticipamos una gran cantidad de errores debido a ellos porque todas las técnicas *Anti-Aliasing* tienen dificultades para reconstruir esta alta densidad de detalles finos.

En la prueba Estática Sin Luz y la prueba Estática Con Luz (Ver Tablas 5.10 y 5.11) podemos observar que la implementación del Proyecto de Grado es la mejor para renderizar las fibras de cabello. Es capaz de reconstruir bordes más suaves que el TAA

de Uncharted y reconstruye más detalles que SMAA. Especialmente, en la versión con luz (Ver Figura 5.20), podemos apreciar qué tan suave es el resultado; se ve casi como la imagen de referencia. Esta prueba superó, con creces, nuestra expectativa de las mejoras. Los resultados visuales y numéricos (Ver tabla 5.11) muestran un gran aumento en la calidad en comparación con las otras soluciones *Anti-Aliasing*.

En la prueba de *Ghosting* Sin Luz y Con Luz, observamos que los resultados de ambas técnicas presentan desenfoque en exceso (Ver Figuras 5.22 y 5.23), especialmente la implementación de la TAA del Proyecto de Grado en la versión con luz. Creemos que esto es causado porque el Buffer de Historia y Pase Temporal de Sobel que no pueden estabilizarse tan rápido como la velocidad de cambio de los colores cuando las fibras se mueven; esto es, el rechazo de color es más lento de lo necesario. Los valores numéricos de las Tablas 5.12 y 5.13 confirman los efectos del desenfoque, dado que el MSE es más alto de lo normal.

A partir de nuestros resultados de la comparación de tiempos (Consulte la sección 5.2.10), podemos ver que nuestras mejoras se ajustan a los requisitos de tiempo para ejecutarse en aplicaciones en tiempo real, ya que está por debajo del límite común de 1 *ms*.

# CAPÍTULO VI

## CONCLUSIONES Y RECOMENDACIONES

Como hemos demostrado numérica y visualmente con las pruebas realizadas, la implementación Mejorada del Proyecto de Grado logró su objetivo de reducir los efectos de desenfoque y los efectos de *Ghosting* de la técnica *Temporal Anti-Aliasing*, con el uso de detección de bordes de color y profundidad e indexación de triángulos. Nuestros resultados muestran que esta técnica puede proporcionar la misma o mejor calidad que otras soluciones estándar de *Anti-Aliasing*.

Como posibles mejoras, creemos en que nuestra implementación podría optimizarse para ejecutarse más rápido que nuestro tiempo actual. Nuestra implementación se realizó con flexibilidad en mente, para ayudarnos a probar diferentes enfoques. Esto podría simplificarse para reducir el número de etapas requeridas.

En cuanto a las recomendaciones para futuras investigaciones, sugerimos mejorar el comportamiento de la técnica bajo objetos en movimiento con alta densidad de detalles, como en las pruebas de Hairball. Otro tema de mejora es la estabilidad de los detalles del tamaño de un píxel, que causan parpadeos como en las pruebas de Ventanas. Además, todavía se requiere una solución de *Anti-Aliasing* de Iluminación Especular compatible con *Temporal Anti-Aliasing* para proporcionar una solución de rango completo al *Aliasing* en aplicaciones en tiempo real. Asimismo, sugerimos buscar métricas de imagen más específicas para gráficos por computadora, especialmente, encontrar valores de ajuste para SSIM que proporcionen una resolución numérica mayor al comparar diferentes imágenes procesadas

## 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*, Agosto 2012. Volúmen 3, Edición 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, Marzo 2010. Último Acceso: 2018-02-15, <https://www.turbosquid.com/3d-models/pipe-stone-bowl-3d-model/522479>.
- [5] J. Chapman. Per-Object Motion Blur, Septiembre 2012. Último Acceso: 2017-11-30, <http://john-chapman-graphics.blogspot.se/2013/01/per-object-motion-blur.html>.
- [6] Valve Corporation. Steam Hardware & Software Survey, Octubre 2018. Último Acceso: 2018-11-12, <https://store.steampowered.com/hwsurvey/>.
- [7] L. J. Fuglsang Pedersen. Temporal Reprojection Anti-Aliasing in INSIDE. *GDC Vault*, 2016. Último Acceso: 2017-11-28, <http://www.gdcvault.com/play/1022970/Temporal-Reprojection-Anti-Aliasing-in>.
- [8] Isabela H. Arched window model, Enero 2016. Último Acceso: 2018-02-15, <https://www.cgtrader.com/free-3d-models/architectural/window/window-arch>.
- [9] P. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. *ACM Computer Graphics*, Agosto 1990. Volúmen 24, Número 4, Agosto 1990.
- [10] 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.

- [11] 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.
- [12] T. Lottes. FXAA. *NVIDIA Docs*, 2009. The Document was last edited in 2011.
- [13] M. Doggett. EDAF80 Computer Graphics, Septiembre 2017.
- [14] M. Doggett. EDAN35 High Performance Computer Graphics, Noviembre 2017.
- [15] 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, Abril 2009.
- [16] M. McGuire. Computer graphics archive, Julio 2017. Último Acceso: 2018-02-15, <https://casual-effects.com/data>.
- [17] 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.
- [18] I. Sobel. An isotropic 3x3 image gradient operator. *Irwin Sobel Correspondence*, 02 2014.
- [19] B. Wronski. Temporal Supersampling and Antialiasing. <https://bartwronski.com/2014/03/15/temporal-supersampling-and-antialiasing/>, Marzo 2014. Último Acceso: 2017-12-01.
- [20] K. XU. Temporal Antialiasing In Uncharted 4. *SIGGRAPH 2016*, 2016.
- [21] 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.
- [22] C. Yim. Window with blinds model, Octubre 2015. Último Acceso: 2018-02-15, <https://www.cgtrader.com/free-3d-models/architectural/window/window-brown>.

# **APÉNDICE A**

## **REPOSITORIO DE GITHUB**

El enlace principal al repositorio es <https://github.com/maniatic0/Christian-TRAA>. Desde allí, se puede acceder a los siguientes enlaces para la versión impresa de este informe.

- Repositorio Completo: <https://github.com/maniatic0/Christian-TRAA>.
- Especificación Completa de la Computadora:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/PC%20Specification>.
- Todas las Pruebas:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/Important%20Tests>.
- Pruebas del Buffer de Acumulación:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/Important%20Tests/Accumulation%20Buffer%20Tests>.
- La Mayoría de las Pruebas del Proyecto de Grado:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/Important%20Tests/Master%20Thesis%20Tests>.
- Pruebas del Filtro de Nitidez:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/Important%20Tests/Sharpening%20Tests>.
- Pruebas de *Ghosting*:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/Important%20Tests/Ghosting>.
- Pruebas de HairBall:  
<https://github.com/maniatic0/Christian-TRAA/tree/master/Important%20Tests/HairBall>.

- Código:

[https://github.com/maniatic0/Christian-TRAA/tree/master/CG\\_Labs.](https://github.com/maniatic0/Christian-TRAA/tree/master/CG_Labs)

- Reporte de L<sup>A</sup>T<sub>E</sub>X:

[https://github.com/maniatic0/Christian-TRAA/tree/master/LaTeX/Master\\_Thesis\\_Spanish.](https://github.com/maniatic0/Christian-TRAA/tree/master/LaTeX/Master_Thesis_Spanish)

## APÉNDICE B

### EXPLICACIÓN DEL CAMERA JITTERING

La Secuencia de *Halton*(2, 3) genera puntos en el espacio  $[0, 1] \times [0, 1]$ . Primero, tenemos que transformarlo en el espacio  $[-1, 1] \times [-1, 1]$  porque consideramos que el píxel está en el centro, es decir, en OpenGL, el primer píxel está en  $(0.5, 0.5)$ . Por esto, aplicamos la transformación  $T_1(x, y) = 2 * (x, y) - (1, 1)$ .

Ahora, solo queremos causar *jittering* dentro del píxel porque solo deberíamos tomar muestras dentro de él. Queremos controlar este movimiento, pero para fines de explicación, podemos suponer que solo lo estaremos dentro del píxel. Entonces, aplicamos la transformación  $T_2(x, y) = \frac{(x, y)}{2}$ .

A partir de ahora, debemos cambiar la forma en que interpretamos el proceso que estamos realizando; estamos calculando la distancia a la que moveremos el centro de píxeles, por lo que debemos verlo como un vector en lugar de como un punto. Necesitamos transformar el vector en un vector normalizado por el tamaño de la pantalla. En consecuencia, aplicamos la transformación  $T_3(x, y) = (x, y) / (w, h)$  con  $(w, h)$  siendo el ancho y alto de la pantalla y el operador ”/” como división por componentes.

Ahora necesitamos transformar nuestro vector normalizado por el tamaño de pantalla  $[0, 1]$  a las Coordenadas Normalizadas del Dispositivo (*Normalized Device Coordinates*, NDC) que van en el rango  $[-1, 1] \times [-1, 1]$ . Esto se hace usando la transformación  $T_4(x, y) = 2 * (x, y)$  (para transformar puntos, usariamos  $2 * (x, y) - (1, 1)$ ). Al final, nuestra transformación se vería así  $T(x, y) = T_4(T_3(T_2(T_1(x, y)))) = (2 * (x, y) - (1, 1)) / (w, h)$  con  $(x, y)$  siendo un punto de la Secuencia de Halton.

Ahora necesitamos modificar nuestra Matriz de Proyección, que toma puntos del espacio de visualización en el Espacio de Clip (*Clip Space*, donde se eliminan fácilmente los vértices trivialmente no visibles). La matriz resultante se toma de la página 14 de presentación de Ke Xu [20]. Sea  $(h_x, h_y)$  el *jittering* que hemos calculado previamente.

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

Veamos su efecto a un punto en el *View Space* (visión desde perspectiva de la cámara), sea  $p_{view} = (x, y, z, 1)$ .

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

Procedemos a hacer la *División en Perspectiva* (corrige la perspectiva a la deseada en la cámara) para transformar el punto en Coordenadas NDC, esto se logra dividiendo el vector por el componente w.

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

En consecuencia, solo necesitamos estar seguros de que podemos aplicar la Matriz de *Jittering*, para usarla dentro de *Pixel Shaders*, con puntos en el espacio NDC.

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