Muestreo Mejorado para Temporal Anti-Aliasing

(Un Temporal Anti-Aliasing Mejorado con Sobel)

Christian Alexander Oliveros Labrador
christianol_01@hotmail.com

30 de junio de 2018

Proyecto de Grado realizado en
el Departamento de Ciencias de la Computación de la Universidad de Lund, en conjunto con la Coordinación de Ingeniería de la Computación de la Universidad Simón Bolívar.

Supervisores: Ángela Di Serio, adiserio@ldc.usb.ve
Michael Doggett, michael.doggett@cs.lth.se

Examiner: Flavius Gruian, Flavius.Gruian@cs.lth.se

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 Coloreado Diferido. Una solución innovadora a este problema es la técnica de *Temporal Anti-Aliasing* (TAA) que combina muestras de cuadros previos con las muestras del cuadro actual para aumentar, efectivamente, la tasa de muestreo. En esta tesis, exploraremos métodos para mejorar la calidad del TAA mediante el uso de detección de bordes, tanto de color como de profundidad, y la indexación de triángulos para asegurar que solo las muestras pertenecientes a los píxeles del cuadro actual se combinen. Nuestro objetivo es reducir el efecto de ghosting y otros artefactos TAA creados con las implementaciones actuales. 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 altos, 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 mejoraron la calidad de la imagen por encima de las técnicas existentes, con valores de PSNR en torno a 39 y valores de SSIM por encima de 0,99.

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

Agradecimientos

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

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 solo 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, esta 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 todos 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 aprovechen este tipo de oportunidades de desarrollo académico y de auto crecimiento.

LUND UNIVERSITY

UNIVERSIDAD SIMÓN BOLÍVAR

Índice general

Capítulo 1
Introducción

La Computación Gráfica moderna se basa en la representación de escenas que están hechas de objetos representados como modelos compuestos de polígonos primitivos, siendo el triángulo el más común. Esto es para aprovechar su simplicidad y todas sus propiedades geométricas para crear algoritmos óptimos que manejan su renderizado. Los triángulos están compuestos por tres vértices, cada uno de los cuales consiste en una posición y otros parámetros asociados a ellos, es decir, el color o las normales del triángulo, para la interpolación.

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

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

El objetivo de este proyecto es mejorar el algoritmo *Temporal Anti-Aliasing*, que es una 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 requisito sería proporcionar la representación de la escena con la mayor calidad posible, con dos restricciones principales: debemos renderizar al menos treinta cuadros por segundo, sin causar pérdida de velocidad al renderizar los cuadros; y debemos trabajar con una cantidad limitada de memoria y ancho de banda, porque necesitamos poder ejecutar en una computadora promedio o un dispositivo móvil. [13, 3]

1.1 Definición del Problema

Temporal Anti-Aliasing (TAA) es una técnica en tiempo real, relativamente nueva, que proporciona buenos resultados sin incurrir en gran consumo de memoria o costos de pro-

cesamiento de otras técnicas. Las técnicas de detección de bordes y de indexación de triángulos, parecen ser buenos candidatos para mejorar la calidad de TAA al ayudar a reducir los efectos no deseados de *ghosting* y desenfoque creados por las implementaciones actuales de TAA.

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

1.2 Trabajos Relacionados

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

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

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

Finalmente, tenemos las implementaciones TRAA de Ke Xu para Uncharted 4 y 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 esta tesis. [6, 19]

1.3 Contribución

Esta tesis mejora los últimos dos avances de Ke Xu y Lasse Fuglsang [6, 19]. Proponemos el uso del Operador de Sobel para realizar detecciones de bordes y técnicas de indexación

de triángulos para detectar píxeles que se consideran con aliasing. Una vez que tenemos estos píxeles detectados, usamos esta información para cambiar la forma en que se rechazan los colores de los cuadros anteriores a fin de reducir los artefactos de *ghosting* y desenfoque que tiene *Temporal Anti-Aliasing*.

Capítulo 2
Marco Tecnológico

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

2.1 C++ y el Bonobo Framework

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

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

2.2 OpenGL y GLSL

Usamos esta *Application Programming Interface* (API, Interfaz de Programación de Aplicaciones) ya que es la base del sistema de renderización del Bonobo Framework, el cual modificamos para desarrollar nuestras mejoras, y debido a su compatibilidad multiplataforma. La *Open Graphics Library Library* (OpenGL, Librería Abierta de Gráficos) es una API de computación gráfica de código abierto, multiplataforma, que maneja 2D y 3D, que abstrae 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*) para usar, normalmente se implementado directamente en hardware.

Usamos el *OpenGL Shading Language* (GLSL, Lenguaje de Coloreado de OpenGL) para implementar TAA y nuestras mejoras, ya que es parte del estándar de OpenGL. GLSL es un lenguaje de coloreado 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 para que fuera 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 3

Marco Teórico

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

3.1 Canal de Renderizado

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

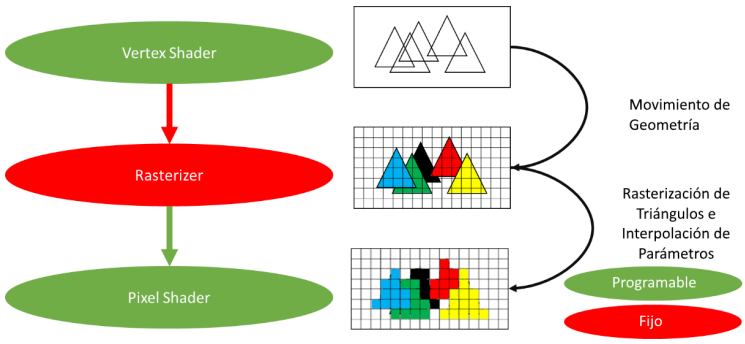

Figura 3.1: Canal de Renderizado, basado en la 5ta clase de EDAF80. [12]

Es importante tener en cuenta que las etapas de *Vertex* y *Pixel Shaders* son controlables por el programador usando programas especiales llamados *Shaders*; estos proporcionan una forma para controlar el hardware de renderizado. Por el contrario, el Rasterizer no

está controlado por el programador y se maneja por completo mediante una función fija de hardware. [12]

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

3.2 Proceso de Rasterización

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

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*.

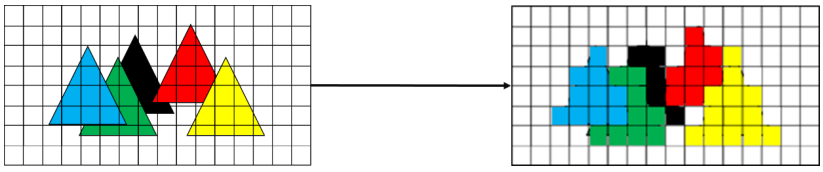

Figura 3.2: Ejemplo de los resultados del Proceso de Rasterización. *Nota*: se agregaron colores para diferenciar los triángulos, pero solo se agregarían en la etapa del *Pixel Shader*.

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.

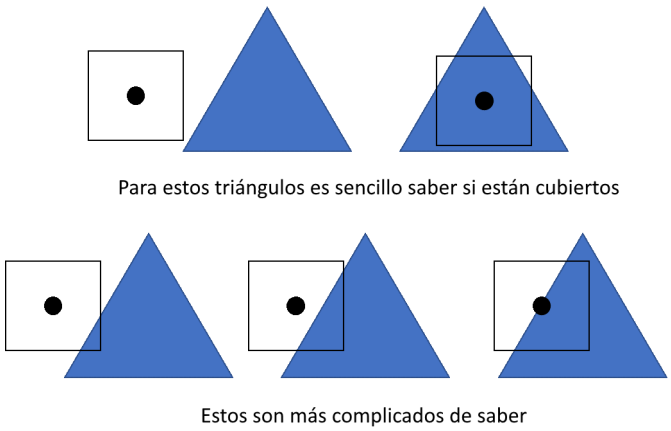

Figura 3.3: Ejemplo del problema de Cobertura Parcial, basado en la segunda clase de EDAN35. [13]

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

3.3 El Problema del Aliasing

Cuando mapeamos una representación continua a una finita vamos a generar errores. Como explican Edward Angel y Dave Shreiner en su libro (página 413) [3], podemos interpretar el proceso de renderización como el muestreo de una función continua $f(x, y)$, que representa el color de la escena en ese punto, a una cuadrícula de píxeles $n \times m$ en la que suponemos que el punto f_{ij} es el valor de f sobre un área pequeña; reconstruyendo la función f, para mostrar la imagen en la pantalla, usando solo lo que sabemos de las muestras finitas. La herramienta matemática utilizada para evaluar los problemas de este proceso es el Análisis de Fourier, que establece que una función se puede descomponer en un conjunto de sinusoides, posiblemente en un número infinito de frecuencias. Para el análisis de imágenes bidimensionales, podemos pensar en la función f como un conjunto de sinusoides en dos frecuencias espaciales. Para esta tesis, usaremos la primera parte del Teorema de Muestreo de Nyquist como una herramienta para ilustrar por qué aparecen problemas de aliasing y se relacionan con problemas de muestreo.

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

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

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

Como explican Edward Angel y Dave Shreiner, este muestreo idealizado supone que podemos tomar un número infinito de muestras por frecuencia de muestreo que no podemos hacer en la práctica. El Problema de *Aliasing* que la Computación Gráfica experiencia proviene de no poder muestrear según lo requerido por el Teorema de Muestreo de Nyquist, creando bordes desiguales que aparecen en el proceso de rasterización (*Aliasing* Espacial), la Figura 3.4 muestra un ejemplo de este tipo de aliasing y saltos entre objetos en movimiento (*Aliasing* Temporal), de acuerdo con Doggett y Wronski [13, 18]. Se han propuesto y utilizado muchas soluciones para resolverlo, por ejemplo la familia de soluciones *Super Sampling Anti-Aliasing* (SSAA) que trabaja en frecuencias más altas que las requeridas a costa de más requisitos 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 Coloreado Diferido

Como humanos, hemos llegado a esperar que los objetos reaccionen 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, especialmente, las sombras nos dan una sensación de tamaño y distancia.

Según lo explicado por Doggett [13], bajo de la Canal de Renderizado Gráfico basado en el Rasterizador, el proceso de cálculo de la sombra es difícil de realizar. El *Rasterizer* no sabe si los objetos están cubiertos o no de una luz, por lo que debemos encontrar un método para calcular si un objeto está en sombras. Este proceso se denomina Mapeado de Sombras, consiste en representar la escena a través de la perspectiva de cada 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, entonces necesitamos una forma de reducir el costo tanto como podamos. La Arquitectura de Coloreado Diferido proporciona esa solución solo para colorear, posiblemente utilizando una operación que podría ser costosa, solo píxeles visibles y, así, evitar desperdiciar recursos coloreando píxeles de geometría que no es visible. Esta arquitectura funciona renderizando primero la escena, sin cálculos de coloreado, 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 Coloreado Diferido para calcular la forma en que las luces afectan solo a los píxeles visibles; calculamos el Mapa de Sombreado de cada luz solo realizando cálculos de profundidad. Después, calculamos y guardamos el efecto de cada luz usando los Mapas de las Sombreado.

Al final, tomamos toda la información de las luces, las 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 que decide qué técnica *Anti-Aliasing* usar es cómo se comporta con las arquitecturas actuales. Por ejemplo, las viejas soluciones Anti-Aliasing no funcionan con el Coloreado Diferido.

3.5.1 Super Sampling Anti-Aliasing (SSAA)

Esta técnica consiste en renderizar la escena a 4 veces el tamaño de la pantalla y luego promediar píxeles, en un área 4x4 para cada pixel, para calcular el resultado [13]. Proporciona buenos resultados, pero requiere más tiempo de renderizado y un gran uso de memoria.

3.5.2 Multi Sample Anti-Aliasing (MSAA)

MSAA consiste en tomar varias muestras por píxel; en cada muestra, los valores de profundidad se calculan, pero solo se calcula 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 Coloreado Diferido [13]. Esto hace que sea complicado utilizarlo con los Canales de Renderización actuales, que normalmente requieren otras correcciones para reducir los artefactos creados cuando se aplica con Coloreado Diferido.

3.5.3 Fast Approximate Anti-Aliasing (FXAA)

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

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

3.5.4 Enhanced Subpixel Morphological Antialiasing (SMAA)

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

3.6 Temporal Anti-Aliasing

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

Una de esas técnicas es el *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 lograr esto, 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 *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 para recopilar los pixeles 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 es estático en relación con los pasados, entonces el sistema está perdiendo información subpíxel que podría usarse para refinarla. [6, 19]

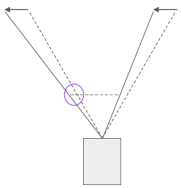

Figura 3.5: Proceso de *Jittering* sobre la proyección de la cámara.
Imagen tomada de la presentación de Fuglsand. [6]

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

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

3.6.2 Buffer de Velocidad

El algoritmo de Buffer de Velocidad (Velocity Buffer) utilizado en esta implementación es el propuesto por Chapman [5], el cual es calculado restando la posición actual del píxel por 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

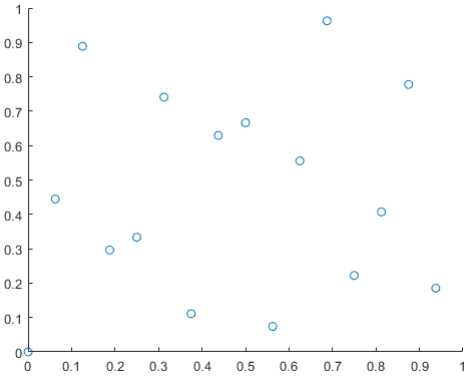

Figura 3.6: Valores usados de la $HaltonSequence(2, 3)$ (Secuencia de Halton).

Buffer de Velocidad. Además, cancelamos matemáticamente el Cámara Jittering antes de calcular las velocidades, para evitar el ruido que crea, como lo sugiere Xu. [19]

3.6.3 Buffer de Historia

Para cada fragmento en el marco actual, buscamos en la vecindad de 3×3 y en la vecindad del patrón más $(+)$ 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 para restringir los píxeles del Buffer de Historia. [6]

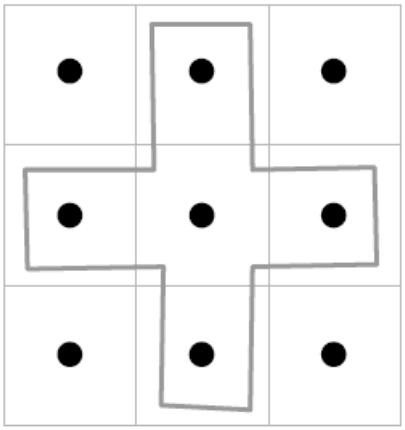

Figura 3.7: Patrón de Muestreo utilizado. Imagen tomada de la presentación de Fuglsand. [6]

En el vecindario 3×3 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 ocluidos [6]. Usamos esta velocidad para reproyectar la posición del píxel del cuadro actual en el Buffer de Historia. [6, 19]

Después de tener los píxeles en el Buffer de Historia, lo restringimos (explicación en la siguiente subsección) y lo interpolamos linealmente con el cuadro actual (Ver la Figura 3.8 para obtener una representación visual del proceso completo). Interpolamos linealmente ambos 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 cierta 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 [6, 19]. Dado que el historial de cada píxel se acumula en el Buffer de Historia, obtenemos el efecto de que el historial de píxeles de los cuadros anteriores pesa menos cuanto más tiempo sea mantenido el historial del píxel dentro Buffer de Historia. [6]

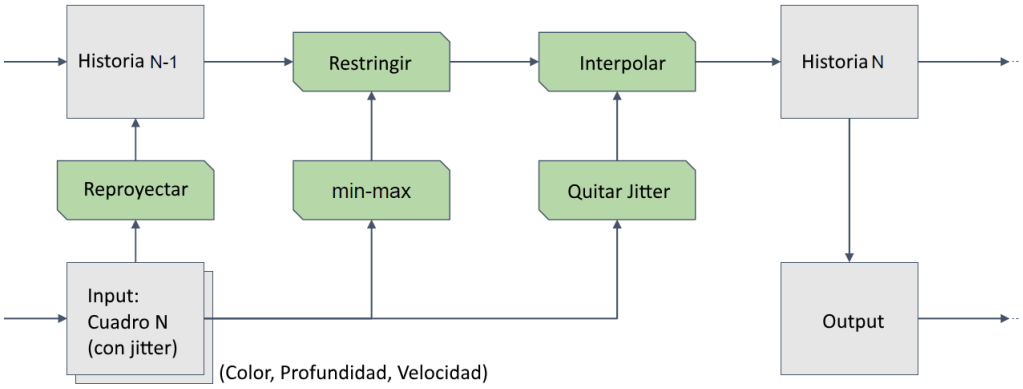

Figura 3.8: Proceso de *Temporal Reprojection Anti-Aliasing*. Imagen tomada de la presentación de Fuglsand [6]

3.6.4 Clipping Color Box

Una *Clipping Color Box* (Caja de Recorte de Colores) a es una caja 3D construida utilizando el color de píxel actual como el centro y el color mínimo y máximo calculado en la última subsección como los límites. Se utiliza para manejar el rechazo de la historia cuando el color del píxel en el Buffer de Historia está demasiado alejado del color actual. Tomamos el color de la historia del como un vector y luego lo proyectamos contra los límites de la caja; si se encuentra fuera del borde de la caja, entonces mantenemos la proyección; de lo contrario, el color de la historia quedará intacto. El uso del Clipping Color Box evita la agrupación de colores en las esquinas que ocurriría si se aplicara *clamping* (Ver Figura 3.9). [6].

3.6.5 Filtro de Nitidez

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

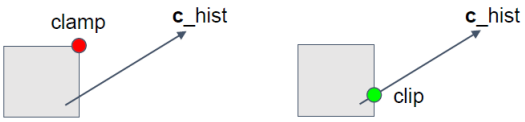

Figura 3.9: *Color Clamping* versus *Color Clipping*. Imagen tomada del *paper* de Fuglsand. [6]

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

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

3.6.6 Motion Blur

Because of the nature of the History Buffer, some ghosting is created by fragments from objects that are moving faster than the time it takes for the Color Clipping Box to reject the color from old pixels, this is especially noticeable under special light and background conditions. Fuglsand and Xu [6, 19] proposed to use Motion Blur solutions to hide these artifacts.

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

3.6.7 Problems

Temporal Anti-Aliasing has two main drawbacks, ghosting from moving objects and blurring from the way the Color Clipping Box works. This master thesis aims to help reduce the effects of these two drawbacks using new approaches but, for completeness, we present some of the current solutions available.

3.6.7.1 Blurriness

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

3.6.7.2 Ghosting

Some Ghosting is created when objects move, especially under particular light and background conditions that make the foreground and background look alike. This is partially corrected with motion blur, nevertheless, some of it remains near objects that move fast enough to create some Ghosting but slow enough to avoid Motion Blur. Xu proposes the use of Motion Blur and to increase the size of everything using a Stencil technique and manual tagging of objects [19], but we aim to avoid requiring artists to manually tag and test objects for their ghosting behaviors. Pederson's implementation allows the jitter in the Velocity Buffer calculations to avoid ghosting but it creates some unwanted blurriness [6].

3.7 Accumulation Buffer

The Accumulation Buffer is an anti-aliasing technique that consists, according to Paul Haeberli and Kurt Akeley [8], on rendering the scene several times with camera jittering and then performing a scaled weighted sum of the renderings to generate the current frame.

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

3.8 Sobel Operator

The Sobel Operator is an efficiently computable 3×3 isotropic gradient operator, as explained by Irwin Sobel [17]. We use this operator to detect edges in the rendered images to mark the as possibly aliased, because they are a common place for aliasing artifacts to appear.

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

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

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

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

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

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

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

Equation 3.4 can be expressed in two weighting matrices. We use Matrix 3.5 for the x component and Matrix3.6 for the y component.

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

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

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

3.9 Image Metrics

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

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

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

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

3.9.1 Mean Square Error (MSE)

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

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

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

3.9.2 Root Mean Square Deviation (RMSD)

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

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

3.9.3 Peak Signal-to-Noise Ratio (PSNR)

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

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

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

3.9.4 Structural Similarity Index (SSIM)

SSIM is a widely used image metric that matches human subjectivity and is highly sensitive to degradations in the spatial structure of image luminance, as explained by Malpica and Bovik. [14]

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

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

Where

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

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

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

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

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

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

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

Capítulo 4
Development

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

4.1 EDAN35 Project Improvements

For this master thesis, we decided to use our project from High Performance Computer Graphics (EDAN35) as the base. During the course, we implemented a Temporal Reprojection Anti-Aliasing technique based on the presentations of Ke Xu and Lasse Fuglsang [19, 6], from the games Inside and Uncharted 4. It proved to be reliable and well documented; and allowed us to put into practice the fundamentals of the technique in an academic environment, providing the base for improvements performed for this master thesis.

The EDAN35 project implementation had errors in the jittering process that were corrected by properly expanding what both implementations meant by camera jittering , see Appendix B for a full explanation. The management of the Halton points was redone to accomplish the improved camera jittering with the inclusion of support of up to 128 points to work as the jittering of the Accumulation Buffer (See Figure 4.1). Note, though, for Temporal Anti-Aliasing only the first 16 points are used as suggested by Xu and Fuglsang. [19, 6]

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

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

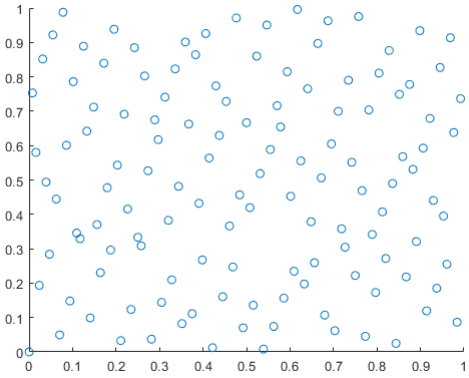

Figura 4.1: The 128 $HaltonSequence(2,3)$ points available to use.

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

4.1.1 Fast Approximate Anti-Aliasing (FXAA)

For this master thesis, the white paper version of Fast Approximate Anti-Aliasing (FXAA) by Lottes [11] was used to compare it against Temporal Anti-Aliasing, on the ground that both techniques are Post-Processing Anti-Aliasing and that FXAA is a popular technique used in the industry. It was implemented under the highest quality preset according to the white paper without taking into consideration the performance impact because we wanted to compare the raw improvement that both techniques can provide.

4.1.2 Enhanced Subpixel Morphological Antialiasing (SMAA)

In order to test a more complex and newer post-processing technique, SMAA was implemented following the instructions provided by Jimenez et al. [9]. It was implemented using the highest preset that works with the deferred shading pipeline.

4.1.3 Accumulation Buffer

We use an Accumulation Buffer to provide a reference image of what the scene is. It was implemented following Haeberli and Akeley [8]. The points used to jitter the camera are Halton Sequence Points as for Temporal Anti-Aliasing, although the Accumulation Buffer can use up to 128. The reasons for using Halton Sequence are that: it fulfils the requirements established by Haeberli and Akeley; it is easy to extend the current camera system to support more Halton Points; and for Temporal Anti-Aliasing, it provides pseudo-random points that do not follow a pattern to help gather as much information of the scene as possible.

4.1.3.1 Sample Number Selection

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

In order to show the difference between using 16 and 128 samples, we performed four tests, named from A to D, to observe how the metrics behave under different arrangements of scene objects and lighting. In the Table 4.1 we see the results of one such test:

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

Test D				
Samples / **Tests**	**16**	**128**	**Difference**	**Relative Difference** (%)
MSE of Temporal	40,647863	38,947297	−1,700566	4,183654132 %
RMSD of Temporal	6,375568	6,240777	−0,134791	2,114180258 %
MSE of No AA	24,872104	24,524992	−0,347112	1,395587603 %
RMSD of No AA	4,987194	4,952271	−0,034923	0,700253489 %
Peak-SNR of Temporal	32,040426	32,22603	0,185604	0,575944353 %
SNR of Temporal	30,30596	30,492374	0,186414	0,611346299 %
Peak-SNR of No AA	34,173678	34,234715	0,061037	0,178289786 %
SNR of No AA	32,439212	32,501058	0,061846	0,190289190 %
SSIM of Temporal	0,993302	0,993451	0,000149	0,014998223 %
SSIM of No AA	0,996826	0,996891	$6{,}5E-05$	0,006520272 %

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

4.2 Testing Framework Implementation

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

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

4.2.1 Static Test

This type of test consists of letting the History Buffer fill for 16 frames by rendering the scene without any moving object with the Temporal Anti-Aliasing technique selected and then saving the last frame rendered. Immediately after saving the last TAA frame rendered, we render the last frame again but now using the Accumulation Buffer, to generate the ground truth image of the scene, and SMAA, FXAA and No Anti-Aliasing (No AA), for comparison purposes.

4.2.2 Ghosting Test

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

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

After all the images rendered are saved, we compare the ones produced by the Master Thesis Temporal Anti-Aliasing and the Uncharted Temporal Anti-Aliasing against the ground truth to calculate the image metrics of them. The image metrics we calculate show how much error was generated by ghosting in both implementations, allowing us to compare how both techniques behave.

4.2.3 MATLAB Image Metrics

The images metrics for each test represent, numerically, the quality of each image. We calculate them to compare how each technique behaved on each test in order to identify if the Master Thesis Temporal Anti-Aliasing generates higher quality images than the other techniques tested.

Once all the test results are saved, one script takes all the images and organizes them into folders by name and type of test. Afterward, all the test results are processed using MATLAB to get the images metrics results. With the image metrics results, we compare how the Mater Thesis Temporal Anti-Aliasing behaved against the Uncharted 4 Temporal Anti-Aliasing and the other Anti-Aliasing techniques, to detect if the improvements worked and to observe how our improved implementation behaved against other common Anti-Aliasing techniques.

For Static Tests, we perform MSE, RMSD, PSNR, SNR and SSIM measurements to the TAA, FXAA, SMAA and No AA rendered images, using the Accumulation Buffer rendered images as the reference to compare with. Also, we generate a local SSIM map for

each rendered image, apart from the one rendered with the Accumulation Buffer, which are saved as PNGs and MATLAB FIGs. All the measurements results are stored in the folder with the test results as plain text.

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

4.3 Temporal Reprojection Anti-Aliasing Modifications

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

The first change consists in that the Colors that were calculated from the average between the 3×3 and Cross neighborhoods by the sampling patterns (See Figure 4.2) are now mixed in a variable amount. This follows the idea that we prefer the Cross neighborhood if the pixel we are currently calculating is not considered to be aliased because the Cross neighborhood is less likely to introduce noise in the Color Clipping Box calculations. But, if the pixel is considered to be aliased, we prefer the 3×3 neighborhood because it provides more information about the surroundings of the pixel to create the unaliased image.

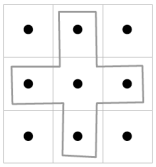

Figura 4.2: Sampling Pattern used. Image taken from Fuglsand presentation. [6]

The second change is that the size of the Color Clipping Box depends on how much the pixel is considered to be aliased. By using smaller Color Clipping boxes (See Figure 4.3) than the original technique on unaliased pixels, we increase the elimination of unwanted colors from the history, reducing the effects of ghosting since we are rejecting colors faster on pixels that we know are not considered to be aliased. This is implemented by linearly interpolating the current pixel color and the minimum and maximum colors calculated for the Color Clipping Box.

On the next subsections, we explain how all the values are calculated that contribute to decide if a pixel is considered aliased or not. All of these values are calculated at the same time as TAA is being applied, with the exception of the Sobel Operator which happens

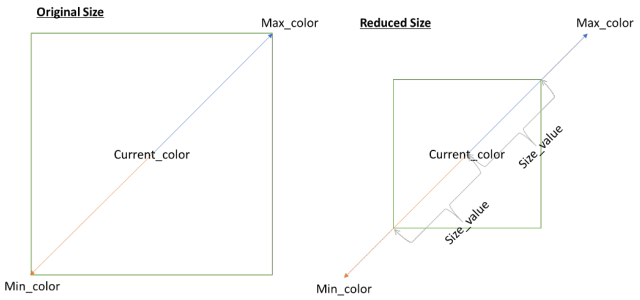

Figura 4.3: Color Clipping Box size reduction

before the main TAA technique. Afterward, we show how we use that value to reduce the size of the Color Clipping Box and the preference of the Sampling Pattern.

4.3.1 Triangle Indexing Improvements Implementation

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

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

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

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

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

Where

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

Next we are going to show some examples of how this technique works. For each matrix, the numbers represent the triangles ID's with the center position being the current

pixel being calculated and the rest being its neighborhood. Matrix 4.2 shows an example of a pixel not considered aliased because most of the neighborhood have the same ID. On the other hand, Matrix 4.3 shows a pixel considered aliased because of the high amount of different ID's around.

$$\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 5 & 23 & 82 \end{bmatrix} \tag{4.2}$$

$$\begin{bmatrix} 3 & 3 & 3 \\ 3 & 5 & 3 \\ 5 & 23 & 82 \end{bmatrix} \tag{4.3}$$

4.3.2 Depth Pseudo-Variance and Depth Temporal Pseudo-Variance

The key point behind this technique is that we want to detect pixels that we consider aliased because they are in a neighborhood of pixels separated by relatively long depth distances or that their depth in the last frame changed relatively a long distance in contrast to its current neighborhood.

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

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

We calculate the normalizing value using the minimum to avoid the interference of outliers. If $maxDepthDistance$ is below $0{,}002$ everything that follows is set to zero because pixels are so close that it is probably not aliased. This threshold was defined by experimenting which values would not get noise inside the calculations but would let the interesting pixels in.

Then we calculate the depth pseudo variance as:

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

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

Finally, we calculate how the depth of the pixel in the last frame relates to the current neighborhood, using the depth temporal pseudo variance. Note that we use the fourth power to reduce the noise in the calculations because the normalized value is between 0 and 1, therefore, making it converge to 0 if the value was near 0.

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

Next, we are going to show examples of how this technique decides if a pixel is considered aliased or not. For each matrix, the numbers represent the triangles depths with the center position being the current pixel being analyzed and the rest being its neighborhood. Matrix 4.7 shows us an example of a pixel not considered aliased because it is relatively close to most of its neighborhoods. In contrast, Matrix 4.8 shows an example of a pixel that would be considered aliased because it has a relatively big distance in comparison to its neighborhood. An example of a pixel considered aliased by the Depth Temporal Pseudo-Variance would be if the depth of the pixel of Matrix 4.7 in the last frame was $4{,}0$.

$$\begin{bmatrix} 9{,}0 & 9{,}3 & 8{,}7 \\ 9{,}2 & 9{,}0 & 9{,}3 \\ 8{,}8 & 8{,}9 & 8{,}7 \end{bmatrix} \tag{4.7}$$

$$\begin{bmatrix} 9{,}0 & 9{,}3 & 8{,}7 \\ 9{,}2 & 4{,}0 & 9{,}3 \\ 8{,}8 & 8{,}9 & 8{,}7 \end{bmatrix} \tag{4.8}$$

4.3.3 Sobel Improvements Implementation

The main idea behind applying Sobel edge detection technique is to concentrate the TAA effects on pixels on edges to correct aliasing, if necessary. It is important to note that this is the only technique from the improvements of the master thesis that runs before the main TAA algorithm.

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

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

$$g = (u * 0{,}3 + l * 0{,}7) + d \tag{4.9}$$

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

Then, we clamp g between $0{,}0$ and $1{,}0$ to finally apply the smoothstep polynomial as follows:

$$sobel = \sqrt{g^2 * (3{,}0 - 2{,}0 * g)} \tag{4.10}$$

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

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

4.3.4 Final Mixing

Finally, we modify how the Clipping Box is calculated using the values we previously calculated. We call this final mixed value $aliasedValue$, it represents how much a pixel is considered to be aliased. After it is calculated, we use it to change the Sampling Pattern from which colors the Color Clipping Box is built upon and its size.

First , we define the mix function as follows:

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

Then, the final mixing is applied as follows:

$$sobelAvgMixVal = Clamp01(modelAverage_i + sobel) \tag{4.12}$$

Where $sobel$ is the Sobel value of the current pixel, Equation 4.10, $modelAverage_i$ comes from the Equation 4.1 and $Clamp01$ is the clamping function between 0 and 1.

$$sobelAvg = Mix(sobelAvgCross, sobelAvg3x3, sobelAvgMixVal) \tag{4.13}$$

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

And, we use Equations 4.13, 4.5, 4.6 and 4.1 to calculate how much this pixel is considered to be aliased:

$$\begin{aligned} aliasedValue = Clamp01(&sobelAvg + depthPseudoVariance \\ &+ depthTemporalPseudoVariance \\ &+ modelAverage_i) \end{aligned} \tag{4.14}$$

With this $aliasedValue$ we proceed to modify the Color Clipping Box. First, we select how much of each Sampling Pattern we want to be part of the Color Clipping Box:

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

Then we modify the size of the Color Clipping Box by interpolating towards the normal size if the $aliasedValue$ is close to one, else, we use the current color, which is the center of the box.

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

The Figures 4.4, 4.5 and 4.6 are examples of the aliased values calculated, each pixel is the current $aliasedValue$ of that picture. The white color represents an $aliasedValue$

of 1 and the black color of 0. As we expect, most the edges are marked as probably aliased by the techniques we used.

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

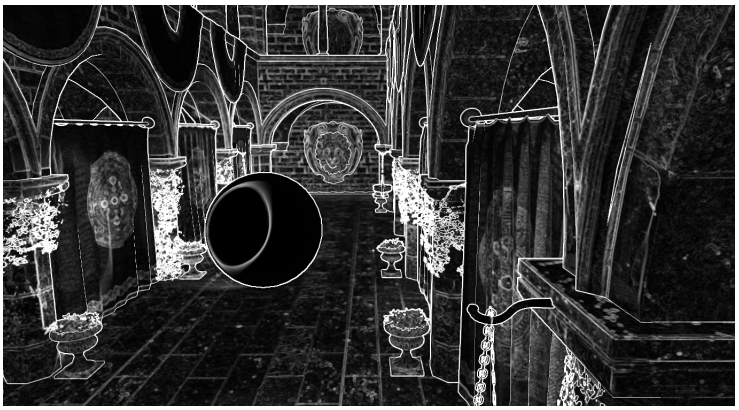

Figura 4.5: Image made of the Aliased Values of each pixel.

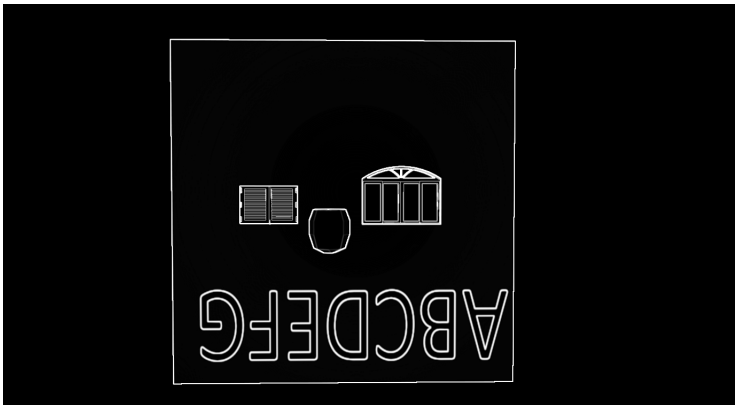

Figura 4.6: Image made of the Aliased Values of each pixel.

4.3.5 Sharpen Filter Modifications

With the current The Sharpen Filter, a color peak is created when the center pixel is bright and the neighboring pixels are dark, this is because dark colors are close to zero but the center pixel is still multiplied by 5. An example of the worst possible case is having the center pixel as a bright color and the neighborhood's pixels as pure black, which are represented as zero. Therefore, we normalized Sharpen Filter to avoid creating that color peak and to work better with bright pixels with dark neighborhoods. The Filter was changed from 3.1 to:

$$\begin{bmatrix} 0 & -0{,}25 & 0 \\ -0{,}25 & 2 & -0{,}25 \\ 0 & -0{,}25 & 0 \end{bmatrix} \tag{4.17}$$

Equation 4.17 shows the new Sharpen Filter Convolution Matrix used.

Capítulo 5
Results

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

5.1 Evaluation Methodology

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

The models used on the tests were:

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

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

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

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

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

5.2 Results and Comparisons

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

5.2.1 Sharpen Filter

For this test, we rendered the Sponza Atrium Model and a Static Sphere Model to evaluate the effects of the Normalized Sharpen Filter to the general quality of the rendered image, especially, regarding the blurring normally caused by TAA. Everything was rendered using both implementations of TAA with and without the Normalized Sharpen Filter to see if this change was the only improvement that was increasing the quality of the rendered image. The scene was selected for the test because it provided an example of a general scene that should not generate any blur. As we see from the Table 5.1, even if the Master Thesis TAA and the Uncharted TAA used the new Sharpen Filter, our other improvements still rendered a higher quality image. If we zoom on Figure 5.1 From Figure 5.2 we can see that the Normalized Sharpen Filter contributes reducing the artifacts in the rendered image, as there are almost no dark areas on the SSIM maps of the TAA's using it.

Cuadro 5.1: Sharpen Filter Test numerical results

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

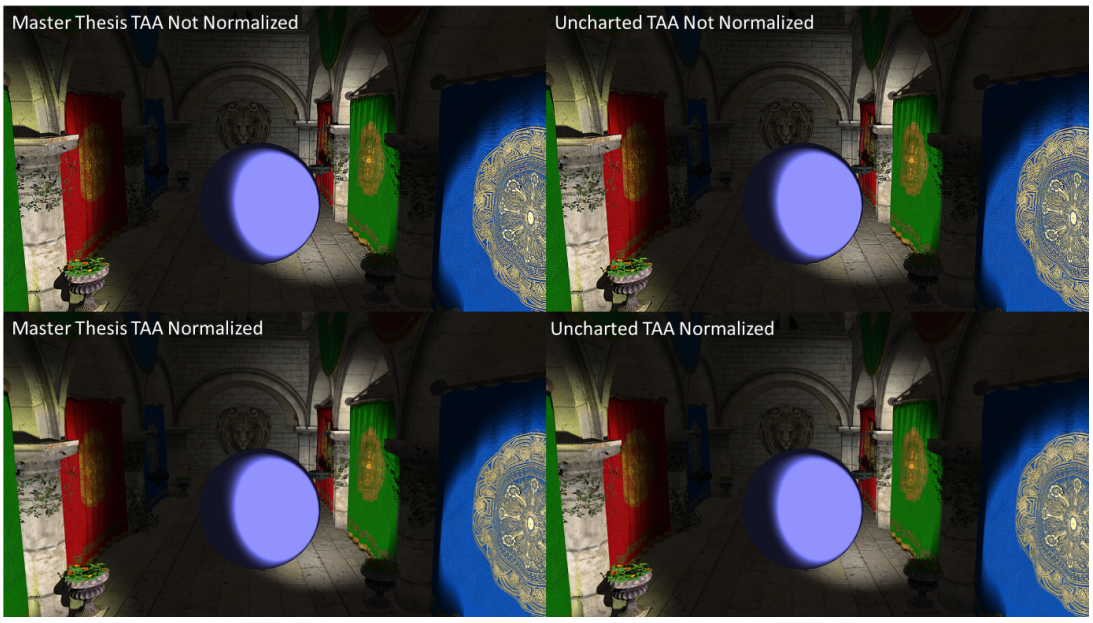

Figura 5.1: Rendered Images comparison.

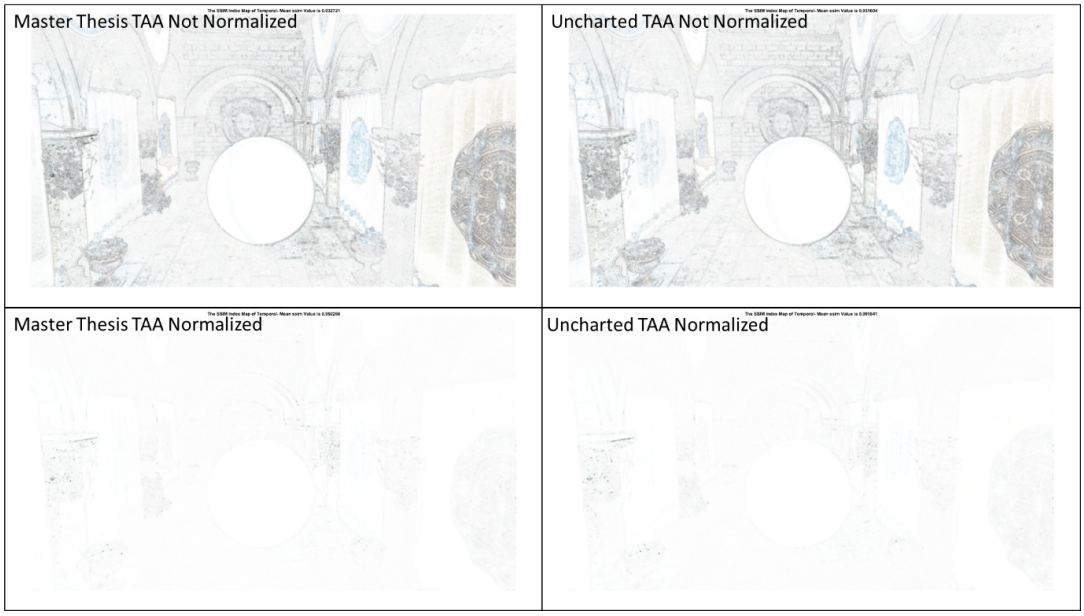

Figura 5.2: SSIM Maps comparison.

5.2.2 Pipe

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

5.2.2.1 Regular

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

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

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

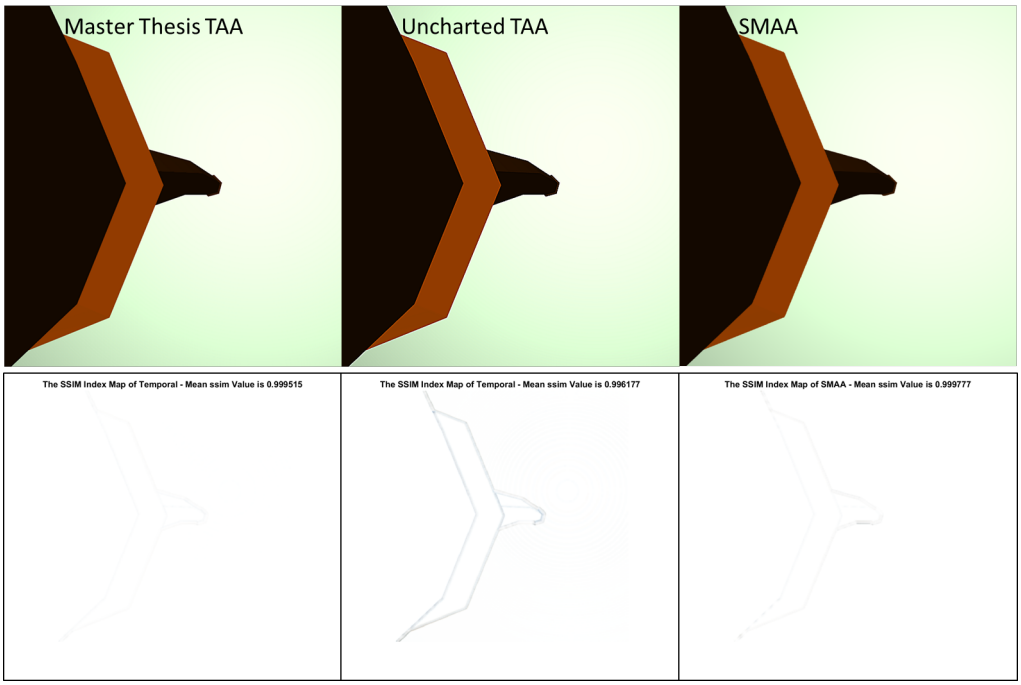

Figura 5.3: Pipe Regular comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

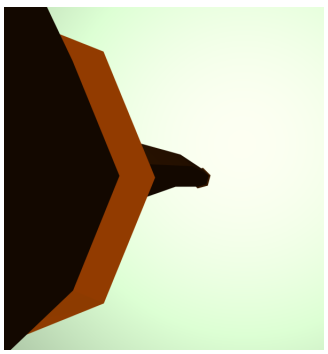

Figura 5.4: Pipe Regular Test ground truth.

5.2.2.2 With Camera Inclination

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

Finally, from Figure 5.5 and Table 5.6 we note that SMAA is not properly detecting the upper edge of the pipe, when zoomed we observe a small staircase forming around the edge. This is due to the fact that the camera was set up with a skewed inclination which pushed to the limit the edge detection techniques used in SMAA.

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

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

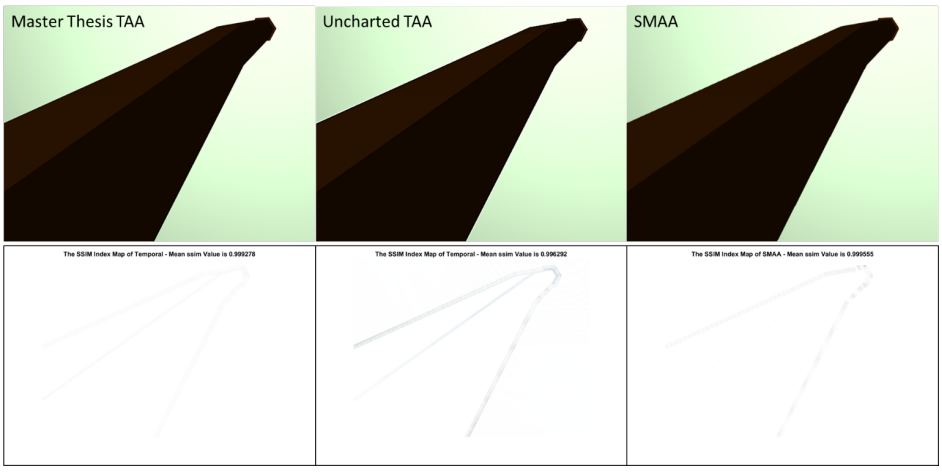

Figura 5.5: Pipe with Camera Inclination comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

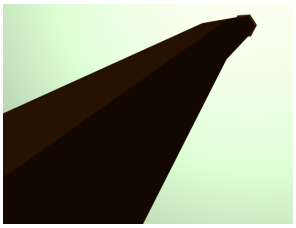

Figura 5.6: Pipe with Camera Inclination Test ground truth.

5.2.3 Window with Blinds

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

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

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

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

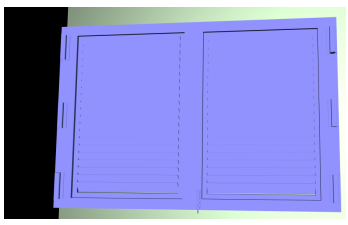

Figura 5.7: Window with Blinds ground truth.

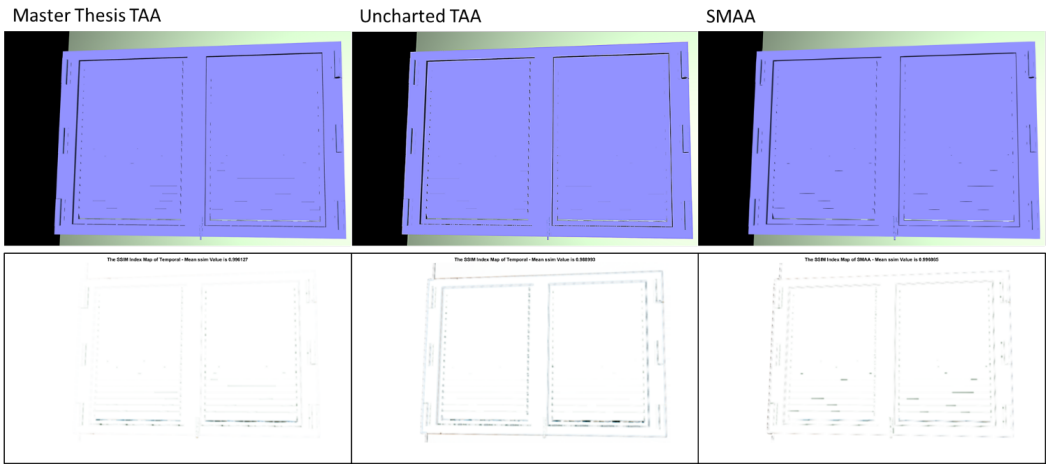

Figura 5.8: Window with Blinds comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

5.2.4 Arched Window

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

Cuadro 5.5: Numerical results of the Arched Window Test.

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

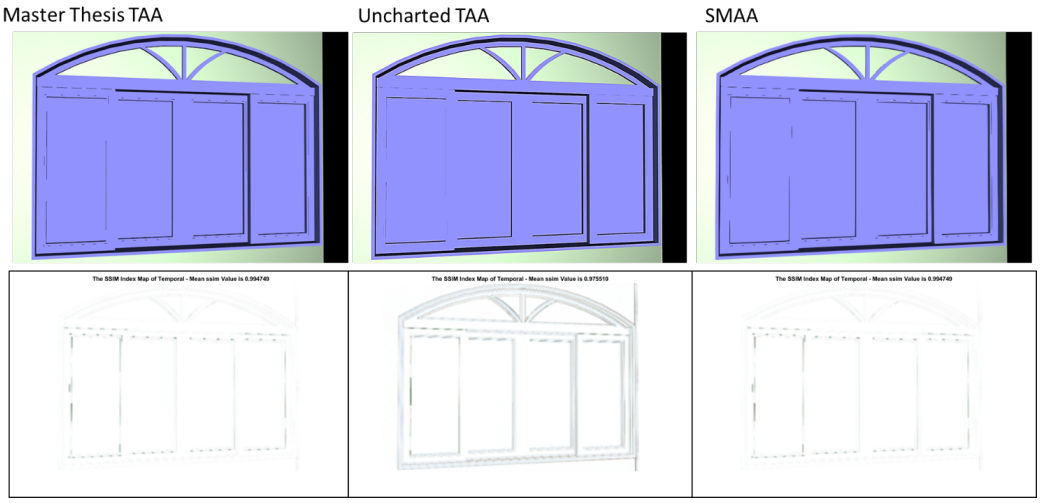

Figura 5.9: Arched Window comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

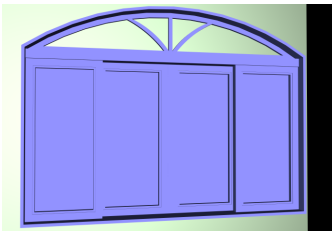

Figura 5.10: Arched Window ground truth.

5.2.5 Sponza Atrium

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

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

Cuadro 5.6: Numerical results of the Sponza Atrium Test.

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

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

Figura 5.12: Sponza Atrium ground truth.

5.2.6 Sponza Atrium Flowers

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

From Figures 5.13 and 5.14, we observe the Uncharted TAA has many problems handling this type of model, especially if we look at its SSIM map; we notice that SMAA could not correct all the aliasing artifacts from the edges of the flowers, we see the edges of the

flowers appear on its SSIM map; and, finally, we observe that the Master Thesis TAA corrected the most aliasing artifacts, some are still left as seen on its SSIM map on the gray areas. Furthermore, Table 5.7 confirm what we observe visually, as the Master TAA got the best scores and SMAA scored worse than average.

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

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

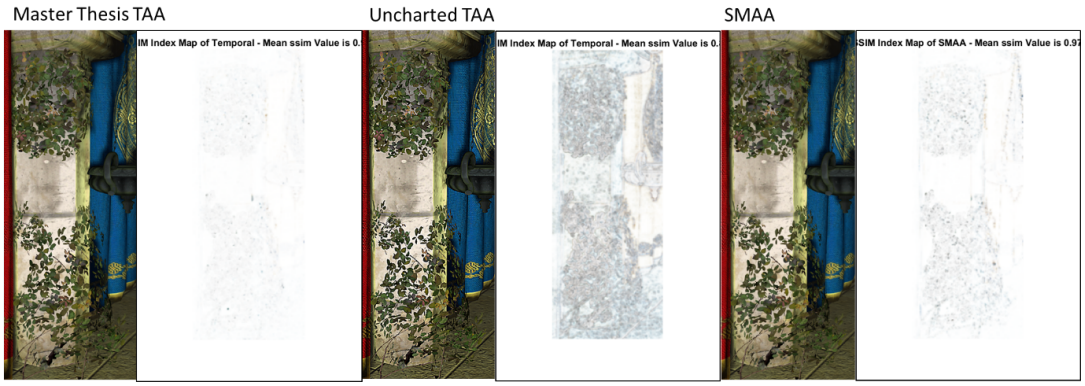

Figura 5.13: Sponza Atrium Flowers comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

Figura 5.14: Sponza Atrium Flowers ground truth.

5.2.7 Hard Edges

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

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

Cuadro 5.8: Numerical results of the Hard Edges Test.

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

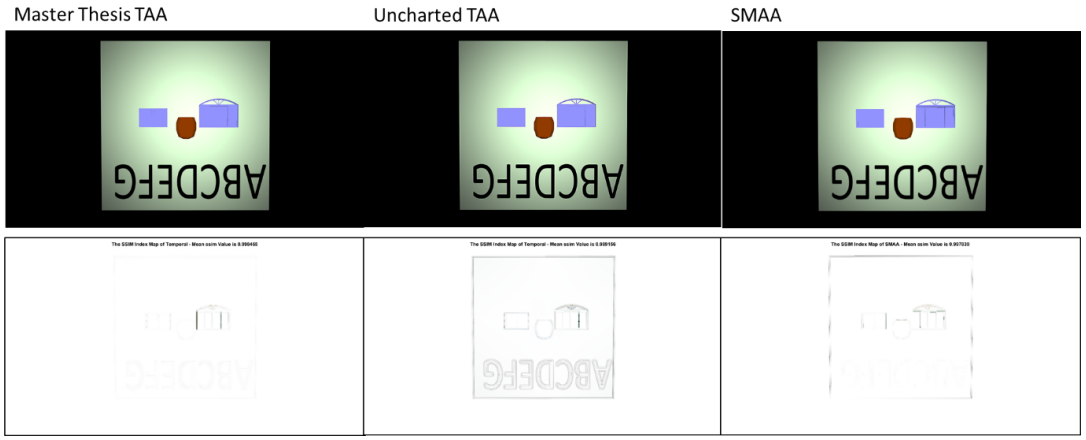

Figura 5.15: Hard Edges comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

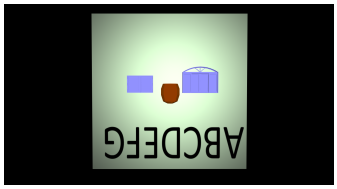

Figura 5.16: Hard Edges ground truth.

5.2.8 Sphere Ghosting

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

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

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

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

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

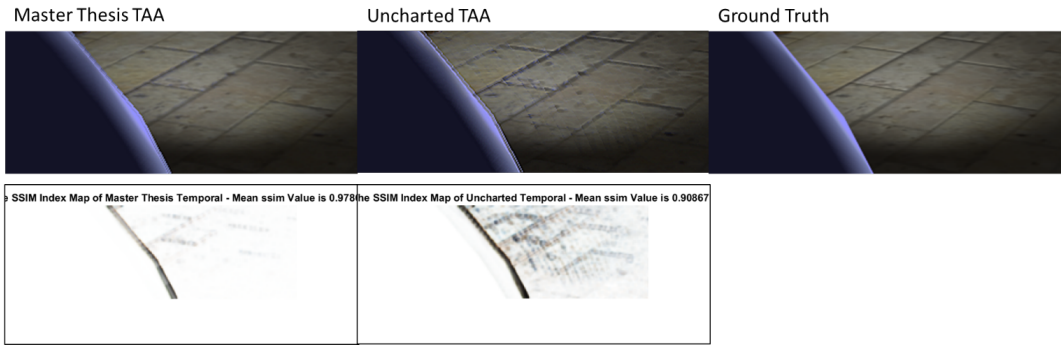

Figura 5.17: Example of Ghosting. Comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 19.

5.2.9 Hairball

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

5.2.9.1 Static Shadow

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

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

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

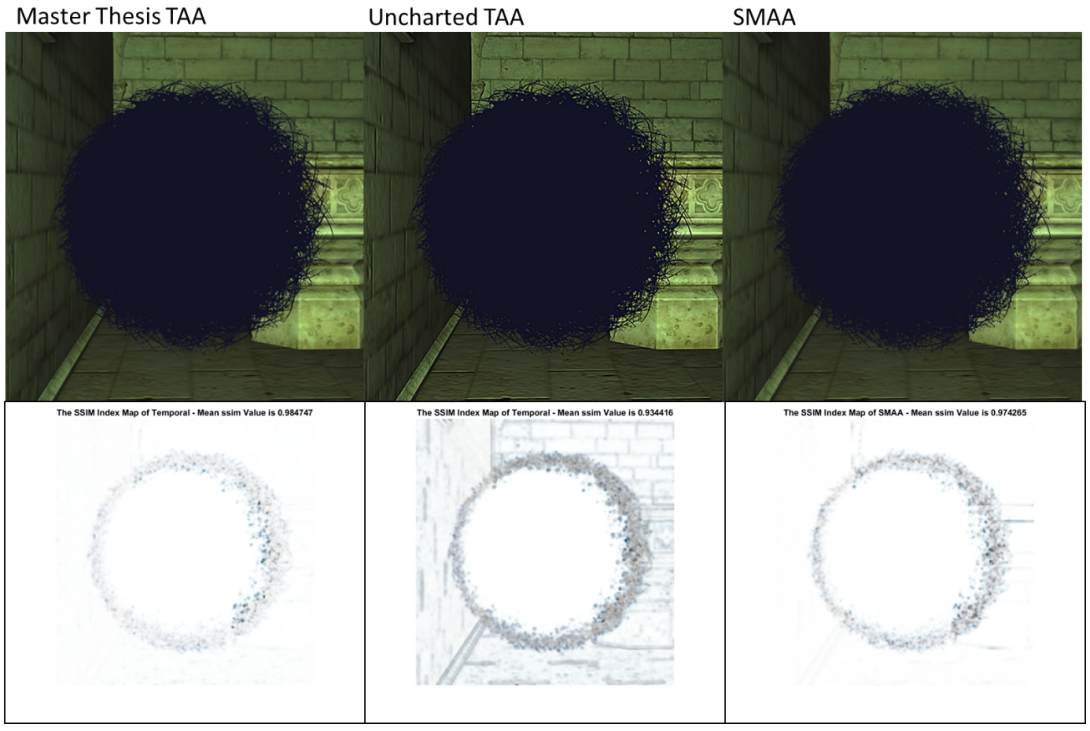

Figura 5.18: Hairball Static Shadow comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

Figura 5.19: Hairball Static Shadow ground truth.

5.2.9.2 Static Light

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

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

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

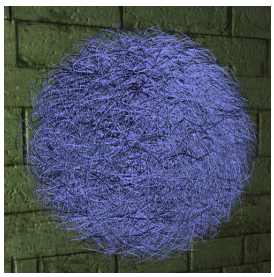

Figura 5.20: Hairball Static Lighted ground truth.

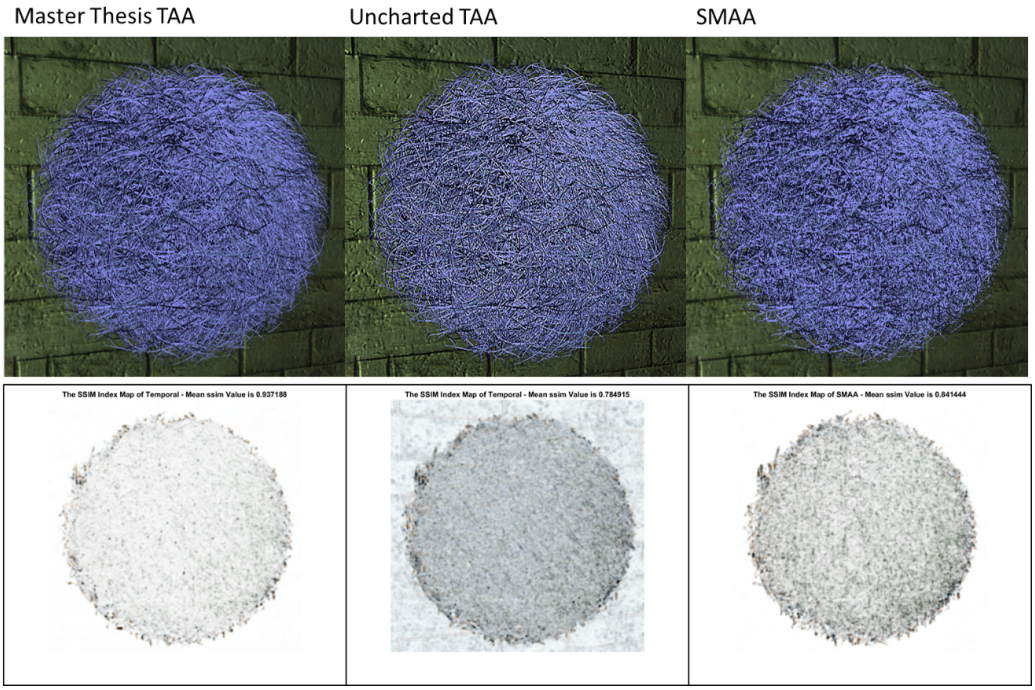

Figura 5.21: Hairball Static Lighted comparison between Master Thesis TAA, Uncharted TAA, and SMAA.

5.2.9.3 Ghosting Shadow

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

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

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

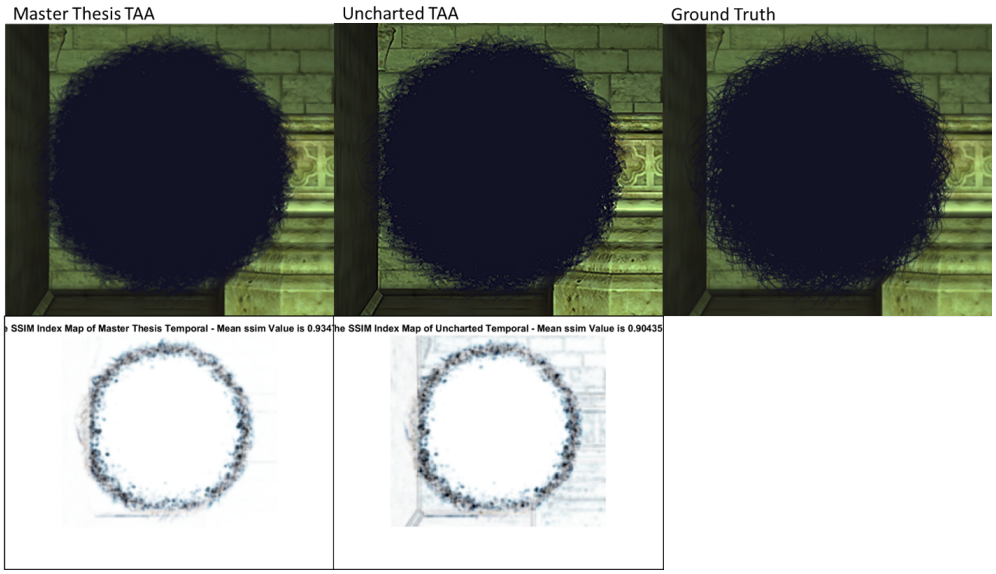

Figura 5.22: Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 90.

5.2.9.4 Ghosting Light

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

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

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

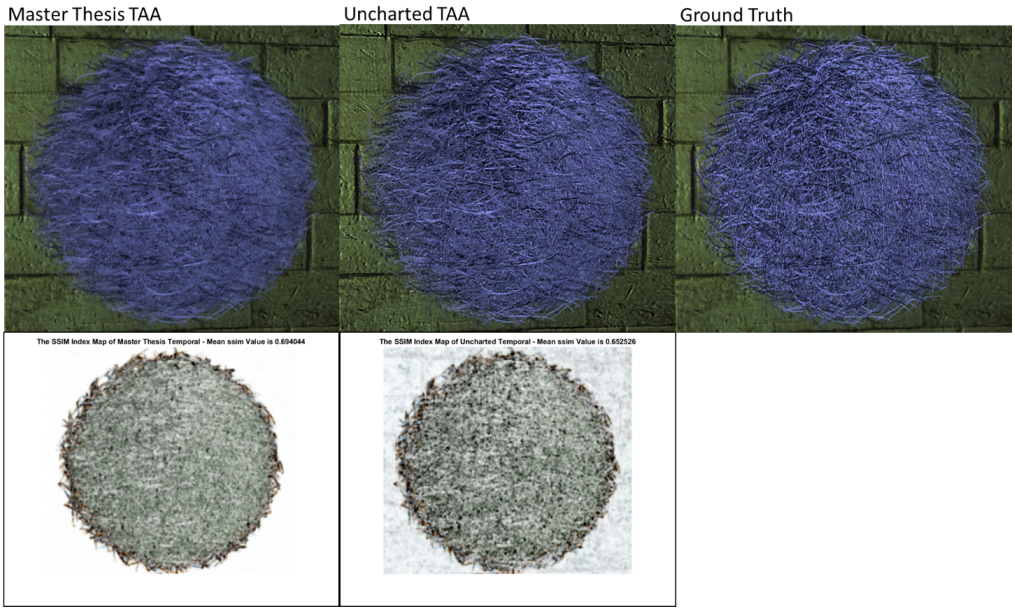

Figura 5.23: Ghosting comparison between Master Thesis TAA, Uncharted TAA and Ground Truth on Test Number 83.

5.2.10 Timing

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

The measured time the Master Thesis TAA technique took to run was between $0{,}5$ and $0{,}6\ ms$ on average. The Sobel pass took between $0{,}2\ ms$ and $0{,}3\ ms$, and the reprojection

was around $0{,}3\ ms$. The measured time the Uncharted TAA technique took was between $0{,}2\ ms$ and $0{,}3\ ms$; FXAA took between $0{,}1\ ms$ and $0{,}2\ ms$; and SMAA took $0{,}2\ ms$ and $0{,}3\ ms$.

5.3 Discussion

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

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

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

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

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

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

Figure 5.15).

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

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

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

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

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

Capítulo 6
Conclusions and Recommendations

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

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

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

Bibliografía

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

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

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

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

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

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

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

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

[9] J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez. SMAA: Enhanced subpixel morphological antialiasing. *EUROGRAPHICS 2012 / P. Cignoni, T. Ertl Volume 31 (2012), Number 2*, 2012.

[10] J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthuis, H. Yu, M. McGuire, T. Lottes, H. Malan, E. Persson, D. Andreev, and T. Sousa. Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses*, 2011.

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

[12] M. Doggett. EDAF80 Computer Graphics, September 2017.

[13] M. Doggett. EDAN35 High Performance Computer Graphics, November 2017.

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

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

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

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

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

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

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

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

Appendices

Apéndice A
GitHub Repository

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

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

Apéndice B

Camera Jittering Explanation

The $HaltonSequence(2,3)$ generates points in the $[0,1] \times [0,1]$ space. First, we need to transform it to the $[-1,1] \times [-1,1]$ space because we consider the pixel to be at the center, i.e. in OpenGL the first pixel is at $(0{,}5, 0{,}5)$, so we apply the transformation $T_1(x,y) = 2 * (x,y)(1,1)$.

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

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

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

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

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

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

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

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

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

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

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