

Máster en Computación Gráfica y Simulación

2018

Trabajo Final de Máster

Estudio sobre sistemas de anti-aliasing e
implementación de anti-aliasing temporal

Hugo Ferrando Seage

Tutor: Dr. Alberto Sánchez Campos

Resumen

Debido a la resolución limitada de las pantallas al rasterizar gráficos 3D se deforman ciertas líneas y curvas. Este fenómeno se llama aliasing. La demanda de gráficos cada vez más realistas ha propiciado la creación de diferentes técnicas y algoritmos para disimular estos defectos, sin tener que recurrir necesariamente a pantallas de mayor resolución.

El antialiasing se ha vuelto una técnica crucial para mejorar la calidad de imagen en el software con gráficos tridimensionales. Este campo de la computación gráfica lleva años desarrollándose, con nuevas técnicas publicadas continuamente.

Este proyecto tiene como objetivo el análisis de las distintas técnicas existentes en el estado del arte y la implementación de un algoritmo de temporal antialiasing, compatible con motores con deferred shading y comparable en calidad al multisampling, junto a otras propiedades.

Abstract

When rasterizing 3D graphics using computer screens, due to their limited resolution some lines and curves will become deformed. This phenomena is called aliasing. The ever increasing demand for more realistic graphics has driven the creation of new techniques and algorithms to hide these artifacts, without necessarily using higher resolution screens.

Antialiasing has become a crucial technique to advance image quality in graphics software. This field has been explored for years, with new approaches being developed continually.

This projects aims to analyze different antialiasing techniques developed and the implementation of a temporal antialiasing technique, compatible with deferred shading engines and with similar quality to mutlisampling, amongst other benefits.

Índice general

1. Introducción	11
1.1. ¿Qué es el aliasing?	11
1.2. ¿Qué es el antialiasing?	11
2. Planteamiento del problema	13
3. Objetivos	15
4. Estado del Arte	17
4.1. Aliasing en bordes de geometría	17
4.2. Aliasing de texturas	17
4.3. Specular Aliasing	17
4.4. Motion Aliasing	17
4.5. Filtrado	18
4.6. Supersampling Antialiasing	18
4.6.1. Ordered Grid Supersampling	19
4.6.2. Rotated Grid Supersampling	20
4.6.3. Quincunx Supersampling	20
4.7. Multisampling	22
4.8. Post Processing Antialiasing	22
4.8.1. Fast Approximate Anti-aliasing	22
4.8.2. Morphological Antialiasing	23
4.8.2.1. Enhanced Subpixel Morphological Antialiasing	23
4.8.3. Temporal Antialiasing	23
5. Desarrollo	25
5.1. History Buffer	25
5.2. Jitter	25
5.3. Motion Vectors	26
5.4. Combinación de texturas	26
5.5. Neighborhood Clamping	26

5.6. Sharpening kernel	27
6. Resultados	29
7. Conclusión	31

Glosario

OpenGL API estándar para el desarrollo de software con gráficos 2D o 3D con aceleración por hardware[10].

rasterizar Conversión de una imagen descrita con alguna primitiva (curvas, vectores, triángulos) a un conjunto de pixels.

Capítulo 1

Introducción

1.1. ¿Qué es el aliasing?

El antialiasing es la deformación de ciertos elementos gráficos al ser rasterizados y plasmados en una pantalla con una resolución finita[1]. Normalmente los bordes de la geometría que tengan ángulos que no se alineen perfectamente con los pixels presentaran bordes de sierra abruptos, que no es fiel a la escena como se vería de forma natural.

Este proyecto se centra en aliasing dentro del campo de la computación gráfica, pero afecta a otras disciplinas, como el procesamiento de señales (por ejemplo al digitalizar señales de audio analógicas). En general, al representar valores continuos en algún medio discreto siempre habrá algún defecto de este tipo[1].

Existen otros tipos de aliasing, como en el interior de texturas o al mover la camera rápidamente.

1.2. ¿Qué es el antialiasing?

El antialiasing es el conjunto de técnicas cuyo objetivo es el de disimular o eliminar estas imperfecciones.

Capítulo 2

Planteamiento del problema

El aliasing siempre ha sido un problema en el campo de la computación gráfica. Desde la década de los 70 se han usado técnicas como la interpolación bicubica para el renderizado de imagenes por ordenador con ray-tracing.

Durante la década de los 80 y buena parte de los 90 hubo un avance constante en términos de potencia, resolución y la capacidad de mostrar cada vez más colores (por ejemplo del paso de gráficos CGA a VGA). Aún así el rendimiento necesario para aplicar muchas de las técnicas de antialiasing solo se podían usar con workstations profesionales, como las IRIS de Silicon Graphics.

A mediados de los 90, con la llegada de las tarjetas aceleradores de gráficos 3D a ordenadores mas modestos las compañías como ATI, NVIDIA y 3DFX empezaron a integrar soluciones de antialiasing a sus productos y APIs.

A la vez que evolucionaba el hardware los gráficos cada vez requerían más potencia. Las técnicas como el supersampling reducen drásticamente el rendimiento, por lo que en muchos casos se empezaron a usar optimizaciones como el multisampling.

Con la llegada de los shaders programables y el deferred shading el multisampling empezó a no poder ser usado en muchos casos. Por ello se desarrollaron algoritmos como el MLAA, FXAA y SMAA que son compatibles tanto con el forward shading como el deferred shading y con un impacto muy bajo en el rendimiento.

Al contrario que el supersampling, el multisampling y el post-processing antialiasing no solucionan todos los tipos de aliasing que tiene una imagen. Para intentar aunar las ventajas del supersampling con el rendimiento y compatibilidad del post-processing antialiasing se han empezado a desarrollar técnicas de antialiasing temporal, como el TXAA.

Capítulo 3

Objetivos

Este proyecto tiene dos objetivos principales:

1. Analizar el estado del arte en el campo de la computación gráfica en relación al anti-aliasing. Explorar los problemas que presenta el aliasing en gráficos 2D y 3D, la evolución de las soluciones y los desarrollos en curso.
2. El segundo objetivo es el de integrar una solución de anti-aliasing temporal a un motor gráfico 3D existente desarrollado durante la asignatura de APIs tridimensionales del Máster en Computación Gráfica y Simulación cursado.

Capítulo 4

Estado del Arte

4.1. Aliasing en bordes de geometría

Cuando se piensa en los defectos del aliasing lo primero que viene a la cabeza es este tipo de defecto. Al rasterizar los polígonos, los bordes de la geometría no se ven con una línea suave, si no que tienen un único color y tienen una forma de escalón.

imagen

4.2. Aliasing de texturas

Dentro de las texturas también puede haber aliasing. Al igual que pasa en los bordes de la geometría, al cambiar de un color a otro rápidamente se pueden ver esos mismos defectos. Si una textura contiene alguna línea que no se alinee correctamente con la pantalla también se pueden apreciar estos defectos.

Posibles soluciones: Mip Maps, LoD

4.3. Specular Aliasing

La iluminación especular es aquella. Durante el cálculo de la iluminación especular que se refleja en los materiales como el metal o el plástico a veces

4.4. Motion Aliasing

A veces al mover la cámara también se pueden producir efectos de aliasing. El efecto no se puede apreciar en una imagen fija, pero al verlo el movimiento es un efecto que distrae. Recuerda al parpadeo de una luz, que una vez que la cámara permanece quieta desaparece.

Este tipo de aliasing se suele solucionar con técnicas de blurring, que recuerdan más al efecto del cine a 24FPS que a técnicas de antialiasing como las que se explican a continuación.

A veces también se conoce como crawling.

Conforme el hardware ha ido evolucionando durante los años, también lo han hecho las técnicas empleadas para simular el efecto de blurring de las cámaras con la intención de hacerlo más realista.

Explicar motion blur en cámaras

Las cámaras de video capturan la luz durante un breve periodo de tiempo para capturar cada instantánea usada en un video.

Una de las primeras técnicas era la de usar un buffer de acumulación, combinarlo con el frame n y ir actualizándolo con n frames. La desventaja de este efecto es que emborrona toda la escena por igual y puede resultar incluso más desagradable que el propio aliasing.

Con la llegada de los shaders se empezó a usar per pixel motion blur. Para esto hace falta calcular donde estaba cada pixel en el frame $n-1$ y simular algún efecto con esa información.

El per object motion blur es un refinamiento de la anterior, donde en vez de calcular donde estaba cada pixel en el frame anterior, haces el cálculo por cada polígono de la escena. Eso significa que es posible añadir más blurring a algún objeto y dejar el resto de la escena con mayor detalle.

4.5. Filtrado

En gráficos raster (bidimensionales) el anti-aliasing se realiza mediante filtros. Para entender el filtrado en imágenes es necesario pensar en pixels no como pequeños cuadrados en una pantalla, si no en muestras de una función (normalmente 3 muestras, una por cada canal RGB)[2]. Los filtros (o kernels) transforman las funciones para intentar suavizar los cambios bruscos de color.

En la figura 4.1 se muestra una comparación de las diferentes formas en las que cada filtro interpola valores de una imagen con un solo canal. Cada punto representa la muestra de la función que se visualizará en la pantalla.

4.6. Supersampling Antialiasing

El supersampling consiste en minimizar el aliasing usando imágenes con una resolución mayor a la usada para su visualización (antialiasing espacial). Cada pixel en la pantalla será representado por más de un pixel en la imagen. Para el color final normalmente se hace la media del valor de los colores de los múltiples pixels.

Figura 4.1: Comparación de interpolación usando diferentes filtros[8]

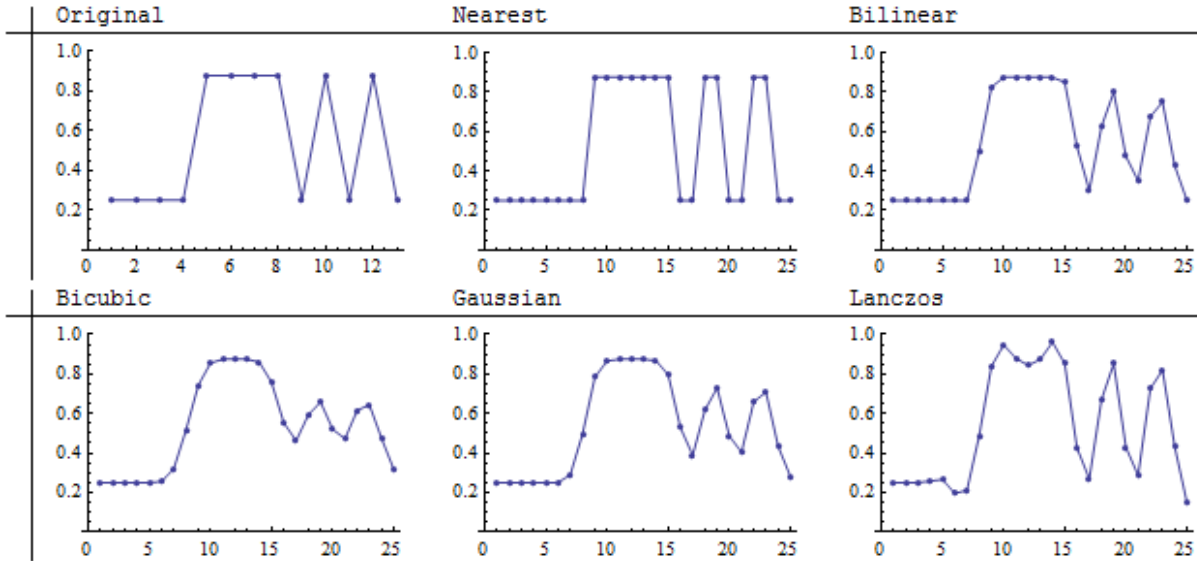
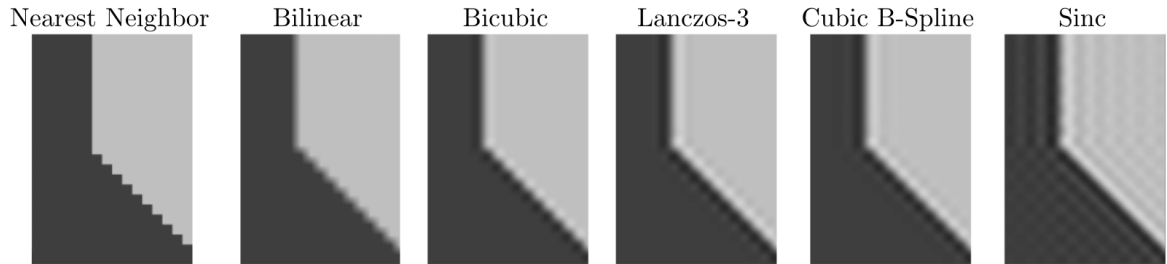


Figura 4.2: Comparación de resultados de diferentes filtros[7]



Estas técnicas consumen más ancho de banda y memoria, ya que los buffers tienen que guardar más información por pixel que la imagen original.

La calidad del supersampling viene dado por la cantidad de samples por pixel y la técnica para saber que pixels usar en la imagen con mayor resolución para cada pixel de la pantalla.

En comparación a otras técnicas de antialiasing esta corrige aliasing de texturas y especular, además de aliasing de geometría. Esto se debe a que los subsamples los coge de en toda la escena.

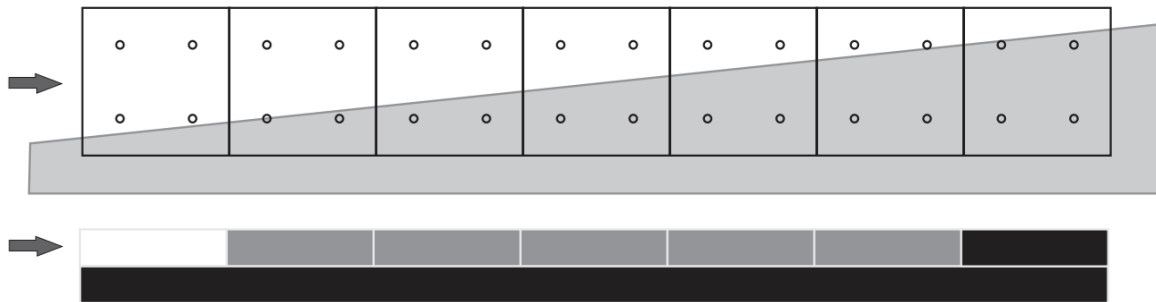
4.6.1. Ordered Grid Supersampling

En esta técnica de supersampling la distribución de los samples es uniforme. Debido a la naturaleza regular de este patrón, los subpixels se encuentran situados en forma de dos columnas y dos filas. Para líneas muy planas, tanto en horizontal como vertical el antialiasing no será muy efectivo, ya que muy rápidamente tocará dos subpixels en fila o columna y hasta el final no tocará los otros subpixels, como se puede ver en la figura 4.3.

Este efecto se puede reducir usando más samples, pero el impacto en el rendimiento será todavía mayor.

Desde 2014 Nvidia ha publicado una técnica llamada Dynamic Super Resolution[9], que consiste en renderizar software a mayor resolución y usar OGSS con un filtro de blur gaussiano. En 2015 AMD implementó una técnica similar llamada Virtual Super Resolution.

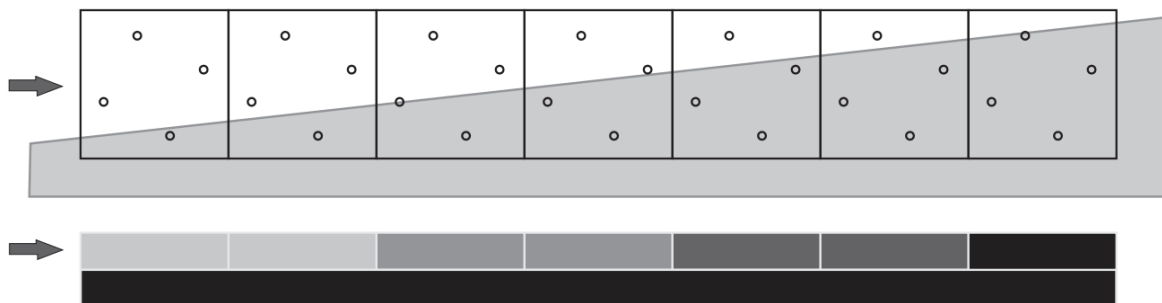
Figura 4.3: Ordered Grid Super-Sampling[3]



4.6.2. Rotated Grid Supersampling

Aquí la idea es cambiar el lugar de los subsamples para minimizar la cantidad de columnas y filas. La distribución sigue siendo uniforme, pero el problema del aliasing en líneas muy planas se soluciona, como se puede apreciar en la figura 4.4, sin tener que recurrir a aumentar de manera significativa el número de subsamples.

Figura 4.4: Rotated Grid Super-Sampling[3]



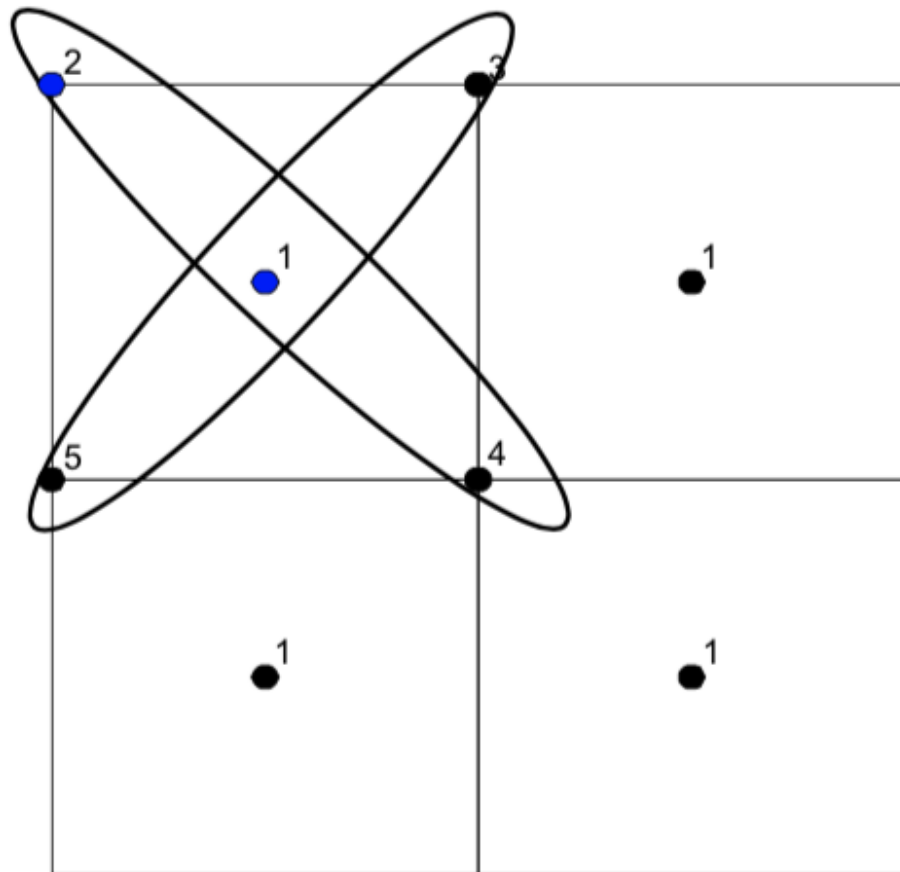
4.6.3. Quincunx Supersampling

Este patrón es interesante como técnica de supersampling ya que usa subsamples localizados en las esquinas de los pixels, lo que permite reutilizar varios subsamples entre pixels (figura 4.5). Esto permite una calidad similar a la de un antialiasing 4x pero con un

coste computacional de un supersampling 2x. Aún así al reutilizar subsamples produce resultados algo más borrosos que otros métodos.

Aunque este patrón se pueda usar para supersampling Nvidia lo implementó en hardware desde la GeForce 3[4] para acelerar el calculo de su multisampling. Sony al utilizar una GPU de Nvidia en la PS3 también disponía de esta tecnología en la consola.

Figura 4.5: Quincunx Pattern[4]



Quincunx Sample Pattern

4.7. Multisampling

Multisampling es una optimización del supersampling. Se sigue usando subpixels, pero en vez de resolver el color para cada subpixel el color se resuelve una única vez por pixel. Después de resolver el color se calcula el stencil buffer y depth buffer con subpixels, y dependiendo de la cantidad de subpixels que estén contenidos dentro del triangulo se usará un porcentaje del valor del color calculado. Esto reduce drásticamente el impacto en el rendimiento de la aplicación, ya que solo hay que acceder una vez por pixel a la textura, pero solo soluciona aliasing en los bordes de la geometría.

En el centro de un polígono todos los subpixels van a estar contenidos dentro, pero el color solo se calcula una vez (por ejemplo en el centro del pixel), por lo que se usará el 100 % del color, pero puede contener aliasing. En cambio, en los bordes, algún subpixel estará fuera de la geometría, por lo que se usará un porcentaje del color calculado, suavizando el borde.

Otra de las desventajas del multisampling anti-aliasing es que, tradicionalmente, no se puede usar en motores con deferred shading. Debido a que el calculo del color final de cada pixel se realiza en un pixel shader posterior al calculo del g-buffer, al hacer el resolve del mutisampling no esta lista toda la información necesaria. Desde OpenGL 3.2 y Direct3D 10.1 esto ha cambiado gracias al explicit multisampling[11]. La creación del g-buffer ahora se puede realizar usando texturas multisampled y al calcular la luz se puede acceder a los subsamples mediante la función texelfetch. Si se realiza de esta forma hay que sacrificar la creación automática de mipmaps y filtrado de texturas, aunque siempre se pueden implementar manualmente en shaders. Aún así muchos desarrolladores han optado por usar post-processing antialiasing al usar deferred shading.

4.8. Post Processing Antialiasing

Este tipo de antialiasing ha ganado popularidad durante los últimos años, debido en parte al uso del deferred shading en los últimos motores gráficos. Los algoritmos presentados a continuación se pueden considerar filtros de rasters, como los de la sección 4.5, pero están pensados para ser usados en motores gráficos. Además, en muchos casos, usan información que no esta presente en la textura final, como pueden ser texturas de profundidad o vectores de movimiento,

4.8.1. Fast Approximate Anti-aliasing

El FXAA[6] es un sistema de anti-aliasing de post procesado desarrollado por NVIDIA.

4.8.2. Morphological Antialiasing

mlaa

4.8.2.1. Enhanced Subpixel Morphological Antialiasing

smaa

4.8.3. Temporal Antialiasing

El temporal antialiasing es un tipo de antialiasing de post procesado, pero que amortiza el coste computacional usando múltiples samples en múltiples frames. Esto presenta unas complicaciones a la hora de escoger samples en frames anteriores en imágenes no estáticas.

Capítulo 5

Desarrollo

Para este proyecto se ha integrado un sistema de anti-aliasing temporal en un motor de gráficos 3D que usa OpenGL como API gráfica.

5.1. History Buffer

Debido a la necesidad de combinar varias texturas de frames anteriores, para computar el anti-aliasing de un frame n es necesario tener acceso al frame $n-1$. Aún usando más de 2 subsamples solo es necesario guardar el último, siempre y cuando tenga el antialiasing ya incluido.

También se puede optar por guardar n frames para n subsamples y combinarlos en un pixel shader, pero esto ocupa mucha más memoria.

5.2. Jitter

En primer lugar es necesario elegir los subsamples a usar en cada frame. Recordemos que en vez de calcular varios subsamples en cada frame, el temporal antialiasing calcula un solo sample por frame. Cada frame cambiará el lugar del subsample a calcular. Gracias a que solo se calcula un subsample por frame el impacto en el rendimiento será mucho menor al del supersampling tradicional.

Para elegir los subsamples se pueden usar los grids vistos anteriormente en la sección sobre supersampling REF. Aún así en la práctica se suelen usar secuencias de Halton.

Las secuencias de Halton proporcionan una baja discrepancia, lo que significa que no se van crear clusters ni en espacio ni en tiempo. Esto proporciona una calidad de subsamples muy alta por lo que la calidad del antialiasing puede ser muy elevado. Siempre que esté bien implementado los resultados serán mejores que los del antialiasing por hardware implementado hasta ahora. CITAR

Una vez se ha calculado la secuencia de Halton en cada frame se alterará la matriz MVP para mover el sample al lugar del subsample que toque en ese frame. Este lugar cambiará cada vez que se genera un nuevo frame.

En este punto si viésemos la imagen seguiríamos notando el aliasing y la imagen parecería que estuviera vibrando ligeramente.

5.3. Motion Vectors

El segundo paso para la implementación del temporal anti-aliasing consiste en la creación de un velocity buffer. Este buffer contendrá una textura con dos canales de color (rojo y verde) donde cada pixel representa un vector.

Estos vectores representan cuanto se ha movido ese pixel en el espacio de la pantalla (screen space) de este frame (n) al frame anterior (n-1). Con este buffer es posible combinar el history buffer que contiene el frame anterior con el antialiasing con la textura del frame del que se quiere hacer el anti-aliasing.

Cada pixel viene dado por su posición en la pantalla x e y. Al combinar texturas se combinan un pixel xy con el pixel xy - rg del history buffer (PONER ECUACIÓN).

Para la creación del velocity buffer en una escena dinámica es imprescindible saber no solo el movimiento de cámara del frame anterior, si no como se han movido los objetos. Esto se complica si hay objetos translucidos como ventanas.

En este proyecto se ha optado por usar el algoritmo descrito en (GPU GEMS CH 27) para su implementación. Solo usan pixel shaders junto al history buffer, el depth buffer y las matrices VPn y VPn-1. La ventaja de este algoritmo es que no depende mucho de la implementación del motor. Nvidia incluye una librería que hace estos calculas en el SDK NvidiaWorks (usado para la implementación de TXAA) CITAR NVIDIAWORKS.

Otra posibilidad es computar la velocidad durante una segunda pasada del vertex shader de cada polígono, pero dependiendo de su implementación puede resultar algo más lenta. Aún así debería resultar en vectores más precisos y en algunas ocasiones no queda más remedio que implementarlo así.

5.4. Combinación de texturas

5.5. Neighborhood Clamping

Los motion vectors a veces no son suficientemente precisos como para poder usarlos por si solos. Si un objeto ocluye a otro durante un cambio de frame el pixel resultante tendrá un color incorrecto, ya que se combinaran texturas de dos objetos diferentes.

Para solucionar esto se puede usar neighborhood clamping. Cada pixel se compara con el valor de los pixels adyacentes. Si el cambio de color es brusco y supera el umbral configurado se descarta la información del history buffer para ese pixel y se queda sin antialiasing hasta el siguiente frame.

EL GHOSTING TAMBIEN E PUEDE DISIMULAR USANDO BLURRING

5.6. Sharpening kernel

Debido a la naturaleza del temporal antialiasing la imagen final puede resultar demasiado suave. Los bordes quizás resulten algo difusos. Este efecto puede no ser del agrado del usuario. Una posible solución es usar un kernel de sharpening que resalta los bordes de la geometría.

Capítulo 6

Resultados

A continuación se encuentran algunos resultados.

Figura 6.1: Anti-aliasing off



Figura 6.2: Temporal anti-aliasing con 8 samples

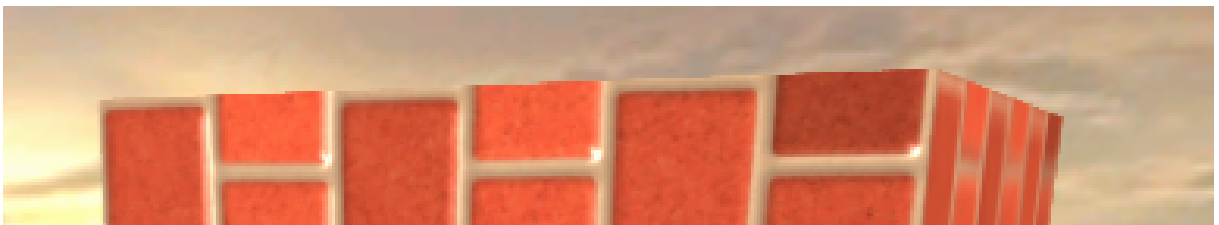
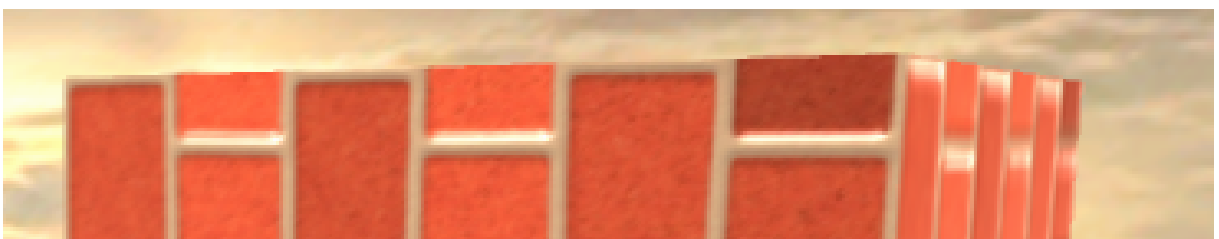


Figura 6.3: Temporal anti-aliasing con 16 samples



Capítulo 7

Conclusión

Bibliografía

- [1] D. P. Mitchell y A. N. Netravali, «Reconstruction Filters in Computer-graphics», *SIGGRAPH Comput. Graph.*, vol. 22, n.º 4, págs. 221-228, jun. de 1988, ISSN: 0097-8930. DOI: 10.1145/378456.378514. dirección: <http://doi.acm.org/10.1145/378456.378514>.
- [2] A. R. Smith, «A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square! (And a Voxel is Not a Little Cube)», Technical Memo 6, Microsoft Research, inf. téc., 1995.
- [3] K. Beets y D. Barron, «Super-sampling Anti-aliasing Analyzed», 2000.
- [4] Nvidia, «HRAA: High-Resolution Antialiasing through Multisampling», inf. téc., 2001. dirección: https://www.nvidia.com/object/feature_hraa.html.
- [5] J. Sachs, «Image Resampling», inf. téc., 2001.
- [6] T. Lottes, «FXAA. NVIDIA White Paper», inf. téc., 2009.
- [7] P. Getreuer, «Linear Methods for Image Interpolation», vol. 1, sep. de 2011.
- [8] *What is Lanczos resampling useful for in a spatial context?*, 2011. dirección: <https://gis.stackexchange.com/questions/10931/what-is-lanczos-resampling-useful-for-in-a-spatial-context>.
- [9] Nvidia, «GeForce GTX 980 Whitepaper», inf. téc., 2014. dirección: https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF.
- [10] J. Kessenich, G. Sellers y D. Shreiner, *OpenGL® Programming Guide: The Official Guide to Learning OpenGL®, Version 4.5 with SPIR-V*, 9.^a ed. Addison-Wesley Professional, 2016, ISBN: 9780134495491.
- [11] K. Nvidia, «OpenGL ARB texture multisample», inf. téc., 2009-2014. dirección: https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_texture_multisample.txt.

Índice de figuras

4.1. Comparación de interpolación usando diferentes filtros[8]	19
4.2. Comparación de resultados de diferentes filtros[7]	19
4.3. Ordered Grid Super-Sampling[3]	20
4.4. Rotated Grid Super-Sampling[3]	20
4.5. Quincunx Pattern[4]	21
6.1. Anti-aliasing off	29
6.2. Temporal anti-aliasing con 8 samples	29
6.3. Temporal anti-aliasing con 16 samples	29

Índice de cuadros

