UNIVERSIDAD SIMÓN BOLÍVAR
Ingeniería de Computación

Rendering Volumétrico Acelerado basado en Mecanismos de Manejo de Texturas

Por
Ciro José Durán Ostos
Francisco Rafael Morillo Morales

Proyecto de Grado
Presentado ante la Ilustre Universidad Simón Bólívar
como Requisito Parcial para optar al Título de
Ingeniero de Computación

Sartenejas, Marzo de 2006

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

ACTA FINAL DEL PROYECTO DE GRADO

Rendering Volumétrico Acelerado basado en Mecanismos de Manejo de Texturas

Presentado por
Ciro José Durán Ostos
Francisco Rafael Morillo Morales

Este Proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

____
Prof. Dionisio Acosta

____
Prof. Víctor Theoktisto

____
Prof. Yudith Cardinale

Sartenejas, Marzo de 2006

Rendering Volumétrico Acelerado basado en Mecanismos de Manejo de Texturas

por

Ciro José Durán Ostos

Francisco Rafael Morillo Morales

RESUMEN

Rendering Volumétrico (VR por sus siglas en inglés) es una técnica para hacer proyecciones bidimensionales a partir de datos tridimensionales discretos. Las aplicaciones principales de esta técnica está en la visualización de datos médicos, geológicos o como medio para hacer representaciones de dinámica de fluidos.

Este proyecto de grado tiene por objetivo evaluar la manera que el hardware gráfico convencional puede ser utilizado para lograr visualizaciones de buena calidad y alto desempeño. Si se observa VR como un proceso físico, simula el comportamiento de la luz al atravesar un medio semitransparente que involucra una evaluación costosa en tiempo y espacio. Nuevos enfoques en este problema tratan de acelerar la simulación al tener trozos del proceso pre-calculados.

La meta de este trabajo fue la implementación y evaluación de diversas técnicas de VR directo (ray casting y basado en el uso de texturas), así como la implementación de los trabajos de Westermann & Sevenich y Engel, Kraus & Ertl. Estos últimos dos trabajos se enfocan en el uso de mecanismos de manejo de texturas y pre-cálculos de la integral de VR (pre-integración) para lograr resultados con alta calidad y rapidez que pueden competir junto a otras técnicas más refinadas de mayor complejidad.[WS01],[EKE01]

Para tener un punto de referencia en cuanto a desempeño y calidad se evaluaron las técnicas de VR directo implementadas usando técnicas avanzadas de OpenGL y Cg (*C para gráficos*). Finalmente se realizaron pruebas para comparar la calidad y desempeño de ambos tipos de técnicas y se contrastó contra los resultados publicados. Los resultados muestran que la visualización de datos usando VR sobre hardware gráfico convencional es viable en tiempo real.

Índice general

Introducción **1**

1. Marco Teórico **4**

1.1. Volume Rendering 4

1.1.1. Descripción del Proceso Físico 6

1.1.2. Métodos de Volume Rendering 7

1.1.2.1. Ray Casting 7

1.1.2.2. Basado en Texturas 8

1.1.2.3. Splatting 8

1.1.2.4. Shear Warp 8

1.1.3. Aplicaciones de Volume Rendering 9

1.1.4. Retos de Volume Rendering 10

1.2. Estado del Arte y Tecnología 11

1.2.1. Antecedentes 11

1.2.2. OpenGL 13

1.2.2.1. Pipeline Gráfico Actual 13

1.2.3. Cg 14

1.2.3.1. Justificación del Lenguaje 16

1.2.3.2. Entorno de Ejecución 17

1.2.3.3. Rendimiento 18

2. Volume Rendering Directo **19**

2.1. Introducción 19

2.2. Ray Casting 19

2.2.1. Descripción del Algoritmo 21

2.2.2. Detalles de Implementación 22

2.2.2.1. Estructura de los Datos de Entrada 22

2.2.2.2. Cálculo de las Normales 23

2.2.2.3. Iluminación 24

2.2.2.4. Interpolación Trilineal 26

2.2.2.5. Composición 27

2.2.2.6. Resultado Final 28

2.3. Volume Rendering Basado en Texturas 28

2.3.1. VR Basado en Texturas 2D 29

2.3.1.1. Algoritmo 30

2.3.1.2. Diagrama Esquemático 30

2.3.1.3. Detalles de Implementación 31

2.3.1.4. Calidad de las Imágenes Obtenidas 34

2.3.2. VR Basado en Texturas 3D 35

2.3.2.1. Algoritmo 37

2.3.2.2. Diagrama Esquemático 38

2.3.2.3. Detalles de Implementación 38

2.3.2.4. Calidad de las Imágenes Obtenidas 41

2.3.2.5. Múltiples Volúmenes 41

3. Técnicas de Pre-integración de Volume Rendering **44**

3.1. Introducción 44

3.2. Metodología 44

3.3. Técnica de Westermann & Sevenich 45
3.3.1. Descripcion del Algoritmo 47
3.3.2. Detalles de Implementación 48
3.4. Técnica de Engel, Kraus & Ertl 52
3.4.1. Descripción del Algoritmo 53
3.4.2. Detalles de Implementación 54
3.4.3. Resultados Obtenidos 55
4. Resultados **57**
4.1. Descripción del Equipo Usado 57
4.2. Parámetros a Medir 57
4.3. Metodología de las Pruebas 58
4.4. Descripción de los Datos de Entrada en las Pruebas 59
4.4.1. Densidad de Voxels 59
4.4.2. Dispersión de Voxels 59
4.5. VR Directo Usando Mecanismo de Texturas 60
4.5.1. Volumen con una Sola Clase 60
4.5.2. Múltiples Volúmenes 63
4.6. VR Directo Usando Técnicas de Preintegración 64
4.6.1. Westermann & Sevenich 64
4.6.2. Engel, Kraus & Ertl 65
4.7. Análisis de los Resultados 65
4.7.1. Técnicas Basadas en Texturas 65
4.7.2. Técnica de Westermann & Sevenich 68
4.7.3. Técnica de Engel, Kraus & Ertl 68
5. Conclusiones **70**
5.1. Trabajo Realizado 70

5.2. Resultados Obtenidos 72

5.3. Trabajo a futuro 73

Bibliografía **75**

A. Resultados de pruebas **78**

A.1. VR Directo basado en texturas 78

A.2. VR Directo Preintegrado - Westermann & Sevenich 82

A.3. VR Directo Preintegrado - Engel, Kraus & Ertl 83

B. Imágenes obtenidas **84**

B.1. Ray casting 84

B.2. Basado en manejo de texturas 85

B.2.1. Texturas bidimensionales 85

B.2.2. Texturas tridimensionales 86

B.3. Basado en técnicas de Preintegración 88

B.3.1. Westermann & Sevenich 88

B.3.2. Engel, Kraus & Ertl 90

Índice de tablas

1.1. Comparación técnica entre algunos CPU y GPU 12

1.2. Perfiles de ejecución de programas en GPU 15

2.1. Presentación de los datos de [MNI] 23

3.1. Criterios de la prueba de alpha en OpenGL 49

4.1. Densidad del volumen total de los datos de prueba 59

4.2. Dispersión sobre el volumen total de los datos de prueba 59

4.3. Volumen Grasa. Nivel de zoom 1.0. En movimiento. 61

4.4. Volumen Grasa. Nivel de zoom 2.0. En movimiento. Todas las cifras en fps. 61

4.5. Volumen Hueso. Nivel de zoom 1.0. En movimiento 62

4.6. Volumen Hueso. Nivel de zoom 2.0. En movimiento 62

4.7. Volumen Materia Gris. Nivel de zoom 1.0. En movimiento 62

4.8. Volumen Materia Gris. Nivel de zoom 2.0. En movimiento 63

4.9. Volumen Músculo y Piel. Nivel de zoom 1.0. En movimiento 63

4.10. Volumen Músculo y Piel. Nivel de zoom 2.0. En movimiento 63

4.11. Múltiples volúmenes 64

4.12. Westermann & Sevenich. Tiempo de Ray cast - Zoom 1.0 64

4.13. Westermann & Sevenich. Tiempo de Ray cast - Zoom 2.0 64

4.14. Engel, Kraus & Ertl - Resultados 65

Índice de figuras

1.1. Una representación de datos mediante VR 5
1.2. Transformación Shear [Lac94]. 9
1.3. Pipeline de visualización en OpenGL 14
1.4. Pipeline gráfico programable 16

2.1. Idea general del Ray Casting 21
2.2. Diagrama esquemático del proceso de Ray Casting 22
2.3. Representación interna de los datos de entrada 23
2.4. Vectores en el modelo de Iluminación del ray caster [MB05]. 25
2.5. Descripción espacial de la interpolación trilineal 26
2.6. Visualización de datos usando ray casting 28
2.7. Superposición de texturas 2D 29
2.8. Diagrama esquemático de la técnica de texturas 2D y texturas 3D con planos orientados al objeto. 31
2.9. Los 3 conjuntos de texturas usados en 2D son perpendiculares entre sí 33
2.10. Visualización utilizando texturas 2D 34
2.11. Diferencias visuales entre los planos orientados al volumen, y los orientados al plano de proyección. Las geometrías intermedias están delineadas en rojo. 36
2.12. Geometrías intermedias orientadas al plano de proyección [WS01]. 37

2.13. Diagrama esquemático de la técnica de texturas 3D con planos orientados al plano de proyección. 39
2.14. La relación de los datos de nuestra implementación con la textura donde se carga. . 40
2.15. Visualización utilizando texturas 3D 42
2.16. Visualización de múltiples volúmenes utilizando texturas 3D 43

3.1. Principio de saltos de espacios vacíos de [WS01]. 46
3.2. Visualización empleando la técnica de Westermann & Sevenich 51
3.3. Visualización empleando la técnica de Engel, Kraus & Ertl 56

A.1. Texturas 2D - Nivel de Zoom 1.0. Datos a partir de las tablas 4.3, 4.5, 4.7 y 4.9 . . . 78
A.2. Texturas 2D en movimiento - Nivel de Zoom 2.0. Datos a partir de las tablas 4.4, 4.6, 4.8 y 4.10 79
A.3. Texturas movimiento - Planos orientados al objeto - Nivel de Zoom 1.0. Datos a partir de las tablas 4.3, 4.5, 4.7 y 4.9 79
A.4. Texturas movimiento - Planos orientados al objeto - Nivel de Zoom 2.0. Datos a partir de las tablas 4.4, 4.6, 4.8 y 4.10 80
A.5. Texturas movimiento - Planos orientados al observador - Nivel de Zoom 1.0. Datos a partir de las tablas 4.3, 4.5, 4.7 y 4.9 80
A.6. Texturas movimiento - Planos orientados al observador - Nivel de Zoom 2.0. Datos a partir de las tablas 4.4, 4.6, 4.8 y 4.10 81
A.7. Texturas 3D Multiples Volúmenes. Datos a partir de la tabla 4.11 81
A.8. Westermann & Sevenich - Nivel de Zoom 1.0. Datos a partir de la tabla 4.12 82
A.9. Westermann & Sevenich - Nivel de Zoom 2.0. Datos a partir de la tabla 4.13 82
A.10. Engel, Kraus & Ertl. Datos a partir de la tabla 4.14 83

B.1. Ray casting - Visualización de tejido cerebral - Materia Blanca 84
B.2. Ray casting - Visualización de tejido craneal - Músculo y Piel 85

B.3. Efecto de popping utilizando texturas 2D 85
B.4. Basado en texturas - Múltiples Texturas - 4 Clases 86
B.5. Basado en texturas - Múltiples Texturas - 3 Clases 87
B.6. Westermann & Sevenich - Clase Músculo y Piel - zoom 2.0 88
B.7. Westermann & Sevenich - Clase Hueso - zoom 2.0 89
B.8. Engel, Kraus & Ertl - Clase Grasa - zoom 2.0 90
B.9. Engel, Kraus & Ertl - Clase Hueso - zoom 2.0 91

Glosario de Términos

A

Aliasing o Alias: Fenómeno que ocurre al muestrear una señal a una frecuencia menor a la frecuencia de Nyquist, por lo que las frecuencias altas y bajas se combinan erróneamente al reconstruir la señal original. Para evitarlo suele aplicarse un filtro de banda para limitar la señal de entrada.

Alpha: Componente de un fragmento el cual representa su nivel de opacidad.

Alpha Test: Es una de las pruebas de OpenGL que ocurren durante la rasterización. Si está activada durante el proceso de rendering, cumple la función de descartar o aceptar fragmentos en el framebuffer comparando su componente alpha con un valor referencial, según un criterio especificado.

API: *Application Programming Interface.* Describe como usar un conjunto de librerías o funciones para obtener provecho de ellas.

ARB: *Architecture Review Board.* La Junta de Revisión de la Arquitectura de OpenGL, es el conjunto de organizaciones y personas que definen los cambios en las especificaciones de OpenGL.

Artefacto: Defectos visuales en la representación de datos.

B

Banding: Un ejemplo específico de alias, ocurre cuando una imagen presenta "bandas" compuestas de diferentes degradados de un color en una sola que originalmente era de un color homogéneo.

Blending: Es la operación en la cual dos componentes de colores se reducen a un solo compo-

nente, por lo general utilizando una interpolación lineal entre ellos.

Buffer: Memoria utilizada para almacenar temporalmente información de entrada o salida.

Buffer de Cuadros: Espacio de memoria de las tarjetas gráficas encargado de almacenar los próximos cuadros a mostrar en la pantalla.

Buffer de Profundidad: Espacio de memoria de las tarjetas gráficas que almacena, para cada coordenada de una imagen, la distancia en profundidad (eje z) del fragmento más cercano perpendicular al plano de proyección.

Byte: Unidad de capacidad de almacenamiento, equivalente a 8 bits.

C

Campo Escalar: Un campo escalar refiere un valor escalar para cada punto de un espacio dado, matemáticamente se define como una función de $\mathbb{R}^n \rightarrow \mathbb{R}$.

Clasificación: Procedimiento por medio del cual, dado un volumen de datos y una función de transferencia, se diferencian las clases que pertenecen al volumen. En base a la clase y la cantidad presente se le otorgan valores de color y opacidad respectivamente.

Coordenadas del Objeto: Para los objetos presentes en una escena se toma en cuenta un sistema local de coordenadas que sea conveniente para su manipulación. Se conocen mejor como *model coordinates*.

Coordenadas del Mundo: Definen un espacio tridimensional cuyas coordenadas corresponden a toda las escena.

Coordenadas normalizadas del dispositivo: Definen el sistema de coordenadas de la imagen final antes de dibujarla en la pantalla.

CPU: Unidad de procesamiento central.

CT: *Computer Tomography.* Tomografía Axial Compurizada.

Cuadro: Equivale a una imagen mostrada por un dispositivo de salida visual continuo, como un monitor.

D

Depth Buffer: *Véase buffer de profundidad.*

Depth Test: Es una de las pruebas de OpenGL que ocurren durante la rasterización. Si está activada durante el proceso de rendering cumple la función de descartar o aceptar fragmentos en el framebuffer comparando su profundidad, es decir, su valor en el eje Z de las coordenadas normalizadas del dispositivo, con su valor correspondiente en el buffer de profundidad, según un criterio especificado.

Display Lists: Son estructuras definidas en OpenGL las cuales almacenan y compilan operaciones de OpenGL para su posterior llamado, ya que son guardadas en la memoria de la tarjeta gráfica la ejecución de estas operaciones se hace de manera más rápida.

F

Fill rate: Velocidad promedio a la cual una tarjeta gráfica puede acceder y dibujar, en un framebuffer, los texels correspondientes a una textura bidimensional.

Fotograma: *Véase cuadro.*

Fragmento: Es el resultado de la rasterización de una primitiva de OpenGL. Suele tener correspondencia uno a uno con un pixel y posee características como color, alpha, profundidad (en coordenadas normalizadas de dispositivo) y coordenadas de textura, de ser posible.

Fragment Shader: Un programa ejecutado en el procesador de la tarjeta gráfica justo después de la rasterización. En particular, define las propiedades finales de color para un fragmento.

Frame: *Véase cuadro.*

Framebuffer: *Véase buffer de cuadros.*

Frecuencia de Nyquist: Es la frecuencia más alta que puede ser reproducida con precisión cuando una señal es discretizada a una frecuencia de muestreo dada.

Función de Transferencia: Es la representación matemática de la relación que existe entre la entrada y salida de un sistema linear no variante en el tiempo.

Función fija de OpenGL: Es el conjunto de transformaciones de vértices y fragmentos especificadas por OpenGL. Se aplica sobre los datos ingresados al pipeline cuando no existe un fragment shader cargado.

G

GLUT: *GL Utility Toolkit.*

GPU: Unidad de Procesamiento Gráfico, del inglés *Graphics Processing Unit.*

Gradiente: Es la aproximación del valor del vector normal de una superficie.

H

Halfway Vector: Se define como el promedio de los vectores de la posición de la luz y del observador. Puede expresarse a través de la siguiente ecuación:

$$\vec{H} = \frac{\vec{L} + \vec{V}}{2}$$

I

Isosuperficies: Se definen como la representación en un campo escalar discreto de valores, de aquellos puntos que tienen en común su valor.

M

MiB: *Mebibyte*, medida de almacenamiento, equivalente a 1,048,576 bytes (1024^2, 2^{20}).

Modelview: Es la matriz que almacena las transformaciones geométricas – rotaciones, escalamientos y traslaciones – que se hacen en una escena. En ella están contenidos las transformaciones necesarias para cambiar las coordenadas del objeto a las coordenadas del ojo.

MRI: *Imagen de Resonancia Magnética*, es una técnica de extracción de imágenes de organismos vivos y su fundamento funcional reside en la excitación de los núcleos de hidrógeno en las moléculas de agua, cambiando la orientación de su campo magnético. Al hacerlo, las moléculas son detectadas

obtieniéndose el valor del campo para su posterior procesamiento y generación de las imágenes correspondientes.

N

NDC: *Normalized Device Coordinates, véase Coordenadas normalizadas del dispositivo.*

O

Object-oriented planes: Son aquéllos que se generan cuando se desea hacer blending de planos de un volumen y se toman las tajadas en el orden del objeto. En un cubo de datos se suelen organizar los planos de 3 maneras una pila para cada eje ortogonal X, Y y Z.

P

Packed Array: Estructura de datos de tipo vector o matriz optimizada para el uso con ciertos procesadores. En este tipo de datos, el acceso a los elementos del vector es mucho más eficiente.

Padding: Consiste en agregar uno o más bits de datos a un bloque de datos para llegar a un tamaño específico.

PET: Tomografía de emisión de positrones. Es una técnica de medicina nuclear que genera una imagen tridimensional de tejidos, usando como principio, la detección de los fotones de rayos gamma que resultan de la colisión de los positrones contra los electrones.

Pipeline: Es la secuencia de etapas de procesamiento de datos hasta alcanzar un resultado.

Pixel: Elemento de una imagen.

Popping: Un ejemplo específico de aliasing. Se puede observar cambios repentinos en la imagen durante el cambio de algún parámetro de visualización, como la rotación de un objeto.

R

Radiancia: Describe la densidad de poder transmitido por los fotones en un punto particular en una dirección dada.

Rasterización: Es el proceso durante el cual imágenes descritas por primitivas tridimensionales o bidimensionales se convierten en fragmentos que son procesados por la tarjeta gráfica.

Rayos X: Es una técnica de generación de imágenes médicas que basa su principio en la emisión de radiación electromagnética en el rango de 10 a $0,1$ nanómetros.

Render: El que realiza el rendering.

Rendering: Rendición visual de un modelo a partir de un conjunto de datos de entrada.

Rendering Volumétrico: Proceso a través del cual, se puede hacer la visualización de datos provenientes de un campo escalar discreto sin necesidad de reconstruir la geometría del objeto en estudio.

RGB: Es un modelo de representación del color, se basa en la composición de los colores rojo, verde y azul.

S

Shading: Proceso de cálculo de iluminación de un objeto.

SIMD: Es un conjunto de operaciones para manipular grandes cantidades de información en paralelo y basa su principio en el uso de una sola instrucción para manipular multiples datos (del inglés *Single Instruction, Multiple Data*).

SSE: Extensiones de flujo SIMD, del inglés *Streaming SIMD Extensions.*

Surface Rendering: Consiste en la reconstrucción y visualización de superficies de un volumen de datos.

Swizzling: Capacidad de reorganizar los elementos de un *packed array* para crear uno nuevo, sin costos extras en el desempeño.

T

Texel: Elemento de textura.

Textura: Corresponde a imágenes de 1, 2 ó 3 dimensiones que pueden ser mapeadas a las primitivas que conforman una escena.

Tupla: Agrupación de elementos que comparten características similares.

V

Vector Normal: Es el vector que es perpendicular al plano tangente de una superficie sobre un punto.

Vertex Shader: Programa que se ejecuta durante el pipeline del GPU antes de la generación de las primitivas geométricas. Su principal función es definir las propiedades finales que tendrá un vértice introducido al pipeline, tales como posición, coordenadas de texturas, vector normal o color primario.

Vértice: Es un punto en un espacio tridimensional.

View-aligned planes: Son aquéllos que se generan cuando se desea hacer blending de planos de un volumen y se toman las tajadas en el orden de la imagen, es decir, se toman planos del volumen paralelos al plano de proyección.

Viewport: Es la matriz encargada de transformar las coordenadas normalizadas del dispositivo a coordenadas de la pantalla.

Volume Rendering: *Véase Rendering Volumétrico.*

Voxel: Elemento de volumen.

Z

Zoom: Nivel de acercamiento a un objeto en el espacio.

Introducción

Volume Rendering es el proceso mediante el cual es posible hacer una representación visual bidimensional de datos discretos que pertenecen a un espacio tridimensional. Cada uno de estos datos representa información escalar o vectorial de un fenómeno, proceso u objeto que se quiere visualizar. Por ejemplo, densidad, presión, carga eléctrica, o cualquier otra propiedad mensurable.

La carga computacional de volume rendering depende directamente del tamaño del volumen de datos de entrada tanto en espacio como en tiempo. Un volumen de datos puede llegar a ocupar desde unos pocos megabytes – como volúmenes de datos médicos – hasta unos cuantos gigabytes – volúmenes para la exploración de datos geológicos –. Muchas técnicas se han desarrollado para disminuir el problema del espacio y el procesamiento, con enfoques basados en el cambio del dominio de los datos, representación basada en datos comprimidos, cómputo paralelo, operaciones que reducen elementos de un volumen en una etapa previa al procesamiento y el uso de memoria de texturas para procesamiento de los datos.

Hasta hace pocos años, el poder de cómputo de los ordenadores de consumo masivo no era suficiente para hacer en un tiempo razonable volume rendering. Con la reducción en los costos de producción y avances en el desarrollo de nuevas tecnologías ya es posible lograr este cometido en tiempos muy razonables y con un alto nivel de calidad.

Es común ver en el mercado, tarjetas que contienen un chip gráfico que cuenta con capacidades aceleradoras avanzadas y una elevada capacidad de almacenamiento en memoria de texturas. Uno de los enfoques mencionados anteriormente se centra en el uso dedicado de la tarjeta gráfica para

hacer el procesamiento de los datos y en la capacidad de memoria de texturas que tienen éstas para almacenar los volúmenes.

El empleo de texturas describe una manera de dibujar capas de un volumen y lograr una representación bastante aproximada a la original. Con esta técnica, se trazan una serie de planos sobre el volumen – esto se conoce como generación de geometrías intermedias – los cuales se representan como una textura en la tarjeta gráfica, para luego dibujarlas en la pantalla en orden y con una función de composición reconstruir el volumen.

En este proyecto de grado se exploraron algunas técnicas que hacen uso de las capacidades de manejo de texturas de las tarjetas gráficas actuales, usando una serie de datos preclasificados y haciendo énfasis en la manera de hacer más eficiente el proceso por medio de métodos de pre-integración de los datos de entrada. Para ello se usaron funcionalidades de alto nivel en OpenGL y Cg (*C para gráficos*).

Los capítulos que componen este trabajo de grado están organizados de la siguiente manera:

En el capítulo 1 se tiene un breve resumen de las ideas principales de volume rendering, descripción general, los procesos asociados y una breve explicación de la base física. Además, se presenta un breve resumen del estado del arte en la tecnología disponible en la actualidad, las posibilidades que ofrece OpenGL y el lenguaje de alto nivel Cg como enlace a la programabilidad de los procesadores gráficos actuales.

El capítulo 2 describe detalladamente dos técnicas de volume rendering directo. Una de ellas es la aproximación en software del proceso de *ray casting* especificando las etapas del proceso y detalles de implementación. La otra explica el proceso de VR por medio del uso de texturas en donde se señalan las optimizaciones sobre el modelo original usando Cg y los detalles técnicos del método.

Para el capítulo 3 se muestran dos técnicas de volume rendering acelerado usando técnicas de pre-integración, cómo funcionan, qué aceleran, sus implementaciones y descripción del pipeline.

Se concluye la investigación con el capítulo 4 donde se indican los resultados obtenidos en las técnicas expuestas y sus correspondientes análisis, pros y contras, algunas ideas, así como propuestas

para trabajos futuros.

Finalmente, en el capítulo 5, se dan las conclusiones y recomendaciones pertinentes.

Capítulo 1

Marco Teórico

1.1. Volume Rendering

Volume Rendering (en lo sucesivo, VR) se define como el proceso mediante el cual datos escalares provenientes de un espacio tridimensional son procesados para mostrar sobre una proyección bidimensional una posible representación gráfica de éstos de acuerdo a Levoy [Lev88], Lacroute [Lac94] y McReynolds & Blythe [MB05]. Mediante VR es posible hacer esta visualización sin la necesidad de construir modelos geométricos intermedios. Las bases que constituyen el proceso de visualización son los modelos físicos de propagación de la luz sobre materiales que tienen índices de color y opacidad variables.

Los datos usados en VR provienen de un espacio tridimensional discreto, dividido en *voxels*. La idea general consiste en tomar cada una de las muestras y con esto, construir un volumen sobre el cual se hará la visualización. Por el hecho de ser un volumen de datos, la cantidad de procesamiento necesaria para representar estos datos aumenta en la medida en que las dimensiones de este volumen lo hacen.

La representación de las primitivas tridimensionales también es un proceso de proyección de datos sobre una superficie. La diferencia de complejidad reside en que las primitivas componen

solamente superficies con valores de color y opacidad bien definidos que por dentro están vacías. En VR no importa únicamente la parte externa del volumen, por el contrario, es necesario conocer los datos que componen el interior del volumen para dar el resultado.

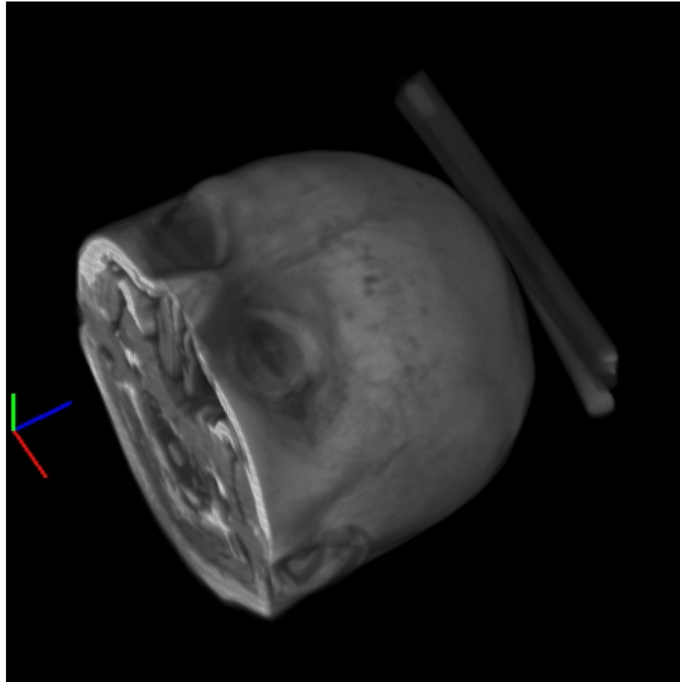

Figura 1.1: Una representación de datos mediante VR

A partir de lo anterior, se hace indispensable el estudio de métodos que permitan reducir la carga de procesamiento asociada a volúmenes de grandes dimensiones, sin sacrificar aspectos como la velocidad del proceso y la calidad de los resultados. VR se mantiene como área activa en el campo de la visualización científica y continuamente surgen avances para mejorar su velocidad y calidad.

Enfoques que toman en cuenta el cambio de dominio de los datos de entrada, la compresión de los datos, la reducción de la cantidad de muestras de los volúmenes y el uso de memoria de texturas para hacer la representacion son algunas de las propuestas en los últimos años y de todas las mencionadas, la del uso de memoria de texturas es la que se estudia en los próximos capítulos.

En la figura 1.1 se puede apreciar una visualización de datos provenientes de una resonancia

magnética de una cabeza humana. La cantidad de muestras del volumen de datos es de $181 \times 217 \times 181 = 7{,}109{,}137$ voxels.

1.1.1. Descripción del Proceso Físico

El proceso simula el comportamiento de la luz al atravesar un medio semi-transparente y mientras lo hace, pueden ocurrir varios fenómenos:

- La luz es absorbida por los elementos del volumen.
- La luz se dispersa en los elementos del volumen.
- La luz se emite desde los elementos del volumen.
- Una mezcla entre los factores anteriores, como puede ser el caso de la fluorescencia[1] o la fosforescencia[2].

Aunque la idea general de VR es simular este proceso, no es tan sencillo en la práctica, pues si se tomaran en consideración todos estos factores, la cantidad de cálculo involucrado sería desmesurada. Numéricamente consistiría en evaluar por completo la simulación del proceso físico, estando presente la simulación del medio circundante, reflexión, refracción entre otros. Una reducción del modelo óptico general es más que suficiente para hacer una visualización de los datos de manera apropiada.

En el modelo simplificado de VR solamente se estudia cómo, en el proceso de transporte, la luz es absorbida y emitida por los elementos del volumen. Se descarta por entero el proceso de dispersión de la luz de los volúmenes a visualizar. Lo anterior puede ser resumido en la siguiente ecuación:

$$L(x) = \int_{x}^{x_B} e^{-\int_{x}^{x'} \phi_t(x'')dx''} \epsilon(x')dx' \tag{1.1}$$

[1] Ocurre cuando una molécula absorbe un fotón de alta energía y luego emite un fotón de baja energía con una mayor longitud de onda.

[2] Es un proceso muy similar a la fluorescencia, la diferencia reside en la cantidad de tiempo que hay entre la absorción y emisión de los fotones, este tiempo puede ser muy prolongado, pues es un proceso lento.

donde $L(x)$ es la radiancia calculada en términos de una variable unidimensional de posición llamada x, y se toma en cuenta la interacción de los coeficientes de extinción (opacidades) $e^{-\int_x^{x'} \phi_t(x'')dx''}$ y de los colores $\epsilon(x')$ para cada muestra del volumen. Actualmente no es posible evaluar numéricamente por completo la integral en un computador por el hecho de requerir sumas infinitas. La solución consiste en llevar esta ecuación a un modelo discreto que pueda ser evaluado de manera simple.

Usando la regla del rectángulo la ecuación anterior, según Lacroute [Lac94], se reduce a:

$$L(x) = \sum_{i=0}^{n-1} e^{-\sum_{j=0}^{i-1} \phi_j \Delta x} \cdot \epsilon_i \Delta x = \sum_{i=0}^{n-1} \epsilon_i \Delta x \cdot \prod_{j=0}^{i-1} e^{-\phi_j \Delta x} \tag{1.2}$$

Haciendo algunas reducciones, se obtiene la siguiente ecuación:

$$\begin{aligned} L(x) &= \sum_{i=0}^{n-1} c_i \alpha_i \cdot \prod_{j=0}^{i-1} 1 - \alpha_j \qquad (1.3) \\ &= c_0\alpha_0 + c_1\alpha_1(1-\alpha_0) + c_2\alpha_2(1-\alpha_0)(1-\alpha_1) + \cdots \\ &\quad \cdots + c_{n-1}\alpha_{n-1}(1-\alpha_0)\cdots(1-\alpha_{n-2}) \end{aligned}$$

en la que c_i representa el valor del color y α_i representa el coeficiente de extinción para el i-ésimo voxel de los datos de entrada, tomando n muestras en una dirección de visión dada. La ecuación 1.3 se conoce como la función de composición de VR.

1.1.2. Métodos de Volume Rendering

1.1.2.1. Ray Casting

Es la aproximación más simple a VR y consiste en lanzar rayos desde cada pixel de la pantalla hacia el volumen. Una vez que está el rayo dentro del volumen se muestrea a intervalos constantes para hacer una composición de los colores y opacidades de acuerdo a un operador específico. Las imágenes obtenidas por este métodos poseen un nivel de calidad elevado.

1.1.2.2. Basado en Texturas

Aprovechar las capacidades de manejo de texturas de los aceleradores gráficos actuales ha demostrado ser una opción a la hora de representar datos volumétricos. Hay dos formas de hacerlo: la primera es usando texturas bidimensionales, construyendo conjuntos de texturas alineadas por ejes y proyectando el conjunto cuyo ángulo con respecto al plano de visión sea el menor y finalmente hacer *blending* sobre la tarjeta gráfica. La segunda manera consiste en tomar todo el conjunto de datos y cargarlo en la tarjeta gráfica como una textura tridimensional; para después generar planos paralelos al plano de visión y sobre estos planos, hacer el blending.

1.1.2.3. Splatting

Son técnicas en las que se hace una proyección desde la parte más lejana hacia la parte más cercana del conjunto de voxels con respecto al plano de visión. La contribución que tienen los voxels en la imagen final es calculada por medio de una convolución en una función que sirve de filtro. Mediante la convolución, los valores suelen distribuirse de forma normal, por ejemplo, con una distribución Gauss.

Una ventaja de esta clase de algoritmos es que los voxels se recorren en el orden en el que están almacenados en memoria. Los problemas de este tipo de VR están relacionados con la calidad, debido a lo difícil que se hace generar un conjunto de funciones filtro de manera eficiente y con buenos resultados (en el caso en que la visualización sea en perspectiva, la función de filtro varía de voxel en voxel).

1.1.2.4. Shear Warp

Los algoritmos de visualización que usan acceso a datos espaciales pueden ser clasificados en dos tipos de acuerdo a la manera en la que acceden a tales datos: por orden de imagen y por orden de objeto.

Los de orden de imagen operan por medio de trazado de rayos sobre los datos y procesan las

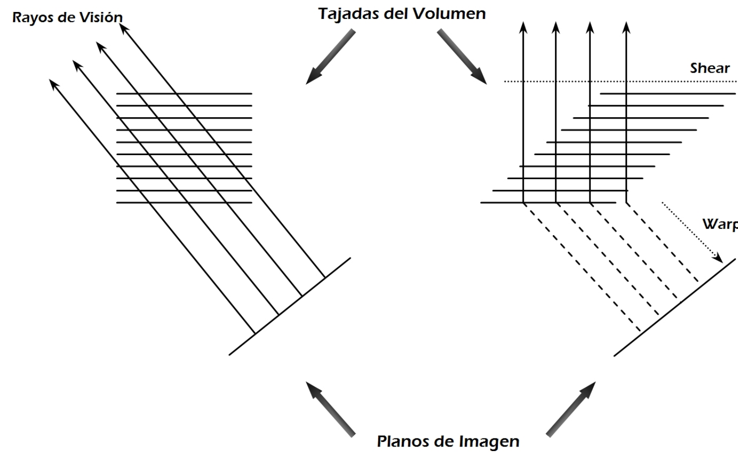

Figura 1.2: Transformación Shear [Lac94].

muestras que están en la trayectoria de cada rayo. Estos algoritmos generan alta calidad y en ellos se pueden aplicar criterios de aceleración como terminación temprana de composición.

En cuanto a los de orden de objeto, se tiene que el acceso a los datos ocurre en el mismo orden en que están almacenados, generando una aritmética de direccionamiento de datos más sencilla y más rápida de ejecutar pero reduciendo la calidad de los resultados.

Lacroute presenta una aproximación en donde es posible tomar los beneficios de los dos tipos de acceso (imagen y orden), pues pretenden resolver el problema de la complejidad del muestreo hacia los datos, usando un sistema de coordenadas intermedio para mapear hacia el sistema de coordenadas del objeto. Esto se logra trasladando secciones del volumen usando la transformación *shear* que puede apreciarse mejor en la figura 1.2 [Lac94].

1.1.3. Aplicaciones de Volume Rendering

En la actualidad, las aplicaciones reales de VR están relacionadas con la visualización de datos médicos – obtenidos de resonancias magnéticas, tomografías computarizadas (CT Scan) o de emisión de positrones (PET Scan) –, o con la visualización de grandes conjuntos de datos geológicos que

hacen las compañías petroleras para encontrar nuevos yacimientos.

VR cobra gran importancia con las aplicaciones médicas, los galenos pueden hacer visualización en 3D de distintos tejidos del cuerpo, dando la capacidad de añadirles o disminuirles contraste, cambiarles el color, rotarlos o hacer acercamientos. Un valor agregado está en poder modificar los valores de opacidad para destacar una mayor cantidad de detalles que permitan hacer el diagnóstico y seguimiento de enfermedades sobre los pacientes.

VR puede ser utilizado en la planeación médica y como recurso educativo con modelos virtuales. Los cirujanos hacen intervenciones quirúrgicas sobre los modelos para prepararse y darse cuenta de posibles problemas al hacer la intervención en la realidad. También se han propuesto como ayuda al momento de la cirugía con la adquisición de datos en tiempo real para mostrar el progreso [Aco97].

VR ha logrado reducir los costos de exploración en la industria petrolera, pues con la visualización no es necesario hacer perforaciones ni trasladar una cantidad enorme de personal para hacer un trabajo lento y costoso[3]. Con los costos de produccion reducidos, es posible invertir en investigación y desarrollo en otras áreas.

A medida que los requerimientos de hardware para hacer VR se hacen mas asequibles, la cantidad de aplicaciones aumenta. Algunas soluciones computacionales de dinámica de fluidos, toman VR para hacer la representación visual de éstos[4]. También se comenta sobre lo mismo para la representación de modelos meteorológicos y fenómenos físicos – cambios de presión atmosférica, distribución en las temperaturas de corrientes marinas, niveles de velocidad del viento y análisis de vorticidad sobre distintos puntos de tormentas o huracanes –[5].

1.1.4. Retos de Volume Rendering

Las reminiscencias cuando se menciona este tema es la gran carga computacional necesaria para hacer la visualización. Con el hardware actual y técnicas recientes es posible llevar a cabo esas tareas

[3] Fuente: Petroleum Technology Transfer Council. `http://www.pttc.org/news/vol4no4.htm`

[4] Fuente: Nasa Jet Propulsion Laboratory. `http://pat.jpl.nasa.gov/public/ParVox/index.html`

[5] Fuente: National Center for Supercomputing Applications. Hurricane Visualization. `http://woodall.ncsa.uiuc.edu/dbock/projects/Opal/`

sobre volúmenes relativamente grandes ($512^3 = 134{,}217{,}728$ muestras) con niveles altos de calidad y a velocidades de procesamiento que permiten interacción.

La fidelidad de la representación es importante durante el proceso de clasificación. Las funciones de transferencia pueden no ser las indicadas y devolver valores que produzcan artefactos sobre la visualización. De éstos, los más graves corresponden a falsos positivos (visualización de una superficie continua donde debería estar una discontinuidad) o falsos negativos (visualización de huecos donde debería estar una superficie).

1.2. Estado del Arte y Tecnología

1.2.1. Antecedentes

Inicialmente las tarjetas gráficas de consumo masivo eran básicamente un /emphbuffer de cuadros, el procesador principal era el encargado de hacer los cálculos involucrados a lo que se dibujaba en pantalla y escribir los resultados en este buffer. A mediados de los años 90, los sistemas con mayor capacidad de procesamiento gráfico eran literalmente máquinas enormes que contenían muchas tarjetas con procesadores especializados, cada uno de ellos con el propósito de hacer un trozo de la tarea de procesamiento, unir los resultados y redibujar la pantalla.

A medida que el tamaño de los chips y los costos de fabricación se reducían, nuevas posibilidades se vislumbraban. Nuevas ideas estaban orientadas a unir todo el diseño antiguo y colocarlo en un solo chip. Ya hacia finales de la década se comenzó a hablar acerca de unidades de procesamiento gráfico, en inglés GPU por *Graphics Processing Unit*. Esas unidades estarían diseñadas para hacer todo lo que sus antecesores de gran tamaño hacían.

La primera generación de estos chips fue lanzada entre 1998 y 1999 con tareas básicas de transformación y procesamiento de vértices. Luego se le fueron añadiendo capacidades de transformación y procesamiento de fragmentos – por ejemplo hacer cálculos de iluminación –[RK03]. Hoy en día es

posible ver tareas como *sorting*[6] sobre GPU usando texturas o simulación de sistemas de partículas en tiempo real[7] como proyectos de investigación.

No es sorprendente ver entonces cómo el número de transistores de los procesadores gráficos supera enormemente al de los procesadores de propósito general, esto es debido a la necesidad de hacer una mayor cantidad de tareas en menos tiempo.

Fabricante	Modelo	Tipo	Número de Transistores	Año	Proceso
Intel	Pentium 4 Extreme Edition	CPU	169 Millones	2005	90 nm
AMD	Athlon 64 FX-57	CPU	114 Millones	2005	90 nm
AMD	Athlon 64 X2 4800+	CPU	233 Millones	2005	90 nm
nVidia	GeForce 7800 GTX	GPU	302 Millones	2005	110 nm
nVidia	Quadro FX 4500	GPU	302 Millones	2005	130 nm
ATI	Radeon X1800	GPU	321 Millones	2005	90 nm
ATI	Radeon X1900	GPU	384 Millones	2005	90 nm

Tabla 1.1: Comparación técnica entre algunos CPU y GPU

En la tabla 1.1[8] se aprecia cómo GPUs recientes duplican la cuenta de transistores de los procesadores de propósito general. Las tareas de procesamiento gráfico que antes demandaban gran uso de procesador (conocidas como tareas de transformación e iluminación) pasaron a un segundo plano, pues se ejecutan ahora sobre la tarjeta gráfica a una velocidad alta, dejando al procesador principal libre para ejecutar otras tareas.

El diseño de los nuevos procesadores gráficos es bastante refinado, tienen capacidades de procesamiento de fragmentos y vértices en paralelo, pues se tienen varios pipelines que pueden tratarse como hilos de ejecución, memoria de alto rendimiento y tasas de llenado por el orden de varios Gigapixels o Gigatexels por segundo[9].

[6]Fuente: A Cache-Efficient Sorting Algorithm for Database and Data Mining Computations using Graphics Processors. `http://www.cs.unc.edu/~geom/SORT/`

[7]Fuente: A Particle System for Interactive Visualization of 3D Flows. `http://wwwcg.in.tum.de/Research/Publications/PartSystem`

[8]Fuentes: nVidia Corp. `http://www.nvidia.com`, ATI Technologies. `http://www.ati.com`, Tom's Hardware. `http://www.tomshardware.com`

[9]Fuente: nVidia Corp. `http://www.nvidia.com/page/fx_5600.html`.

1.2.2. OpenGL

OpenGL es una de las especificaciones de desarrollo gráfico que se ha consolidado como un estándar de la industria. Desde 1992, la Junta de Revisión de la Arquitectura (ARB, por sus siglas en inglés) se ha encargado de orientar el crecimiento de OpenGL hacia un API con un gran rango de capacidades.

La ARB está constituida en su mayoría por grandes compañías de computación como Dell, IBM, Intel, Silicon Graphics Inc., Apple, ATI Technologies, nVidia y Sun Microsystems, entre otras[10]. Las especificaciones de OpenGL son tomadas por los fabricantes de hardware gráfico, para que el diseño de sus productos esté orientado hacia un mayor rendimiento. Cuando alguna de las especificaciones no puede ser ejecutada directamente en el hardware, dicha especificación se procesa en software como una capacidad del manejador (driver).

Lo importante de este proceso es que los desarrolladores sólo se preocupan por el funcionamiento de su aplicación y el correcto uso del API que les provee OpenGL, mientras que los fabricantes del hardware se encargan de la parte de la ejecución de las instrucciones y el rendimiento.

1.2.2.1. Pipeline Gráfico Actual

Como OpenGL es una interfaz de programación para gráficos tridimensionales, los datos correspondientes al render de una escena particular suelen entrar como un flujo de datos y comandos al pipeline de procesamiento.

En la figura 1.3[11] se puede apreciar gráficamente cómo es el proceso de *rendering* de escenas en OpenGL. A continuación se presenta una descripción breve del funcionamiento:

1. Por una parte, se tiene que los comandos pueden almacenarse en la forma de *display lists*[12] o pueden ser procesados dentro de las etapas del pipeline.

[10] Fuente: OpenGL website. `http://www.opengl.org/about/arb/overview/`

[11] Fuente: `http://www.opengl.org/about/overview/`

[12] Los display lists generan un aumento en el rendimiento, en la medida de lo posible los comandos que se ejecutan dentro de ella tratarán de ser pre-compilados y almacenados en la memoria de la tarjeta gráfica.

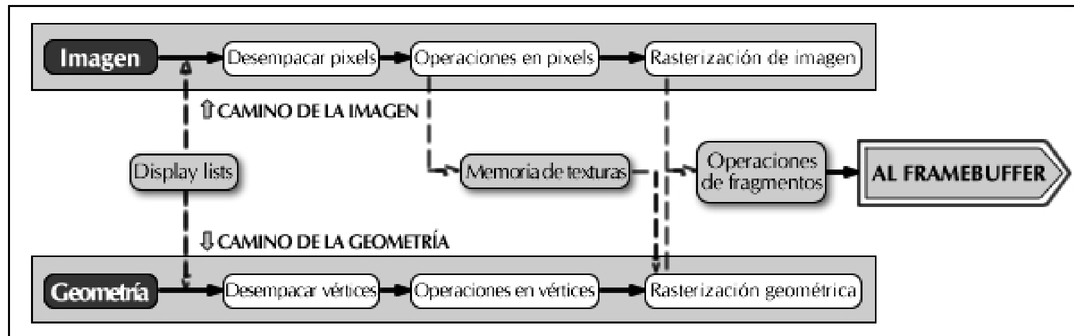

Figura 1.3: Pipeline de visualización en OpenGL

2. La siguiente fase corresponde al evaluador de los puntos que son pasados como flujo de datos. El evaluador es quien se encarga de procesar esos puntos y transformarlos en vértices con sus correspondientes atributos.

3. Una vez que se tienen los vértices en la escena se procede a realizar las operaciones de procesamiento sobre ellos. Estas operaciones son las de transformación, iluminación, creación de las primitivas, *clipping*, proyección y el mapeo a coordenadas del dispositivo de salida.

4. Con la rasterización es que se producen los fragmentos y se aplican las operaciones sobre ellos. Aquí es donde se aplican una serie de pruebas para ver qué fragmentos quedan para dibujar en la pantalla.

5. La última etapa consiste en dibujar el resultado de la rasterización al *framebuffer* en el dispositivo de salida el cual suele ser un monitor.

1.2.3. Cg

Cg significa *"C para gráficos"*. Es un lenguaje de programación orientado a modificar la forma y apariencia de los objetos en la escena. Pero no sólo se limita a eso, es posible hacer simulación física o simulación de movimiento entre otras cosas.

nVidia propuso una revisión inicial del lenguaje como una manera sencilla de acceder a las

capacidades gráficas del nuevo hardware, sin necesidad de atarse a una platarforma en específico. Cg le da a los desarrolladores la capacidad de crear nuevos y mejores efectos de manera fácil y rápida sobre unidades de procesamiento gráfico. Dando un nuevo nivel de abstracción, se elimina la necesidad de utilizar código ensamblador para hacer lo mismo, y aún mejor, puede ejecutarse sin ningún problema en diversas plataformas y sistemas de operación sin hacer cambios en el código.

Perfil	Tipo	Descripción General
vp20, vp30, vp40	Vértices	OpenGL NV_vertex_program
arbvp1	Vértices	OpenGL ARB_vertex_program
vs_1_1	Vértices	DirectX 8 Vertex Shaders 1.1
vs_2_0, vs_2_x	Vértices	DirectX 9 Vertex Shaders 2.0, 2.x
fp30, fp40	Fragmentos	OpenGL NV_fragment_program
fp20	Fragmentos	NV_register_combiners y NV_texture_shader
arbfp1	Fragmentos	OpenGL ARB_fragment_program
ps_1_1, ps_1_2, ps_1_3	Fragmentos	DirectX 8 Pixel Shaders 1.1 - 1.3
ps_2_0, ps_2_x	Fragmentos	DirectX 9 Pixel Shaders 2.0, 2.x

Tabla 1.2: Perfiles de ejecución de programas en GPU

Algo muy particular del enfoque de Cg reside en la especialización de los GPUs, al igual que un programa hecho en C puede correr en distintas máquinas con diferentes procesadores, si se compila usando capacidades extendidas de los procesadores como SSE[13] o para un procesador específico, es muy posible que no logre correr en otros procesdores. Con Cg esto es similar dado que los GPUs aumentan sus capacidades de procesamiento, nuevas funciones y optimizaciones son introducidas en cada iteración, no será posible hacer un estándar hasta que se establezca un mínimo de condiciones necesarias para el desarrollo de los GPUs.

Basado en esto, Cg funciona a nivel de perfiles que definen el tipo y las capacidades del procesador que va a ejecutar el programa en cuestión. Si una funcionalidad deseada no está presente en ese GPU, simplemente el programa no se ejecutará. Esto se logra haciendo compilación dinámica de los programas – en tiempo de ejecución –.

[13]Streaming SIMD Extensions

En la tabla 1.2, tomada de [RK03] están descritos algunos de los perfiles de programas para vértices y fragmentos más usados. En la práctica se suele dejar que, durante la ejecución, se seleccione el perfil óptimo para el hardware con el que se disponga.

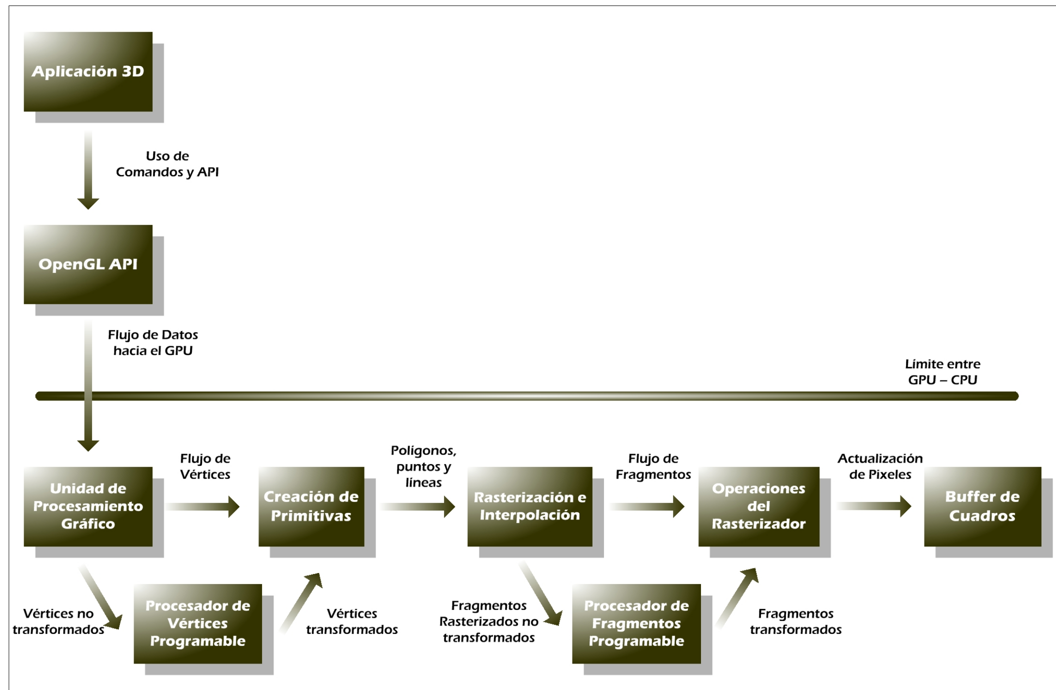

Figura 1.4: Pipeline gráfico programable

Con Cg la capacidad de manipular elementos de una escena está sobre las capacidades de OpenGL, su ejecución está sobre el procesamiento de los vértices y fragmentos de la escena a dibujar, la figura 1.4 tomada de [RK03] muestra el proceso y cuál es la frontera entre el cómputo gráfico y el de propósito general.

1.2.3.1. Justificación del Lenguaje

Como se comenta en la sección anterior, hacer el proceso de programación de vértices y fragmentos es muy complicado sin Cg y nada natural para el programador, el código de las aplicaciones no es fácilmente entendible dejando a un lado el hecho que son escritas en lenguaje ensamblador.

Código 1 Programa de fragmentos – Ensamblador

```
1: DEFINE LUMINANCE = 0.299, 0.587, 0.114, 0.0;
2: TEX H0, f[TEX0], TEX4, 2D;
3: TEX H1, f[TEX2], TEX5, CUBE;
4: DP3X H1.xyz, H1, LUMINANCE;
5: MULX H0.w, H0.w, LUMINANCE.w;
6: MULX H1.w, H1.x, H1.x;
7: MOVH H2, f[TEX3].wxyz;
8: MULX H0.xyz, H0, H1.w;
9: ...
```

Código 2 Programa de fragmentos – Cg

```
1: void basicColor(
   float3 texCoord : TEXCOORD0,
   uniform float4 tissueColor,
   out float4 color : COLOR,
   uniform sampler3D tissueTexture : TEXUNIT0)
   {
2: float4 luminance = tex3D(tissueTexture, texCoord);
3: color = float4(tissueColor.rgb*luminance.r, tissueColor.a*luminance.r);
   }
```

Como se puede apreciar en el código 1, las operaciones de asignación o simple aritmética son extremadamente tediosas. Se debe conocer el lenguaje ensamblador del GPU en el que se está ejecutando el software y es muy común cometer equivocaciones. En el código 2 es más fácil hacer operaciones, Cg define en su librería estándar operadores sobre los tipos empacados – vectores y matrices –, *swizzling* y semánticas que definen el tipo de datos de entrada y salida.

1.2.3.2. Entorno de Ejecución

Cg posee un contexto de ejecución muy específico. En particular, los programas escritos en Cg manejan vértices o fragmentos, lo que no los hace de propósito general. Entre las limitaciones que existen con respecto a un lenguaje de propósito general se pueden enumerar:

1. El contexto de ejecución de Cg está ubicado dentro de programas que utilizan OpenGL o DirectX. No hay modo de ejecutar un programa de Cg de manera *stand-alone*.

2. Cg no tiene funciones de acceso a archivos, todos los parámetros de texturas o datos son procesados como parámetros uniformes desde la aplicación llamante ó son extraídos del contexto de OpenGL o DirectX.

3. No existe el concepto de apuntadores en Cg, por lo tanto, las funciones de reserva y liberación de memoria no están contempladas en el diseño del lenguaje.

1.2.3.3. Rendimiento

El rendimiento de las aplicaciones hechas en el lenguaje Cg comparado a los programas hechos en lenguaje ensamblador, es igual o inclusive mejor. De acuerdo al perfil de GPU con el que se cuente, el compilador dinámico de Cg generará el código más apropiado para correr en él. Esto es algo que no se puede lograr de manera trivial a menos que se conozca todo el conjunto de instrucciones de un GPU específico, sin contar además de quitarle portabilidad al programa.

Capítulo 2

Volume Rendering Directo

2.1. Introducción

En este capítulo se hace una presentación acerca de dos técnicas de VR directo: la primera, un enfoque basado en software usando *ray casting* y la segunda, un enfoque que emplea las capacidades actuales de los chips gráficos, utilizando capacidades extendidas en el manejo de texturas.

Se explica conceptualmente el funcionamiento de cada técnica y la descripción algorítmica de las implementaciones hechas.

2.2. Ray Casting

Una aproximación a VR basado en el trabajo de Levoy [Lev88] consiste en lanzar (cast por su definición en inglés) rayos perpendiculares al plano de proyección. Estos rayos van desde el ojo del observador hacia cada pixel de la imagen y de aquí, hacia adentro del volumen. Una vez que el rayo comienza la trayectoria por dentro del volumen, se colectan muestras de datos en múltiples intervalos de igual separación.

Luego para estas muestras, es posible aplicarles cualquier operador para obtener el valor final del pixel en la pantalla. Cuando se hace la proyección es necesario definir el operador que permi-

tirá obtener el color final del pixel en la pantalla. Existen muchas maneras de definirlo. Algunas de ellas son: [MB05]

Proyección de máxima intensidad es usada para visualizar estructuras de alta intensidad dentro de datos volumétricos. Esto es, durante el muestreo en la proyección, se selecciona el voxel con el valor de intensidad más alto. Este operador suele usarse en aplicaciones médicas para la visualización de estructuras vasculares (angiografía).

Proyección de la intensidad promedio en donde sobre todas las muestras se promedia el valor de la contribución global.

Proyección de la suma se tiene que los resultados obtenidos son similares a los obtenidos al usar rayos X. Lo que se hace es sumar los valores de intensidad sobre las muestras.

Proyección de la composición se refiere a un proceso de interpolación lineal de colores y opacidades de los voxels encontrados durante las proyecciones. Esta es la que se implementa y describe en detalle en el trabajo de Levoy [Lev88].

Para cada uno de los operadores presentados existen aplicaciones bien definidas, la idea general es encontrar la mejor manera de aproximar a la integral que define VR y lograr resultados muy puntuales. Esa integral fue señalada en el capítulo 1 y haciendo reducciones quedó como la ecuación 1.3. Considerando en particular, la proyección por composición, se tiene la posibilidad de realizar el muestreo *back-to-front* o *front-to-back*, obteniendo resultados diferentes para cada uno de ellos.

Como se comentó anteriormente, la proyección de la composición trabaja sobre los valores de colores y opacidades, pero en la adquisición de valores lo que se obtiene comúnmente son valores de intensidad. Tales valores deben pasar por un proceso de clasificación, esto es, para cada voxel, asignar valores de opacidad y color.

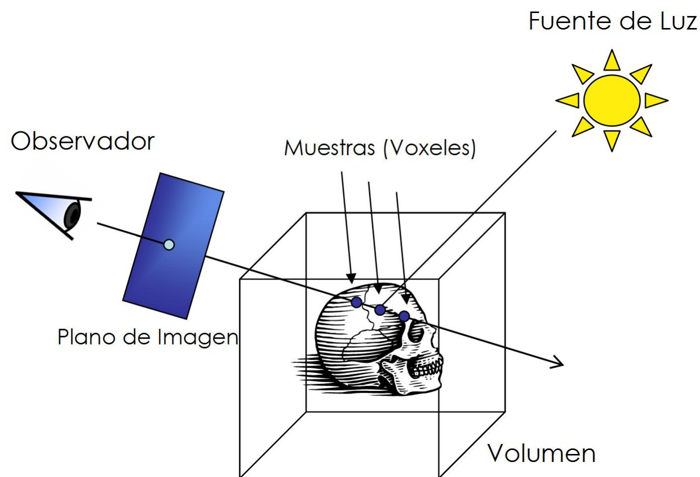

Figura 2.1: Idea general del Ray Casting

2.2.1. Descripción del Algoritmo

La primera etapa del proceso consiste en subir a memoria principal los datos a procesar. Una vez cargada la información, se procede a generar la información de las normales en cada voxel. Como no se generan representaciones geométricas del volumen, se aproxima mediante el gradiente en el punto. El gradiente representa la orientación de superficies locales en el volumen. Con el gradiente se procesa la iluminación y el sombreado del volumen, esto es lo que le da un aspecto menos artificial al resultado visual.

Ahora que se tienen los valores de la iluminación, se procede a lanzar los rayos de acuerdo a la posición del ojo que se desee - en los experimentos realizados por motivos de simplificación, sólo se toman las vistas de frente, de lado y de planta del cubo que representa los datos - y se muestrea el volumen a intervalos regulares sobre el rayo.

Posteriormente se hace la composición de las muestras desde atrás hacia adelante. El valor final del pixel va a ser el último valor de color que dejó la composición. Es posible que el rayo no toque un solo voxel, sino que pase entre algunos de ellos, en cuyo caso es necesario interpolar trilinealmente los valores de los voxels vecinos. La imagen final es el resultado de todas las composiciones finales.

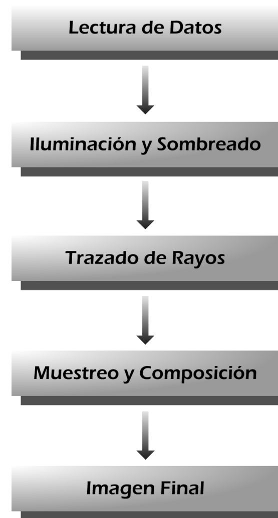

Figura 2.2: Diagrama esquemático del proceso de Ray Casting

Luego se procede a dibujarla en pantalla.

Todo este proceso se puede resumir brevemente en las etapas presentadas en la figura 2.2.

2.2.2. Detalles de Implementación

2.2.2.1. Estructura de los Datos de Entrada

Los datos usados para la experimentación provienen del Instituto de Neurología de Montreal, en la Universidad de McGill [MNI]. Estos datos son generados usando un simulador de resonancia magnética y separados por clases bien diferenciadas usando funciones de transferencia. Mayores detalles se pueden encontrar en [KEP99], [CZK+98], [KEP96] y [CKE97].

Cuando se obtienen los datos, la interfaz de adquisición presenta un encabezado acerca de la estructura interna de estos. Un ejemplo de ello se aprecia en la tabla 2.1 y gráficamente, en la figura 2.3.

Basado en estos datos es posible destacar los siguientes aspectos:

- La última variable de la tabla debe interpretarse como aquélla que varía mas rápido, mientras que la primera es la que varía más lento.

Dimension name	Lenght	Step	Start
zspace	36	5	-72
yspace	217	1	-126
xspace	181	1	-90

Tabla 2.1: Presentación de los datos de [MNI]

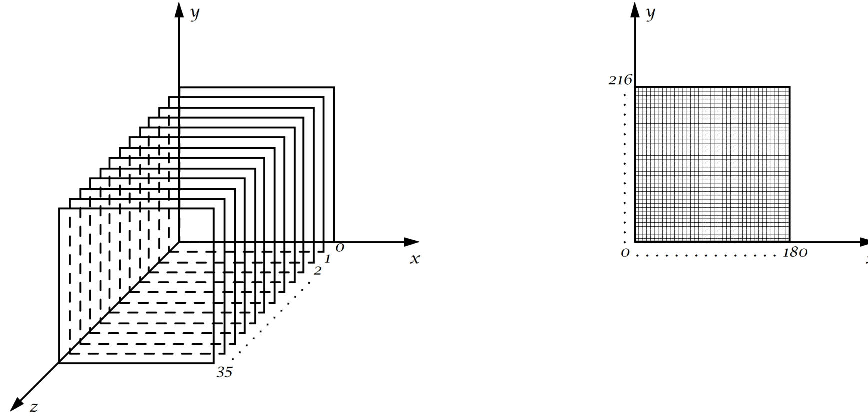

Figura 2.3: Representación interna de los datos de entrada

- El tamaño del cubo de datos en voxels viene dado por $181 \times 217 \times 36$.
- El tamaño real del voxel es de $1 \times 1 \times 5\ mm^3$ según las mediciones del simulador.

En cuanto a los requerimientos de espacio pueden generalizarse las cifras. Si se tiene un volumen V que contiene R_x planos en el eje x, R_y para el eje y, y finalmente R_z para el eje z. La cantidad de voxels es $R_x \times R_y \times R_z$, si se tiene que el contenido del voxel es de n bytes, entonces el tamaño requerido para cargar el volumen V es de $R_x \times R_y \times R_z \times n$ bytes.

2.2.2.2. Cálculo de las Normales

La normal, por definición, es el vector perpendicular a la tangente de un punto sobre una superficie. En el caso de VR, no existen modelos geométricos que definan el volumen, y se hace necesario hacer una aproximación numérica al valor de la normal.

Si se supone que existen superficies locales dentro del volumen debido a las interfaces entre materiales, entonces podemos aproximar el valor numérico utilizando el gradiente por diferencias centrales. La ecuación es la siguiente:

$$N(x_i) = \frac{\nabla f(x_i)}{|\nabla f(x_i)|} \tag{2.1}$$

En donde $\nabla f(x_i)$ es el gradiente del voxel x_i, equivalente en forma de ecuación a:

$$\begin{aligned}
\nabla f(x_i) &= \nabla f(x_i, y_j, z_k) \\
\nabla f_x(x_i, y_j, z_k) &\approx \frac{1}{\Delta x}\left(f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)\right) \\
\nabla f_y(x_i, y_j, z_k) &\approx \frac{1}{\Delta y}\left(f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)\right) \\
\nabla f_z(x_i, y_j, z_k) &\approx \frac{1}{\Delta z}\left(f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})\right)
\end{aligned} \tag{2.2}$$

Con estas ecuaciones se aproxima el valor de la normal, posteriormente cada una de las componentes se divide entre el módulo del vector para llevarlo a una forma unitaria. Estos valores se calculan una sola vez, justo después de la carga de los archivos.

2.2.2.3. Iluminación

La iluminación presenta el objeto y se debe decidir sus características, teniendo en cuenta que al hacer VR no se pretende lograr realismo, sino mejorar la calidad de la representación para ayudar a comprender los datos que se están visualizando.

En esta implementación se utilizó el modelo de Phong en su formulación de vector medio [FDF+94]. Este modelo propone que la iluminación de un objeto en el espacio, dada una fuente de luz particular, esté determinada por la suma de las contribuciones de las componentes ambientales, difusos y especulares sobre dicho objeto.

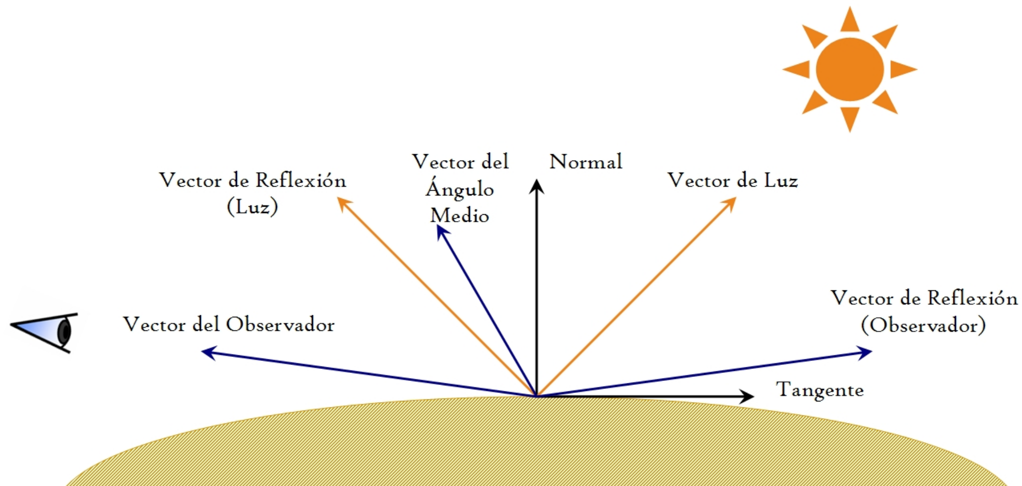

Figura 2.4: Vectores en el modelo de Iluminación del ray caster [MB05].

Seguidamente sepresenta una breve descripción de las componentes del modelo usado:

Iluminación Ambiental representa una luz difusa y no direccional que ilumina a todos los objetos. Está determinada por $I = I_a k_a$ donde I_a es la intensidad de la luz ambiental, y k_a un coeficiente de reflexión ambiental de un objeto.

Iluminación Difusa representa la iluminación producida por una luz puntual omnidireccional, independiente de la posición del observador. Esta contribución en una superficie depende de la posición de la luz y el vector normal de la superficie. En particular, la ecuación describe un proceso de reflexión de Lambert, se define como $I = I_d k_d \left(\vec{N} \cdot \vec{L}\right)$, siendo I_d la intensidad de esta luz, k_d un coeficiente de reflexión difusa de la superficie, y $\vec{N}$ y $\vec{L}$ los vectores normal y posición de la luz, normalizados.

Iluminación Especular representa el brillo producido por el reflejo directo de la luz sobre el ojo y es dependiente de la posición del observador. La contribución especular de la iluminación viene dada por: $I = I_s k_s \left(\vec{N} \cdot \vec{H}\right)^n$, donde el vector $\vec{H}$ es el vector cuya dirección está entre la dirección del vector de la luz y el vector hacia el observador. Este vector indica la dirección

del máximo brillo, y su valor es el coseno del ángulo de los vectores que lo conforman. El valor n representa el exponente de reflexión especular. Si la superficie fuese un reflector ideal, este valor sería infinito.

Iluminación Final dados los términos de las contribuciones ambiental, difusa y especular del proceso de iluminación, se tiene entonces una fórmula que determina el valor final del color para cualquier elemento del volumen en estudio $I = I_a k_a + I_d k_d \left(\vec{N} \cdot \vec{L}\right) + I_s k_s \left(\vec{N} \cdot \vec{H}\right)^n$.

2.2.2.4. Interpolación Trilineal

Cuando uno de los rayos lanzados sobre el volumen no coincide exactamente sobre los elementos internos del volumen existe la interrogante sobre cuál de los voxels seleccionar. En muchos casos sería conveniente tomar el más cercano en cuanto a distancia, pero no es suficiente, pues pueden generarse errores de reconstrucción del volumen conllevando a errores en la visualización.

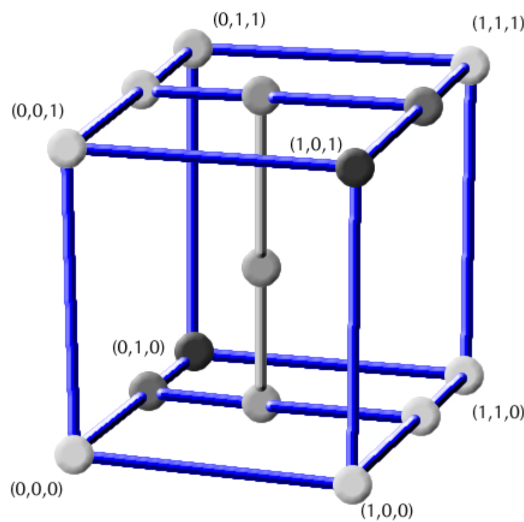

Figura 2.5: Descripción espacial de la interpolación trilineal

La intención de interpolar con los voxels vecinos es la de garantizar que la contribución de éstos sea empleada para generar una mejor aproximación al valor real. Para encontrar el valor numérico

se toman las contribuciones de los voxels vecinos y su distancia al punto. Se expresa de la siguiente manera:

$$
\begin{aligned}
V_{xyz} &= V_{000}x'y'z' + V_{100}xy'z' + V_{010}x'yz' + V_{001}x'y'z + \\
&\quad V_{101}xy'z + V_{011}x'yz + V_{110}xyz' + V_{111}xyz
\end{aligned} \tag{2.3}
$$

de donde x, y, z corresponden a la distancia del punto hacia el voxel. Mientras que x', y', z' corresponden a $(1 - x)$, $(1 - y)$, $(1 - z)$ respectivamente. Se toma el valor de la unidad (1) pues es la distancia entre voxels en todas las direcciones.

En la figura 2.5[1] se muestra cómo a partir de los 8 vecinos a un punto cualquiera en el espacio se pueden relacionar sus distancias y valores.

2.2.2.5. Composición

En el proceso de composición se obtienen los valores finales para el color sobre un punto en la pantalla. En el capítulo 1, cuando se explica el proceso físico involucrado en VR se hace referencia a la ecuación 1.3 como la relación entre los coeficientes de extinción y el color de los distintos puntos sobre la trayectoria de un haz de luz en el volumen.

La ecuación viene dada por la siguiente recurrencia [WS01]:

$$
\begin{aligned}
C_0 &= 0 \\
A_0 &= 1 \\
C_i &= C_i + C_{i-1} \times A_{i-1} \\
A_i &= A_i \times A_{i-1}
\end{aligned}
$$

[1] Fuente: `http://groups.csail.mit.edu/graphics/classes/6.837/F98/Lecture22/Slide24.html`

2.2.2.6. Resultado Final

Después de aplicar todas las etapas explicadas anteriormente se obtiene una imagen que representa gráficamente los datos. En la figura 2.6 se muestra el resultado de aplicar ray casting sobre un volumen de datos de materia gris.

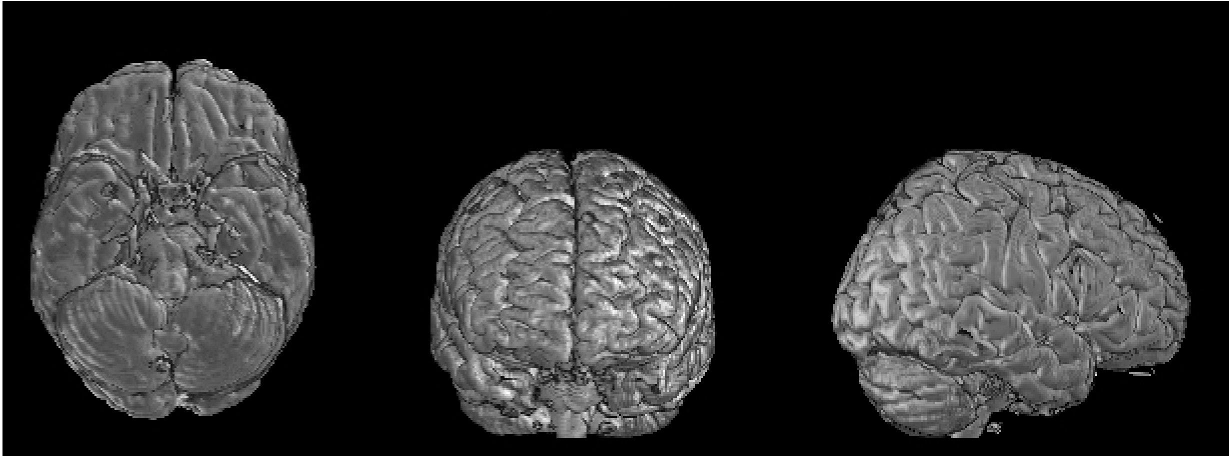

Figura 2.6: Visualización de datos usando ray casting

2.3. Volume Rendering Basado en Texturas

Como se explicó en el marco teórico, las tarjetas gráficas son capaces de aplicar imágenes, llamadas texturas, a primitivas de dibujo en una pantalla. Se puede aprovechar esta capacidad para hacer VR a velocidades interactivas empleando el volumen de datos como texturas dentro de las tarjetas gráficas. Las texturas se utilizan de la siguiente manera: la opacidad de los voxels en la clase se traducen a la opacidad en un *texel* y la operación de blending que ofrece el hardware de la tarjeta provee el operador de composición necesario para reconstruir la vista del volumen. Los tipos de texturas que se cargarán en las tarjetas gráficas son de tipo bidimensional y tridimensional. Nuestra implementación mediante texturas 2D se basó en la técnica mencionada por [WS01], y la

implementación mediante texturas 3D fue estudiada a partir del trabajo de Cullip y Neumann en [CN94].

En esta sección se describe el uso de cada técnica de VR por medio de manejo de texturas, el diagrama esquemático, los algoritmos aplicados y las consideraciones tomadas en cuenta en nuestras implementaciones.

2.3.1. VR Basado en Texturas 2D

Esta técnica consiste en separar el volumen de datos en una serie de texturas 2D, las cuales se montan sobre una geometría intermedia (*proxy geometry*, en inglés) orientada al objeto. En este caso el objeto es el volumen a visuaizar. Al dibujar esta geometría comenzando por la textura más lejana a la cámara, cada textura se superpone a la siguiente y se deja la tarea de composición a la tarjeta gráfica, la cual mediante la operación de blending irá reconstruyendo el volumen emulando la integral de VR. La superposición de texturas se ilustra en la figura 2.7.

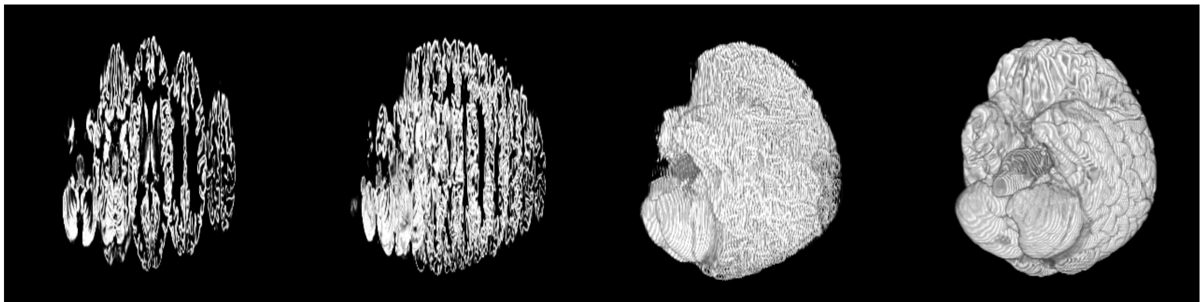

Figura 2.7: Superposición de texturas 2D

Ya que en este trabajo se emplean datos pre-clasificados, el único componente que se incluye en las texturas es la opacidad. Suponemos que una clase posee un color uniforme, el cual es multiplicado por la opacidad (*opacity-weighted color*), para simular la composición sobre el color negro. En el caso de combinar múltiples clases, se hace necesario también incluir los colores correspondientes a cada clase del volumen.

2.3.1.1. Algoritmo

Creación de texturas y carga en la tarjeta gráfica

1. Para cada eje X, Y y Z:

    a) Para cada plano paralelo a un eje:

        1) Crear una textura bidimensional de ese plano.

        2) Cargar dicha textura en la memoria de la tarjeta gráfica.

Render

1. Determinar el eje del objeto cuyo ángulo sea el menor al plano de proyección (X, Y o Z).

2. Del conjunto correspondiente al eje escogido en el paso anterior, determinar cuál es la textura que está más lejana a la cámara.

3. Rotar la cámara alrededor del objeto y dibujar el conjunto de texturas correspondientes al eje escogido en el paso 1, comenzando por la textura escogida en el paso 2.

2.3.1.2. Diagrama Esquemático

Para realizar VR con texturas bidimensionales, se crean los conjuntos de texturas correspondientes a los 3 ejes de visualización (creación de texturas) a partir del volumen de opacidad y se cargan estas texturas a la memoria de tarjeta gráfica.

Después de esto, se inicializa el contexto de OpenGL y GLUT, estableciendo el tamaño de la ventana y la matriz de proyección, utilizando una proyección ortogonal.

A continuación, se crean 6 *display lists*, correspondientes a cada orden de render (back-to-front o front-to-back) de cada eje de visualización. Compilar las 6 maneras de representarse el volumen en OpenGL a través de los display lists permite reducir el número de llamadas que se hacen en el código para mostrarlas en pantalla.

Finalmente se hace el render de las texturas cargadas, primero determinando cuál es el eje de visualización más paralelo al eje de proyección, luego determinando cuál será el orden de render de las texturas de ese plano, y mostrando el display list correspondiente a los parámetros determinados anteriormente. Este proceso se esquematiza en la figura 2.8.

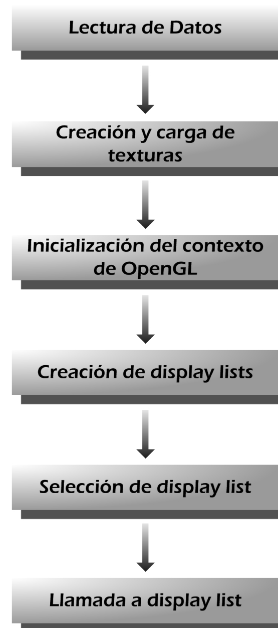

Figura 2.8: Diagrama esquemático de la técnica de texturas 2D y texturas 3D con planos orientados al objeto.

2.3.1.3. Detalles de Implementación

Carga de las texturas en la tarjeta gráfica

Esta técnica hace uso de la misma estructura de datos que la técnica de ray casting, con unas variaciones debido a las limitaciones actuales de OpenGL. El resultado del procesamiento de los datos, realizado previamente en [MNI], son volúmenes que representan la opacidad de las clases. Estos volúmenes pueden ser cargados en la memoria de la tarjeta gráfica como texturas con un solo

componente, el cual representa la opacidad correspondiente a la posición en el volumen.

En OpenGL existen varios formatos internos con los que se pueden almacenar los datos. Un formato posible para adoptar es el de intensidad, conocido como `GL_INTENSITY` en la librería, pero el hardware sobre el que se hicieron las pruebas no soporta este formato, por lo que se decidió utilizar el componente de luminosidad (`GL_LUMINANCE`) en las texturas a cargar en la tarjeta. Durante la etapa de render, implementamos un *fragment shader* el cual se encarga de obtener el valor de la textura y multiplicar ese valor por los componentes rojo, verde, azul y de opacidad establecidos por el usuario, se especifica la fuente en el código 3.

Código 3 Programa de fragmentos – Asignacion del color sobre el GPU

```
1: void basicColor(
   uniform sampler3D tissueTexture : TEXUNIT0,
   float3 texCoord : TEXCOORD0,
   uniform float4 tissueColor,
   out float4 color : COLOR)
   {
2:   float4 luminance = tex3D(tissueTexture, texCoord);
3:   color = float4(tissueColor.rgb*luminance.r, tissueColor.a*luminance.r);
   }
```

El tamaño original del volumen de datos es de $181 \times 217 \times 181$. OpenGL añade una limitación sobre las dimensiones de las texturas, indicando que deben ser potencias de dos. Este hecho llevó a que los datos se colocaran, sin ningún tipo de redimensión, en texturas de 256^2, es decir, el tamaño más pequeño donde se puede ajustar los datos originales, llenando el resto de la textura con ceros. La cantidad de texturas bidimensionales necesarias para representar la totalidad del volumen es de $181 + 217 + 181 = 579$. La cantidad de memoria principal necesaria para la creación de las texturas es de $256^2 \times (181 + 217 + 181) = 37{,}945{,}344$ bytes. Ya que OpenGL deja el formato interno de la textura a elección de los implementadores del hardware, no es posible precisar si la misma cantidad de memoria es empleada cuando la textura se almacena en la memoria de la tarjeta gráfica.

Creación de Display Lists

Se puede notar que en la técnica existen 6 maneras de visualizar el volumen dado el número de ejes de visualización (3) y las maneras de ordenarlos a la hora de hacer render (2). Por ello, se decidió aprovechar el mecanismo de display lists en OpenGL para acelerar el proceso.

Los display lists son estructuras donde se pueden compilar comandos de OpenGL para luego ser llamados a través de la función `glCallList/1`. Los comandos que son introducidos en estas estructuras se guardan dentro de la tarjeta gráfica, lo cual reduce el tiempo que tardan en ejecutarse al momento de ser llamadas.

Rendering de un Display List

La selección del display list a llamar depende de cuál eje de visualización tiene el menor ángulo con respecto al plano de proyección y del orden en el que se vayan a dibujar las texturas. Nuestra implementación utiliza dos ejes de rotación del objeto, X e Y, y estos dos parámetros determinan lo mencionado anteriormente. En consecuencia, la selección depende directamente del valor de estos ejes de rotación. Por ejemplo, si en el eje Y el volumen está rotado entre los 315 y 45 grados, y el eje X entre el mismo rango, entonces el eje de visualización más paralelo al eje de proyección sería el eje X, lo cual mostraría el volumen utilizando el conjunto de planos YZ. En la figura 2.9 se puede apreciar los tres conjuntos de texturas, haciéndose evidente los ángulos en los que se debe cambiar de conjunto.

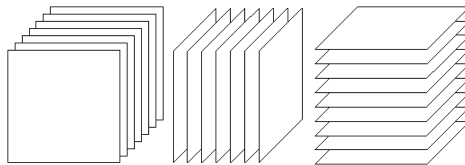

Figura 2.9: Los 3 conjuntos de texturas usados en 2D son perpendiculares entre sí

2.3.1.4. Calidad de las Imágenes Obtenidas

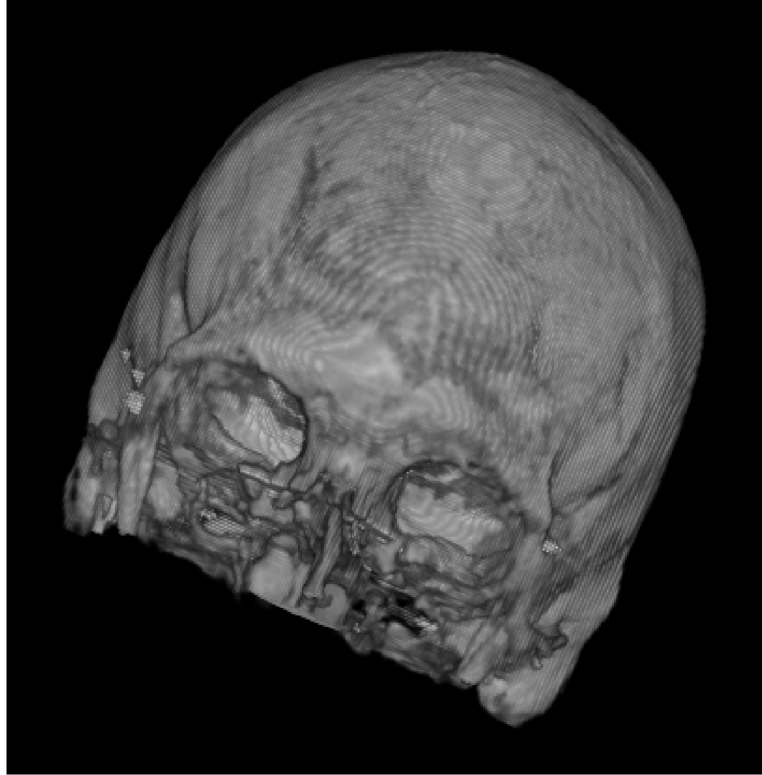

Figura 2.10: Visualización utilizando texturas 2D

Las imágenes obtenidas presentan bandas alrededor del volumen debido a que sólo se aplica una interpolación bilineal para cada textura del volumen, lo cual produce una reconstrucción errónea del volumen con respecto a lo capturado inicialmente. Esta técnica al considerar cada plano del volumen bidimensionalmente, solamente implementa interpolaciones bilineales, ignorando la tercera dimensión. Como resultado se hacen presentes bandas muy visibles alrededor de cada textura (*banding*). Un ejemplo de esto puede ser apreciado en la figura 2.10.

Con el uso de los 3 conjuntos de texturas, se introduce otro tipo de artefactos denominado *popping*, ya que siempre se nota el cambio de conjuntos de texturas, bien sea por el cambio de orientación de los mismos artefactos creados por la interpolación bilineal, o el tiempo que puede

tomar el intercambio de texturas si la capacidad de memoria de la tarjeta no es la suficiente.

Nótese que en esta técnica al rotar la cámara con respecto al volumen, se cambia la frecuencia de muestreo del volumen produciendo artefactos. En casos extremos, hará que las primeras queden totalmente perpendiculares al plano de proyección, por lo que no se verían en la imagen final. Una solución parcial a este problema es utilizar los tres conjuntos de texturas explicados anteriormente, y mostrar exclusivamente el conjunto cuyo ángulo en relación al plano de proyección sea el menor.

2.3.2. VR Basado en Texturas 3D

A diferencia de la técnica de texturas bidimensionales, en la que hay que cargar varias texturas generadas a partir del volumen de datos, en esta técnica hay que crear una sola textura. En el momento del render se asignan las coordenadas de texturas correspondientes a las geometrías intermedias, dejando la tarea de interpolación trilineal a la tarjeta gráfica. Esta técnica ha sido explorada también por Ikits et al. en [IKLH04].

Esta técnica no se limita al uso de planos orientados al objeto para muestrear el volumen de datos, como en el caso de las texturas bidimensionales, sino que es posible tener geometrías intermedias orientadas siempre al plano de proyección, eliminando el problema generado por el cambio de conjuntos para determinados ángulos de rotación de la cámara, con la posibilidad de introducir otro tipo de artefactos. La figura 2.11 muestra las diferencias entre los tipos de orientación. Esta geometría se puede derivar de dos maneras: obteniendo los planos paralelos al plano de proyección que intersecten el volumen de datos, y calculando para cada vértice sus respectivas coordenadas de texturas; o teniendo un conjunto de polígonos con unas coordenadas de texturas fijas, modificando la matriz de coordenadas de texturas para efectuar la rotación. La figura 2.12 ejemplifica la primera manera.

Con esta técnica implementamos dos programas: uno utiliza planos orientados al objeto, y el otro emplea planos orientados al plano de proyección.

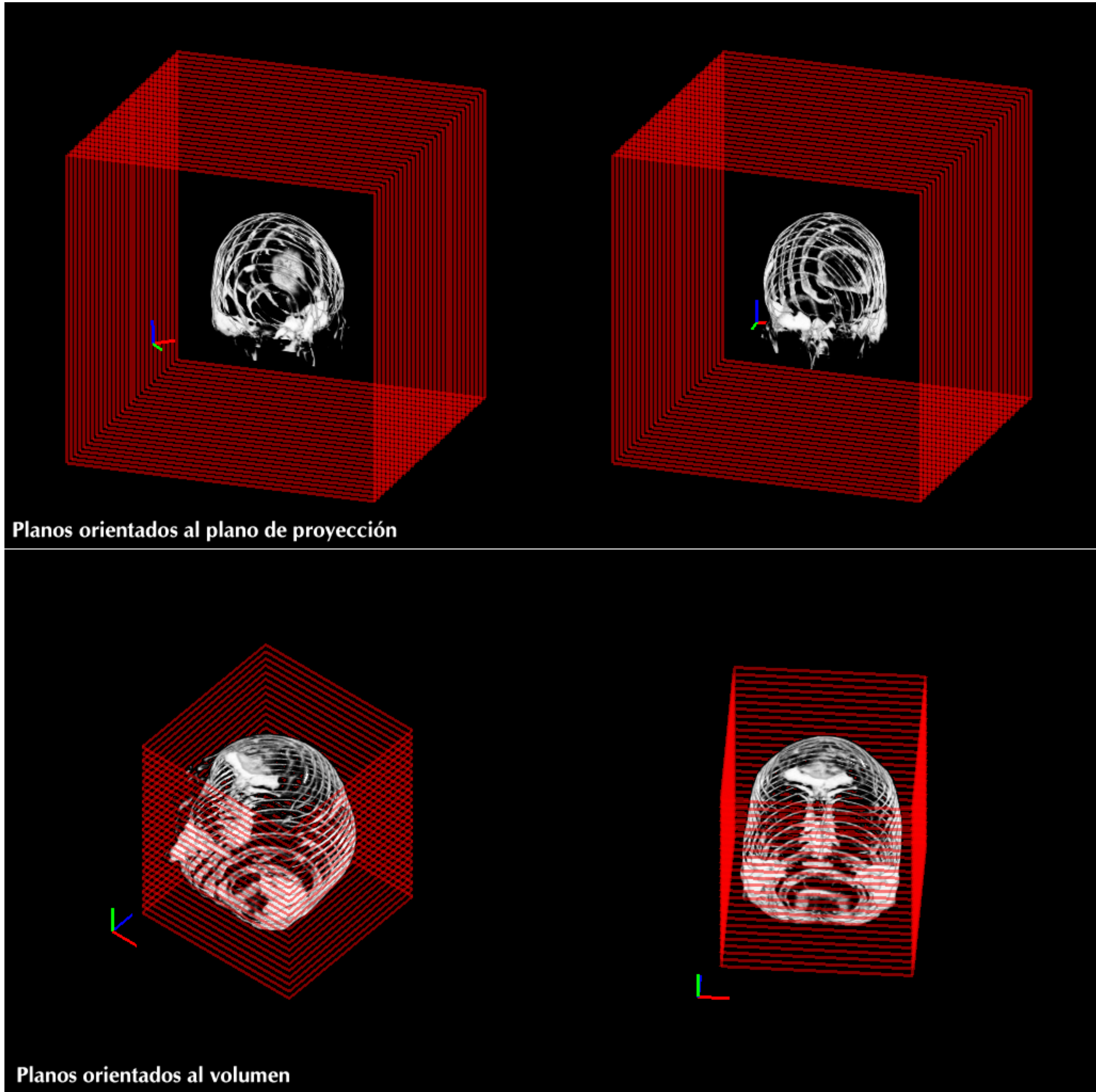

Figura 2.11: Diferencias visuales entre los planos orientados al volumen, y los orientados al plano de proyección. Las geometrías intermedias están delineadas en rojo.

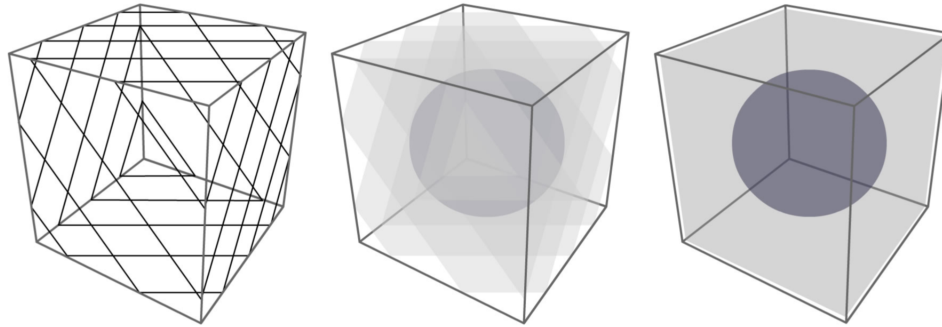

Figura 2.12: Geometrías intermedias orientadas al plano de proyección [WS01].

2.3.2.1. Algoritmo

Creación de textura y carga en la tarjeta gráfica

1. Reservar un espacio en memoria cuyas dimensiones sean las potencias de dos más pequeñas que puedan contener el volumen de datos.
2. Rellenar el espacio con ceros.
3. Cargar el volumen de datos dentro del espacio. Los datos en el espacio ocupan los mismos lugares que en el volumen.
4. Cargar el espacio reservado como una textura tridimensional a la tarjeta gráfica.

Render con geometrías orientadas al volumen

1. Determinar el eje del objeto cuyo ángulo sea el menor al plano de proyección (X, Y o Z).
2. Del conjunto correspondiente al eje escogido en el paso anterior, determinar cuál es la textura que está más lejana a la cámara.
3. Rotar la cámara alrededor del objeto y dibujar el conjunto de texturas correspondientes al eje escogido en el paso 1, comenzando por la textura seleccionada en el paso 2.

Render con geometrías orientadas al plano de proyección

1. Trasladar la textura y rotarla de acuerdo a los parámetros escogidos por el usuario.

2. Hacer render de la textura utilizando las geometrías intermedias en el plano ST, variando sobre el eje R.

2.3.2.2. Diagrama Esquemático

Para realizar el VR con texturas tridimensionales en general, es necesario crear una sola textura y cargarse en la memoria de la tarjeta gráfica. El diagrama esquemático de la técnica con planos orientados al volumen es similar a la técnica que utilizamos con las texturas bidimensionales, y se representa en la figura 2.8.

A diferencia del uso de las texturas bidimensionales, la tarjeta gráfica se encarga de hacer el muestreo y reconstrucción del volumen, utilizando la interpolación trilineal disponible mediante hardware, sin importar las coordenadas de texturas que se escojan. En el caso de los polígonos orientados al plano de proyección, estos polígonos tienen coordenadas fijas, sin modificarse la matriz de modelado y sólo se manipula la matriz de textura. El diagrama esquemático de esta variante se muestra en la figura 2.13.

2.3.2.3. Detalles de Implementación

Uso de Memoria

Los volúmenes de datos se representan de una manera más directa cuando se usan texturas tridimensionales. En este caso, la textura creada es de 256^3 debido a la limitación de las dimensiones de las texturas en OpenGL. El volumen de datos original se copia a esa textura sin ningún tipo de redimensionamiento, dejándose esta tarea a la tarjeta gráfica, una técnica que se explica en el siguiente apartado en los detalles de implementación. Las texturas en esta implementación ocupan en memoria principal 16MiB.

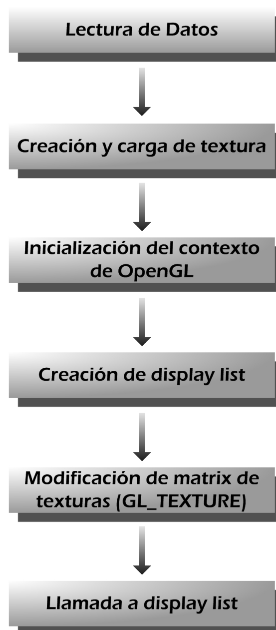

Figura 2.13: Diagrama esquemático de la técnica de texturas 3D con planos orientados al plano de proyección.

Creación de los Display Lists

En el caso de las geometrías orientadas al objeto, la creación de los display lists se hace de manera similar al caso de las texturas bidimensionales. A la hora del render, se dibujan tantos polígonos como voxels tenga el volumen en el eje que se está visualizando. A diferencia de la anterior técnica, en este caso se manejan 3 coordenadas de texturas llamadas S, T y R. Los ejes S y T se corresponden los ejes del mismo nombre en las texturas bidimensionales, y el eje R se corresponde con el eje Z de las coordenadas locales de los polígonos sobre los que se aplican estas texturas.

Como no se hace redimensión del volumen para que ocupe todo el espacio de la textura en la tarjeta gráfica; la redimensión del volumen es realizada por la tarjeta asignándole a las primitivas, las coordenadas en las cuales finaliza el volumen de datos y comienzan los ceros de la textura cargada. Por ejemplo, si el volumen de datos tiene 181 voxels en los ejes X y Z, y 217 voxels en el eje Y, entonces en los 3 ejes S, T y R la textura debe tener 256 voxels (256 es el menor

número potencia de dos que contiene a 181 y a 217), y dada la coordenada $(0, 0, 0)$ del volumen de datos como origen de las coordenadas de textura, la coordenada donde finaliza el volumen sería $\left(\frac{181}{256}, \frac{217}{256}, \frac{181}{256}\right) = (0{,}7070, 0{,}8476, 0{,}7070)$. La figura 2.14 muestra la relación de los datos de nuestra implementación con el tamaño de la textura donde se carga. La línea roja indica los límites del volumen, y la línea azul muestra el tamaño de la textura. El punto verde indica el centro del volumen.

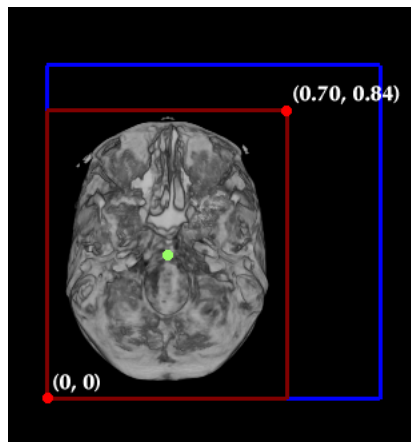

Figura 2.14: La relación de los datos de nuestra implementación con la textura donde se carga.

En el caso de las geometrías orientadas al plano de proyección, se crea exclusivamente un display list ya que únicamente existe un modo de hacer render de las geometrías intermedias, modificando solamente la matriz de textura.

Polígonos Orientados al Plano de Proyección

La orientación de los polígonos al plano de proyección consiste en posicionar las primitivas en un sitio fijo y manipular la matriz de texturas para simular la rotación del volumen.

Para lograr la rotación del volumen, previamente se debe asignar unas coordenadas de textura a los polígonos y agregar una serie de traslaciones a la matriz de texturas de tal manera que el centro del volumen coincida con el centro del conjunto de polígonos a los que se le asigna la textura. Si

el volumen ocupa exactamente toda la textura tridimensional, no es necesario hacer traslaciones a la matriz de texturas. En nuestra implementación, esto no se cumple, y se puede notar en la figura 2.14, donde el punto verde indica el centro del volumen, y claramente no coincide con el centro de la textura.

El tamaño de las primitivas debe dar cabida a todo el volumen. Para que el volumen quepa dentro de las primitivas dada cualquier rotación, se establece como mínimo, el tamaño de la diagonal del volumen. Esto se aprecia en la figura 2.11.

2.3.2.4. Calidad de las Imágenes Obtenidas

Las imagenes obtenidas presentan bandas que cruzan el volumen, el ancho de estas bandas depende de la cantidad de planos que interceptan el volumen, y la orientación de las mismas depende de la orientación del volumen. Al hacerse efectivamente la interpolación trilineal sobre la textura, las bandas que se presentaban con la texturización bidimensional desaparecen. Esto puede ser apreciado en la figura 2.15 que muestra el resultado de esta aproximación empleando como entrada datos de materia gris.

Al utilizarse polígonos orientados al objeto se reduce en gran medida el popping que ocurre al cambiar de conjunto de polígonos, aunque todavía se nota levemente. El uso de polígonos orientados al plano de proyección, en cambio, introduce bandas horizontales o verticales al volumen, dependiendo del número de polígonos que se utilicen para muestrear el mismo. El aumento del tamaño de las primitivas para hacer que ajuste todo el volumen puede afectar de manera negativa el desempeño del render, a menos que se aplique una prueba de alpha para evitar que los fragmentos transparentes se dibujen al framebuffer.

2.3.2.5. Múltiples Volúmenes

Una funcionalidad que proveen los chips gráficos recientes es la capacidad de manejar simultáneamente múltiples elementos de textura. Una aproximación a la visualización de un volumen de datos

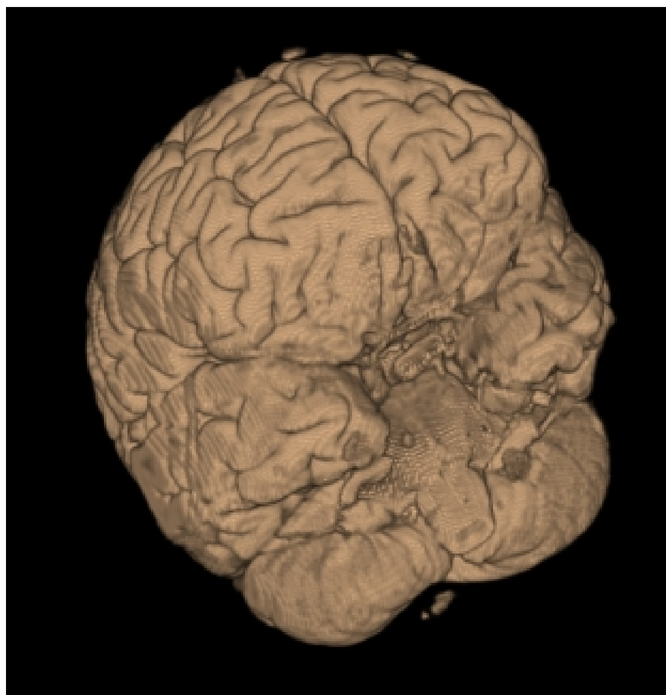

Figura 2.15: Visualización utilizando texturas 3D

con varias clases al mismo tiempo puede ser alcanzada utilizando una textura para cada clase. El límite en la capacidad de unidades de manejo de texturas lo pone el hardware que se emplea. Para el caso de los chips GeForce FX de nVidia el número máximo de unidades de textura utilizando la extension `ARB_multitexture` es 4, mientras que usando la extensión `NV_fragment_program` se documenta el acceso simultáneo a un máximo de 16 unidades de textura.

Para implementar esta técnica se toman los valores de opacidad de cada clase, para cada una se asigna una unidad de textura independiente en el contexto de OpenGL mediante el uso de `glActiveTextureARB/1`. Una vez cargadas las texturas, se dibujan sobre las mismas primitivas para luego hacer una interpolación de las opacidades usando el fragment shader mostrado en el código 4.

En la figura 2.16 puede apreciarse cómo reduciendo las opacidades de las clases se pueden obtener resultados de alta calidad y con alto índice de interactividad. Para esa representación se tomaron

las clases correspondientes a materia gris, músculo y piel, grasa y hueso.

Código 4 Programa de fragmentos – Composición de 4 clases simultáneamente

```
1: void main(float3 texCoord : TEXCOORD0,
   uniform float4 tissueColor[4],
   uniform sampler3D tissueTexture1 : TEXUNIT0,
   uniform sampler3D tissueTexture2 : TEXUNIT1,
   uniform sampler3D tissueTexture3 : TEXUNIT2,
   uniform sampler3D tissueTexture4 : TEXUNIT3,
   out float4 color: COLOR)
   {
2: float4 luminance1 = tex3D(tissueTexture1, texCoord);
3: float4 luminance2 = tex3D(tissueTexture2, texCoord);
4: float4 luminance3 = tex3D(tissueTexture3, texCoord);
5: float4 luminance4 = tex3D(tissueTexture4, texCoord);
6: float4 color1 = float4(tissueColor[0].rgb*luminance1.r, tissueColor[0].a*luminance1.r);
7: float4 color2 = float4(tissueColor[1].rgb*luminance2.r, tissueColor[1].a*luminance2.r);
8: float4 color3 = float4(tissueColor[2].rgb*luminance3.r, tissueColor[2].a*luminance3.r);
9: float4 color4 = float4(tissueColor[3].rgb*luminance4.r, tissueColor[3].a*luminance4.r);
10: color = color1 + color2 + color3 + color4;
   }
```

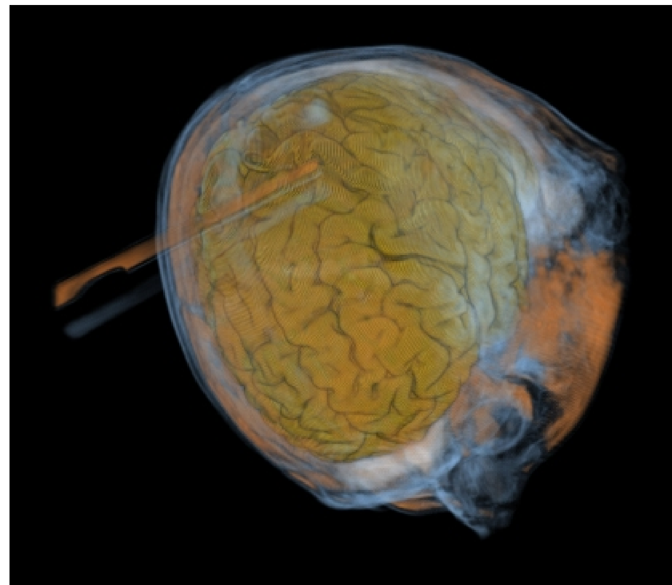

Figura 2.16: Visualización de múltiples volúmenes utilizando texturas 3D

Capítulo 3

Técnicas de Pre-integración de Volume Rendering

3.1. Introducción

En este capítulo se presentan en detalle la investigación de las propuestas de Westermann & Sevenich [WS01] y Engel, Kraus & Ertl [EKE01] para acelerar el proceso de VR. Los autores proponen precalcular partes de la integral de VR y tenerla almacenada para eliminar la complicación de la continua evaluación.

3.2. Metodología

Para la investigación de las propuestas mencionadas se realizaron las siguientes actividades:

- Lecturas correspondientes a técnicas de aceleración en VR.
- Lectura de los trabajos de Westermann & Sevenich y de Engel, Kraus & Ertl y discusiones sobre su implementación.
- Se hicieron las implementaciones de cada una de las propuestas.

- Se definió un conjunto de pruebas para verificar los conceptos a través de nuestras implementaciones.

3.3. Técnica de Westermann & Sevenich

El enfoque de la técnica presentada por Westermann & Sevenich es la de realizar VR directo optimizando su velocidad. Esta optimización está directamente relacionada con la manera en la que se evalúa la integral de VR. En este caso en particular, se eliminan de la ecuación aquellos términos que no van a contribuir en el resultado final de la visualización [WS01].

Anteriormente se explicó el funcionamiento de VR como un proceso óptico que involucra la interacción de la luz con los objetos de un espacio, éste se resume en la siguiente ecuación:

$$L(x) = \int_{x_A}^{x_B} e^{-\int_{x_A}^{x_B} \phi_t(x'')dx''} \epsilon(x')dx' \tag{3.1}$$

para la cual, los coeficientes de extinción (opacidades) $e^{-\int_{x}^{x'} \phi_t(x'')dx''}$ y los colores $\epsilon(x')$ son compuestos para dar el resultado final del proceso.

En la propuesta de Westermann se presenta un enfoque en el cual, por medio del uso de texturas bidimensionales y una serie de pruebas que provee OpenGL, se descartan algunos intervalos de la ecuación de VR pues no tendrán efecto en el resultado final, reduciendo la cantidad de fragmentos a escribir en el framebuffer y así acelerando el proceso de ray casting de la forma que se explicará a continuación.

Para explicar el funcionamiento de esta técnica se toma uno de los rayos que son lanzados hacia el volumen, con inicio en A y fin en B. En este caso particular, se tiene que la representación gráfica de ese rayo viene dada por la ecuación (3.1).

Suponiendo ahora que el volumen no está presente en toda la trayectoria del rayo, sino que comienza en una posición A' y termina en B', la ecuación (3.1) puede expresarse de la siguiente forma:

$$
\begin{aligned}
L(x) &= \int_{x_A}^{x'_A} e^{-\int_{x_A}^{x_{A'}} \phi_t(x'')dx''} \epsilon(x')dx' + \\
&\int_{x'_A}^{x'_B} e^{-\int_{x_{A'}}^{x_{B'}} \phi_t(x'')dx''} \epsilon(x')dx' + \\
&\int_{x'_B}^{x_B} e^{-\int_{x_B}^{x_{B'}} \phi_t(x'')dx''} \epsilon(x')dx'
\end{aligned} \tag{3.2}
$$

Dado que el trazo entre A y A' así como el trazo entre B y B' es vacío, se tiene que su contribución numérica va a ser despreciable. Por lo que (3.2) queda reducida a:

$$
L(x) = \int_{x'_A}^{x'_B} e^{-\int_{x}^{x'} \phi_t(x'')dx''} \epsilon(x')dx' \tag{3.3}
$$

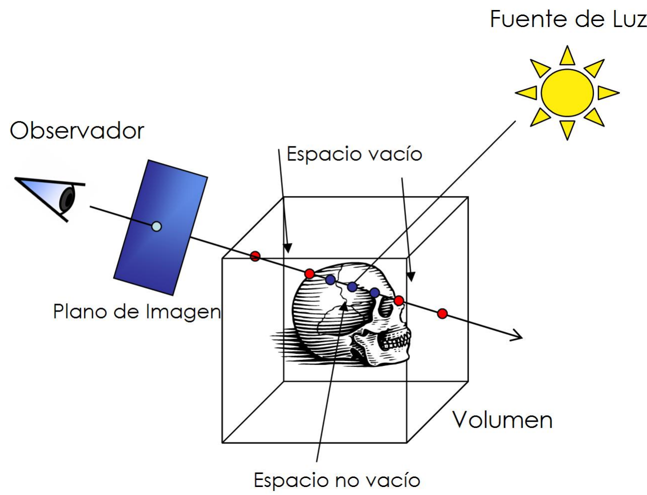

Figura 3.1: Principio de saltos de espacios vacíos de [WS01].

El proceso se conoce mejor como una técnica de preintegración, esto es, una manera de resolver trozos de la integral de VR sin necesidad de hacer la evaluación exhaustiva. Para este caso consiste en hacer salto sobre los espacios vacíos del volumen, proceso conocido en inglés como *empty space skipping* y puede apreciarse gráficamente en la figura 3.1.

3.3.1. Descripcion del Algoritmo

Carga de datos en memoria e iluminación del volumen

1. Se calcula el gradiente de cada elemento del volumen por el método de diferencias centrales.
2. En base a las variables del modelo de iluminación se hace el proceso de iluminación y sombreado del cubo de datos y se almacena en memoria principal.

Creación de texturas y carga en la tarjeta gráfica

1. Para cada eje X, Y y Z:
    a) Para cada plano paralelo a un eje:
        1) Crear una textura bidimensional de ese plano.
        2) Cargar dicha textura en la memoria de la tarjeta gráfica.
        3) Crear un *display list* para cada set de texturas, *front-to-back* y *back-to-front*

Render de textura

1. Determinar el eje del objeto cuyo ángulo sea el menor al plano de proyección (X, Y o Z).
2. Del conjunto correspondiente al eje seleccionado en el paso anterior, determinar cuál es la textura que está más lejana a la cámara.
3. Rotar la cámara alrededor del objeto y dibujar el conjunto de texturas correspondientes al eje escogido en el paso 1, comenzando por la textura escogida en el paso 2.

Render usando Ray Casting

1. Usando las pruebas de alpha y profundidad, se obtienen los puntos más cercanos del plano de proyección que se van a dibujar, así como los más lejanos y se forman tuplas de la forma $((X_{near}, Y_{near}, Z_{near}), (X_{far}, Y_{far}, Z_{far}))$, para las cuales si tomamos el segmento de recta que

forman los dos puntos *near* y *far*, éste es perpendicular al plano de proyección de la imagen final.

2. Para cada una de las tuplas, hacer el trazado del rayo entre esos puntos, la composición, y como paso final el color dejado por la composición se dibuja en el framebuffer.

3.3.2. Detalles de Implementación

Carga de datos en memoria e iluminación del volumen

En esta implementación se tomaron las subrutinas de carga de archivos de la implementación del programa de texturas 2D comentada en el capítulo 2. Con esto se tiene el volumen de opacidad cargado en memoria y a partir de esos datos, se construye el volumen de normales para proceder a calcular la iluminación y sombreado del volumen. Los modelos utilizados para este propósito fueron los mismos de la implementación del ray caster, explicados con detalle en el capítulo 2.

Render acelerado para la previsualización

La implementación del render utilizando la tarjeta gráfica emplea las mismas técnicas de creación de texturas, display lists y visualización del volumen que en la implementación de texturas bidimensionales comentada en el capítulo 2.

Render usando ray casting

Para efectuar el ray casting se hace el salto de espacios vacíos obteniendo para cada rayo las posiciones donde comienza y donde termina la intercepción del rayo con el volumen, utilizando dos pasadas del volumen en OpenGL. Luego se aplica ray casting, obteniendo las muestras solamente entre los puntos obtenidos en el paso anterior.

La generación de los puntos que contienen realmente al volumen se hace mediante la combinación de dos pruebas (*tests*) que provee OpenGL: la prueba de alpha (*alpha test*) y la prueba de profundidad (*depth test*).

La prueba de alpha consiste en comparar el componente alpha con un valor referencial y según el criterio especificado a OpenGL, se descarta o se acepta ese fragmento dentro del framebuffer. Los criterios que se pueden especificar en OpenGL se enumeran en la tabla 3.1.

Criterio	Descripción
GL_NEVER	los fragmentos nunca son dibujados
GL_LESS	los fragmentos con valor de *alpha* menor al valor de referencia son dibujados
GL_EQUAL	los fragmentos con valor de *alpha* igual al valor de referencia son dibujados
GL_LEQUAL	los fragmentos con valor de *alpha* igual o menor al valor de referencia son dibujados
GL_GREATER	los fragmentos con valor de *alpha* mayor al valor de referencia son dibujados
GL_NOTEQUAL	los fragmentos con valor de *alpha* distinto al valor de referencia son dibujados
GL_GEQUAL	los fragmentos con valor de *alpha* igual o mayor al valor de referencia son dibujados
GL_ALWAYS	los fragmentos siempre son dibujados.

Tabla 3.1: Criterios de la prueba de alpha en OpenGL

Al usar el alpha test con la función `GL_GREATER` para valores mayores a cero, sólo se aceptarán en la escena aquellos fragmentos que verdaderamente van a modificar el framebuffer y que por consiguiente, son visibles al usuario.

La segunda prueba se refiere a la de profundidad o *depth test*. Un contexto de OpenGL puede almacenar en un *buffer*, llamado buffer de profundidad, la distancia del fragmento dibujado al plano de proyección para cada fragmento del framebuffer en coordenadas normalizadas del dispositivo. La prueba de profundidad compara el valor de los subsiguientes fragmentos a dibujar con el valor del buffer de profundidad en la misma posición, y acepta el fragmento dependiendo de cómo se establezca el criterio, que puede ser cualquiera de los explicados anteriormente. Las coordenadas normalizadas de dispositivo están en el rango [0,1]. Puede establecerse si al aceptarse un fragmento, se sobreescribe o no el valor correspondiente en el buffer de profundidad.

El objetivo de la prueba de alpha es descartar los fragmentos totalmente transparentes, de manera de escribir en el buffer de profundidad la distancia de los fragmentos que sí contribuyen a la imagen final. En la primera pasada se inicializa el buffer de profundidad con el mayor valor del mismo, equivalente a 1,0, y se establece `GL_LESS` como el criterio de escritura al buffer. De esta

modo, se obtienen las distancias de los fragmentos con contribución significativa más cercanos al plano de proyección. En la segunda pasada se inicializa el buffer con el menor valor, en este caso 0,0, y el criterio se establece en `GL_GREATER`, obteniéndose las distancias de los fragmentos significativos más lejanos del plano de proyección.

Dado que el buffer de profundidad usa coordenadas normalizadas del dispositivo, y las coordenadas para el ray casting son locales al volumen, deben transformarse las coordenadas del primero al segundo, mediante la transformación a coordenadas del mundo. Todo el procedimiento queda reducido a procesar los puntos de la pantalla (X_s, Y_s, Z_s) con los siguientes cálculos:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = (PM)^{-1} V \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} \tag{3.4}$$

donde el arreglo de valores (x, y, z, w) representan las coordenadas homogéneas del mundo. P, M y V son respectivamente la matriz de proyección, la matriz de *modelview* y finalmente la transformación del *viewport*. Todo el procedimiento numérico asociado lo hace una función de GLUT, llamada `gluUnProject/9` que recibe las matrices y las coordenadas de la pantalla.

Cuando se tienen las tuplas ordenadas de puntos cercanos y lejanos, se procede a hacer el ray casting. En este paso, ya el punto de visión no es ortogonal sobre una de las caras del cubo de datos y es necesario trazar el rayo. Para hacerlo, tenemos un par de puntos en el espacio P y Q, donde P representa el punto de origen del rayo – el punto cercano al observador – y Q el fin del rayo – el punto alejado al observador – y dado que queremos muestrear sobre estos puntos podemos recurrir a las siguientes ecuaciones paramétricas:

$$X(t) = P_x + (Q_x - P_x)t$$

$$Y(y) = P_y + (Q_y - P_y)t$$

$$Z(t) = P_z + (Q_z - P_z)t$$

Cada una de las ecuaciones anteriores define el valor de X, Y e Z en la trayectoria del rayo, mediante una aproximación paramétrica en $t \in [0, 1]$. Los valores de incremento en t vienen dados por el número de muestras n que se quiere en cada rayo, por ejemplo puede ser $\frac{1}{n}$.

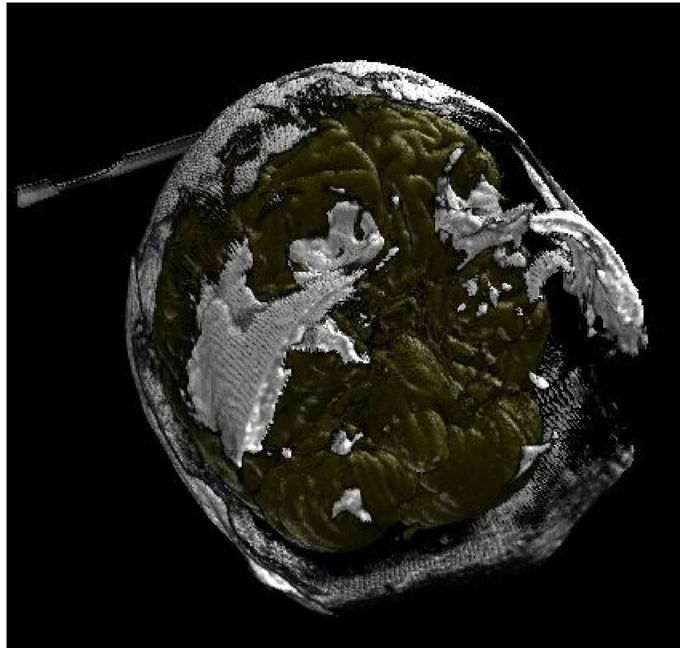

Figura 3.2: Visualización empleando la técnica de Westermann & Sevenich

Con todos los datos necesarios se procede finalmente a realizar para cada rayo la composición, y se dibuja en la pantalla para cada *pixel* el valor final del color dejado por la composición. En la figura 3.2 se muestra una visualización de múltiples clases que hace uso de las técnicas de salto de espacios vacíos y ray casting con proyección oblicua presentadas en esta sección.

3.4. Técnica de Engel, Kraus & Ertl

Una de las fases críticas en VR es la de clasificación. Como se comentó anteriormente en ella se asignan valores de color y opacidad para cada uno de los elementos del volumen dada una función de transferencia.

Engel, Kraus & Ertl proponen el uso de memoria de texturas con dos fines: el primero es evaluar la función de clasificación y guardar los resultados sobre una textura bidimensional dependiente. El segundo consiste en evaluar cada voxel almacenado en textura contra la textura de clasificación, para obtener los valores finales en la etapa de rasterización [EKE01].

En el último caso se presentan dos opciones, un proceso de pre-clasificación o de post-clasificación. Para la pre-clasificación, dado el volumen de datos, se toma la muestra de un voxel y se le aplica la función de transferencia. Hecho esto para todas las muestras del volumen se procede a interpolar los valores de todo el volumen y se le aplica el proceso de shading. En la post-clasificación, primero se interpolan las muestras del volumen para después aplicar la función de transferencia y el shading.

Según Engel, Kraus & Ertl, la pre- y post-clasificación generan resultados distintos cuando la interpolación no es conmutativa con la función de transferencia original. Al aplicar post-clasificación, las imágenes tienden a ser difusas, mientras que para la pre-clasificación están muy bien definidas.

Al tener la función de transferencia ya calculada, el proceso de clasificación se limita a obtener el valor directamente en la etapa de rasterización usando un acceso a textura. Conocido el valor del color y la opacidad para todas las muestras de las clases del volumen, es posible aplicar un modelo de iluminación para agregar calidad a la visualización. Finalizado esto se disminuye el número de planos que componen el volumen, reduciendo el número de geometrías a dibujar y preintegrando los planos del volumen correspondientes a esas geometrías. Esta preintegración consiste en interpolar dos planos del volumen para generar otro intermedio llamado *slab*.

En la fase de iluminación, Engel, Kraus & Ertl proponen el uso de un modelo que considera los componentes difusos y especulares de la iluminación. Para evaluar este modelo se requiere el valor

aproximado de la normal en cada elemento del volumen. Estas normales se guardan en una textura que contiene en sus canales RGB, sus componentes en el rango [0,1] para luego en el procesador de fragmentos expandirla y usarla.

3.4.1. Descripción del Algoritmo

En la implementación realizada no se toma en cuenta la fase de pre y post clasificación de los datos de entrada mencionada anteriormente, dado que se emplearon datos provenientes de [MNI] los cuales ya vienen pre-clasificados.

Carga de datos en memoria

1. Se obtienen los valores de opacidad para la clase del volumen a visualizar.

2. Se calcula el gradiente de cada elemento del volumen por el método de diferencias centrales.

Creación de texturas y carga en la tarjeta gráfica

1. Reservar un espacio en memoria cuyas dimensiones sean las potencias de dos más pequeñas que puedan contener el cubo de datos de 4 componentes y de tipo `float`.

2. Rellenar el espacio con ceros.

3. Cargar el cubo de datos dentro del espacio. Los datos en el espacio ocupan los mismos lugares que en el cubo, en las componentes RGB se cargan los valores del gradiente normalizado, y en el componente alpha se carga el valor de opacidad obtenido en la carga inicial para esa clase.

4. Cargar el espacio reservado con los datos como una textura tridimensional a la tarjeta gráfica.

Render con geometrías orientadas al objeto

1. Determinar el eje del objeto que está más paralelo al eje de proyección. (X, Y o Z)

2. Del conjunto correspondiente al eje escogido en el paso anterior, determinar cuál es la textura que está más lejana a la cámara.

3. Rotar la cámara alrededor del objeto y dibujar el conjunto de texturas correspondientes al eje escogido en el paso 1, comenzando por la textura escogida en el paso 2.

4. Durante la rasterización un programa en Cg se encarga de hacer shading del volumen y hacer el blending de los valores finales.

3.4.2. Detalles de Implementación

Carga de datos en memoria

Para esta implementación se aprovecharon las rutinas de carga de archivos de la implementación del programa de texturas 3D comentado en el capítulo 2. Con los datos cargados en memoria, se calculó el gradiente para cada uno de los elementos del volumen.

Creación de texturas y carga en la tarjeta gráfica

Ya obtenido el gradiente, se llenó un cubo de $256^3 \times 4$ voxels. En las primeras tres componentes va el valor del gradiente[1] en el rango [0,1] de acuerdo a la especificación de `glTexImage3D/10` para el tipo `GL_FLOAT` – del mismo modo que funciona para texturas de 1 y 2 dimensiones, especificado en [Boa92] y [BSW+92] – y en el canal alpha se colocó el valor de opacidad también en el rango [0,1]; posteriormente el cubo entero se carga en la tarjeta gráfica como una textura 3D.

Render con geometrías orientadas al objeto

Se seleccionó la textura cargada a memoria de la tarjeta gráfica como textura activa, y se hizo render del conjunto de primitivas cuyo ángulo era el menor con respecto al plano de proyección.

[1]Dado que el gradiente puede tener valor negativo, es indispensable conservar los signos de las componentes en la memoria de textura. Para esta implementación, se normaliza para que quede en el rango [-1,1] a través de un escalamiento y luego se lleva a una forma entre [0,1] con la ecuación $\nabla f(x) = \frac{\nabla f(x)+1,0f}{2}$ por medio de una constante para desplazarlo.

Se empleó un fragment shader para hacer los cálculos de iluminación en el procesador de la tarjeta gráfica tomando en consideración las componentes ambientales, difusas y especulares como una modificación del modelo que propone el trabajo original y puede observarse en el código 5.

Código 5 Programa de fragmentos – Composición de 4 clases simultáneamente

```
1: float3 expand(float3 v) { return (v - 0.5) * 2; }
2: float4 main(
   float4 texPos0 : TEXCOORD0,
   uniform float3 globalAmbient,
   uniform float3 lightColor,
   uniform float3 lightPosition,
   uniform float3 eyePosition,
   uniform float3 Ka,
   uniform float3 Kd,
   uniform float3 Ks,
   uniform float shininess,
   uniform sampler3D samplerTex0 : TEXUNIT0) : COLOR {
3: float4 OUT = float4(1.0f,1.0f,1.0f,1.0f);
4: float4 texvalue = tex3D(samplerTex0, texPos0);
5: float3 normal = expand(texvalue.rgb);
6: OUT.a = texvalue.a;
7: if(OUT.a > 0) {
8: float3 ambient = lightColor * Ka;
9: float3 halfway = normalize(lightPosition - eyePosition);
10: float3 diffuse = max(dot(lightPosition, normal),0) * Kd * lightColor;
11: float3 specular = pow(dot(halfway, normal), shininess) * Ks;
12: OUT.rgb = ambient + diffuse + specular;}
13: return OUT;
14: }
```

3.4.3. Resultados Obtenidos

Con la función de iluminación del trabajo original de Engel, Kraus & Ertl, los resultados iniciales alcanzan un nivel de calidad visual aceptable, mas no excelente. Consideramos que el modelo de iluminación mejora al implementarse aspectos del modelo de Phong, como la contribución ambiental. Los resultados para el render de un volumen correspondiente a músculo y piel se muestran en la figura 3.3.

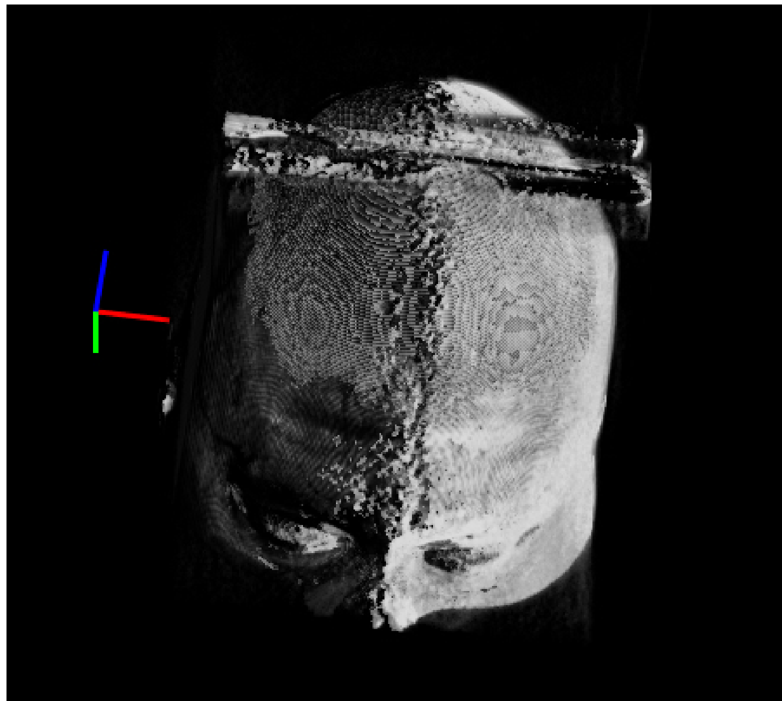

Figura 3.3: Visualización empleando la técnica de Engel, Kraus & Ertl

Capítulo 4

Resultados

4.1. Descripción del Equipo Usado

En los experimentos realizados para la ejecución de las implementaciones hechas, se dispuso de un equipo con las siguientes características:

- Procesador AMD Athlon® XP a 2083MHz.
- Memoria de Sistema 512 MB.
- Tarjeta aceleradora de gráficos Nvidia GeForce FX5600 Ultra con 128 MB de memoria.
- Espacio de intercambio 1024 MB.
- Sistema Operativo Microsoft Windows XP Professional con Service Pack 2.

4.2. Parámetros a Medir

Los parámetros medidos en estas pruebas fueron:

Velocidad de render en fotogramas (*frames*) por segundo (fps) Un contador de fotogramas se incrementa en las implementaciones cada vez que se completa una pasada. Cada

segundo se actualiza el contador en la pantalla y se reestablece a cero.

Zoom El zoom es un factor que multiplica el tamaño de las primitivas que representan el volumen de datos con respecto al tamaño del cubo de proyección. Esto afecta el tamaño del volumen con relación a la ventana de visualización.

Densidad de ocupación del volumen Este número indica la relación entre la cantidad de voxels con transparencia mayor que cero y la cantidad de voxels del volumen de datos.

***Bounding box* del volumen** Indica el nivel de dispersión de los datos dentro del volumen de datos, define el tamaño del cubo mínimo que contiene a los datos de entrada y se compara al tamaño del cubo original.

4.3. Metodología de las Pruebas

Los resultados de las pruebas donde el volumen aparece en movimiento, se obtuvieron haciendo una ejecución de cada implementación, rotando el volumen constantemente en su eje Y. Se registró para cada prueba, la cantidad de fotogramas por segundo que muestra el programa durante una rotación completa del volumen. Para los resultados de las pruebas con el volumen estático se dejó el volumen en el mismo estado como inicia la implementación, anotando en cada caso la cantidad de fotogramas que muestra el programa. Por lo general, cuando el programa no muestra un único valor de fotogramas, varía de 1 a 3 fotogramas, haciéndose notar esto en los resultados obtenidos. Todas las pruebas fueron ejecutadas en una ventana de tamaño de 512×512 pixels.

4.4. Descripción de los Datos de Entrada en las Pruebas

Para las clases usadas en la ejecución de las pruebas se tienen los siguientes datos:

4.4.1. Densidad de Voxels

La densidad revela la cantidad de voxels que contribuyen realmente a la imagen final. En la tabla 4.1 se tienen los datos para cada clase. El porcentaje usado corresponde a la relación entre el número de voxels con alpha mayor que cero y el número total de voxels del volumen, en este caso éste corresponde a $181 \times 217 \times 181 = 7{,}109{,}137$ voxels.

Clase	Cantidad (voxels)	Densidad (%)
Grasa	288.726	4.061
Hueso	494.039	6.949
Materia Gris	1.268.754	17.847
Músculo y Piel	3.064.450	43.106

Tabla 4.1: Densidad del volumen total de los datos de prueba

4.4.2. Dispersión de Voxels

Con este valor se aprecia qué tan dispersos están los datos a procesar para generar una imagen final. En la tabla 4.2 se tienen los datos para cada clase. La relación que se muestra es la del volumen del bounding box mínimo que contiene a los datos entre el volumen de los datos por completo, el cual corresponde a $181 \times 217 \times 181 = 7{,}109{,}137$ unidades de volumen.

Clase	Tamaño del cubo mínimo (voxels)	Dispersión (%)
Grasa	6.467.580	90.976
Hueso	4.987.800	70.160
Materia Gris	3.747.744	52.717
Músculo y Piel	6.998.400	98.442

Tabla 4.2: Dispersión sobre el volumen total de los datos de prueba

4.5. VR Directo Usando Mecanismo de Texturas

A continuación se presentan los resultados de las pruebas realizadas con los algoritmos que utilizan la tarjeta gráfica. Cada tabla se definió de la siguiente manera:

- Cada ejecución se hizo con un volumen y nivel de zoom especificados en el título.
- La primera columna en la tabla indica las librerías empleadas para realizar el render: "OpenGL" indica que el render se hizo con la función fija de esta librería, y "OpenGL y Cg" indica que se efectuó con un fragment shader.
- La segunda columna indica si se estaba aplicando el alpha test para ese momento. El resto de las columnas indican el número de fotogramas por segundo a la que corre cada algoritmo dados los parámetros especificados en la primera y segunda columna.
- La tercera columna está identificada por la frase "Texturas 2D Objeto", e indica la velocidad en fotogramas por segundo de la ejecución que se hizo de los algoritmos con texturas bidimensionales con planos orientados al objeto.
- La cuarta columna y quinta columna están identificadas respectivamente por las frases "Texturas 3D Objeto" y "Texturas 3D Visión", e indican la velocidad en fotogramas por segundo de la ejecución del algoritmo con texturas tridimensionales con planos orientados al objeto y al plano de proyección.

4.5.1. Volumen con una Sola Clase

Nótese que en general las pruebas indican que cuando se desactiva el alpha test, los algoritmos proveen una cota inferior de desempeño, es decir, todos los algoritmos con alpha test activado generan más fotogramas por segundo comparado con aquellos con alpha test desactivado. Se observa también que la técnica de planos orientados al de proyección ofrece el rendimiento más bajo. Por

último, el rendimiento decrece cuando se utiliza un fragment shader con respecto al uso de la función fija de OpenGL.

En la tabla 4.3 se puede observar que la técnica de planos orientados al objeto con texturas tridimensionales obtiene un rendimiento menor que con texturas bidimensionales.

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	32 - 34	15	4 - 5
OpenGL	No	23 - 24	12 - 13	3
OpenGL y Cg	Sí	29	21 - 23	4
OpenGL y Cg	No	24	19 - 21	3 - 4

Tabla 4.3: Volumen Grasa. Nivel de zoom 1.0. En movimiento.

En la tabla 4.4 se recogieron los resultados de la ejecución visualizando el mismo volumen con un nivel mayor de zoom. En estas resultados se aprecia que el algoritmo con texturas bidimensionales aún supera al algoritmo con texturas tridimensionales, aunque con una menor diferencia con respecto al zoom 1.0, específicamente alrededor de uno o dos cuadros. Esta observación se cumple también para el resto de los volúmenes probados (ver apéndice A).

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	19 - 20	11 - 12	5 - 6
OpenGL	No	8 - 9	7 - 8	4
OpenGL y Cg	Sí	7 - 9	8 - 10	4
OpenGL y Cg	No	7 - 9	7 - 9	3 - 4

Tabla 4.4: Volumen Grasa. Nivel de zoom 2.0. En movimiento. Todas las cifras en fps.

Las tablas 4.5, 4.6, 4.7, 4.8, 4.9 y 4.10 se muestran los mismos algoritmos visualizando los volúmenes de hueso, materia gris y músculo más piel. Estas tablas están organizadas por el nivel de densidad del volumen correspondiente. Se puede observar en estos casos que la diferencia de rendimiento entre la aplicación o no de la prueba alpha se va reduciendo a medida que aumenta la densidad del volumen. La diferencia de rendimiento entre la activación y desactivación de la prueba

de alpha se mantiene cuando se utiliza Cg.

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	31 - 33	15	4
OpenGL	No	25	12 - 13	3
OpenGL y Cg	Sí	22 - 24	22 - 23	4
OpenGL y Cg	No	21 - 22	19 - 21	3 - 4

Tabla 4.5: Volumen Hueso. Nivel de zoom 1.0. En movimiento

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	19 - 20	11 - 12	5
OpenGL	No	9 - 10	8	4
OpenGL y Cg	Sí	8 - 9	8 - 9	4
OpenGL y Cg	No	6 - 8	8	4

Tabla 4.6: Volumen Hueso. Nivel de zoom 2.0. En movimiento

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	30 - 31	15	4
OpenGL	No	24 - 25	12 - 13	3 - 4
OpenGL y Cg	Sí	29	22 - 23	4
OpenGL y Cg	No	24 - 25	20 - 21	3 - 4

Tabla 4.7: Volumen Materia Gris. Nivel de zoom 1.0. En movimiento

En todas las tablas se puede apreciar que la técnica con texturas tridimensionales con los polígonos orientados al plano de proyección posee un rendimiento sustancialmente menor a las misma con planos orientados al objeto. También se puede observar en todos los casos que el rendimiento en los polígonos orientados al plano de proyección mejora cuando el zoom aumenta.

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	17 - 18	11	6
OpenGL	No	9 - 10	7 - 8	4 - 5
OpenGL y Cg	Sí	8 - 9	8 - 9	4
OpenGL y Cg	No	8 - 9	7 - 9	4

Tabla 4.8: Volumen Materia Gris. Nivel de zoom 2.0. En movimiento

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	28 - 29	13 - 14	4
OpenGL	No	24 - 27	12 - 13	3
OpenGL y Cg	Sí	25 - 26	21 - 22	4
OpenGL y Cg	No	24	20 - 21	3 - 4

Tabla 4.9: Volumen Músculo y Piel. Nivel de zoom 1.0. En movimiento

Render	Alpha Test	Texturas 2D Objeto	Texturas 3D Objeto	Texturas 3D Visión
OpenGL	Sí	12 - 13	9	5
OpenGL	No	9 - 10	7 - 8	4
OpenGL y Cg	Sí	8 - 9	8 - 9	4
OpenGL y Cg	No	7 - 9	7 - 8	4

Tabla 4.10: Volumen Músculo y Piel. Nivel de zoom 2.0. En movimiento

4.5.2. Múltiples Volúmenes

Al tomar en consideración el render de varias clases simultáneamente, se registró la cantidad de cuadros por segundo en los casos de 1, 2, 3 y 4 clases. Se recuerda que esta implementación usa varias unidades de texturas al mismo tiempo. Se puede notar cómo a medida que aumenta el número de clases, el rendimiento de la visualización decrece.

Zoom	Alpha Test	1 Clase	2 Clases	3 Clases	4 Clases
1.0	Sí	31	16	9	7
1.0	No	30	13	9	7
2.0	Sí	11	5	3	2
2.0	No	9	4	3	2

Tabla 4.11: Múltiples volúmenes

4.6. VR Directo Usando Técnicas de Preintegración

4.6.1. Westermann & Sevenich

Para estas pruebas, se midió el tiempo en el que se hace el render detallado de un volumen utilizando ray casting. Este tiempo fue registrado en microsegundos ($1\mu s = 10^{-6}s$). Para ello se estableció dos niveles de zoom. Un caso se ejecutó usando una implementación ingenua de un ray caster – que lanza todos los rayos y compone a lo largo de toda su extensión – y el otro con el método de Westermann & Sevenich de salto de espacios vacíos. Se observa que la optimización de Westermann & Sevenich reduce el tiempo de ray casting a casi un cuarto del tiempo que toma la implementación ingenua.

Método	Grasa	Hueso	Materia Gris	Músculo y Hueso
Empty Space Skipping	1.812.500	1.848.958	1.766.041	3.703.215
Naive Ray cast	4.250.000	4.208.333	5.364.583	4.395.833

Tabla 4.12: Westermann & Sevenich. Tiempo de Ray cast - Zoom 1.0

Método	Grasa	Hueso	Materia Gris	Músculo y Hueso
Empty Space Skipping	4.859.375	5.401.041	4.505.208	12.526.041
Naive Ray cast	13.989.583	14.562.916	15.510.416	15.187.500

Tabla 4.13: Westermann & Sevenich. Tiempo de Ray cast - Zoom 2.0

4.6.2. Engel, Kraus & Ertl

Para las ejecuciones del software que implementa la técnica de Engel, Kraus & Ertl, los resultados se encuentran resumidos en la tabla 4.14. Las columnas de la tabla representan: el nivel de zoom con el que se visualiza el volumen, cómo se encuentra el alpha test para el momento del render, y cada una de las clases evaluadas durante la ejecución.

Esta prueba mide la cantidad de fotogramas por segundo para cada una de las clases mostradas, haciendo iluminación en el GPU. Nótese que a medida que la dispersión de los volúmenes aumenta, disminuye en un cuadro el desempeño del rendering.

Zoom	Alpha Test	Grasa	Hueso	Materia Gris	Músculo y Piel
1.0	Sí	5	5	4	4
1.0	No	5	4	4	4
2.0	Sí	2	2	2	2
2.0	No	2	2	1	1

Tabla 4.14: Engel, Kraus & Ertl - Resultados

4.7. Análisis de los Resultados

4.7.1. Técnicas Basadas en Texturas

Alpha Test

Las diferencias en el desempeño según la aplicación o no del alpha test se explica por el funcionamiento de ésta. La prueba de alpha está determinada para que descarte aquellos fragmentos cuyo componente alpha sea cero. Ya que gran parte de los voxels en los volúmenes de opacidad tienen valor cero, el alpha test ayuda notoriamente a aumentar la velocidad de render al no dibujar estos fragmentos en el framebuffer. La calidad del resultado final no es afectada, dado que un fragmento con alpha cero no altera el color con la función de blending escogida para el algoritmo.

Diferencias en la Velocidad de Acceso a las Texturas

Las diferencias en el desempeño de las aplicaciones según el tipo de textura utilizada indica que los accesos a texturas tridimensionales son más lentos que los accesos a texturas bidimensionales, lo cual se debe a que el acceso a las primeras es irregular con respecto a las segundas, por el hecho de estar guardados una gran cantidad de datos contínuos en un bloque de memoria. Esto fue mencionado por Engel, Kraus & Ertl [EKE01].

El mayor valor práctico de las texturas bidimensionales ha resultado que la investigación se concentre en ese tipo de texturas. Finalmente, la mayor cantidad de cálculos para aplicar las tridimensionales es otro factor que explica la disminución del desempeño del render. Un ejemplo de esto es que una interpolación trilineal necesita acceder ocho veces a la textura mientras que la interpolación bilineal sólo accede cuatro veces.

La técnica de planos orientados a la visión ofrece el menor rendimiento debido a que no se hizo una cantidad suficiente de optimizaciones. El algoritmo hecho para estas pruebas posee una cantidad fija de planos que cubren la totalidad del volumen de datos. Para obtener una calidad de imagen aceptable, es necesario que los planos intercepten todos los voxels posibles cuando el tamaño del volumen rotado en los planos sea máximo. Para lograr esto, se debía colocar como mínimo un numero de planos igual a la cantidad de voxels que caben en la diagonal del volumen de datos, esto es, $\sqrt{181^2 + 217^2 + 181^2} = 336$ planos. Estas pruebas se hicieron con 400 planos, una configuración que aumenta la calidad de la imagen obtenida, pero reduce en gran medida la velocidad por la cantidad de fragmentos generados, la cantidad de accesos a memoria de la textura, y la cantidad de blending asociado a los fragmentos aceptados. La cantidad de accesos a memoria hace que la ganancia en las otras dos técnicas por la aplicación del alpha test sea despreciable.

Influencia del Zoom

De los resultados, es notorio que un mayor nivel de zoom en los volúmenes disminuye notablemente el número de fotogramas por segundo. Un mayor tamaño de las primitivas con respecto al

cubo de proyección genera durante la rasterización, un mayor número de fragmentos. Esto resulta en un mayor número de operaciones y la consecuente disminución del rendimiento. Si los resultados con zoom igual a 2.0 se comparan con un zoom de 1.0 se puede observar que se mantienen las mismas tendencias en el rendimiento.

La disminución del rendimiento con el zoom indica que el algoritmo está limitado principalmente por el *fill rate*. Para el chip GeForce FX que se utilizó en las pruebas, nVidia en sus especificaciones técnicas documenta que el fill rate de éste es 1.6 billones de texels por segundo[1]. No se pudo realizar comparaciones con otras arquitecturas.

En el caso de los polígonos orientados al plano de proyección, se observa que al aumentar el zoom mejora el desempeño de la aplicación. Debido al tamaño de las primitivas en esta técnica, muchos fragmentos con opacidad transparente se generan a la hora de rasterizar. El aumento del zoom hace que muchos fragmentos generados caigan fuera del cubo de visualización, una operación que se hace en una etapa temprana del pipeline gráfico, por lo que sólo los fragmentos adentro de este cubo son los que generan operaciones de acceso de memoria y alpha test en el GPU.

Densidad de los Datos

La densidad de los volúmenes afecta directamente el rendimiento de los algoritmos cuando se utiliza el alpha test, no así cuando se desactiva éste. Mientras que el desempeño entre los volúmenes cuando no se utiliza la prueba se mantiene sin variaciones, en el caso contrario las variaciones son notorias, y la diferencias entre los mismos volúmenes cuando se activa o desactiva la prueba alpha son inversamente proporcionales a la densidad. Esto se debe a la cantidad de fragmentos que no se dibujan cuando se aplica la prueba, dejando en evidencia que evitar la operación de blending sobre esos fragmento mejora notablemente el rendimiento del algoritmo.

[1] Fuente: nVidia Corp., `http://www.nvidia.com/page/fx_5600.html`

4.7.2. Técnica de Westermann & Sevenich

Salto de Espacios Vacíos

En los resultados se nota una gran diferencia en los tiempos del ray casting cuando se aplica la prueba alpha y cuando se deja de aplicar. Cuando se aplica la prueba, se dejan de dibujar los fragmentos con alpha cero, por lo que se puede determinar los puntos dónde un rayo trazado entraría en contacto por primera vez con el material y dónde lo dejaría por última. Las diferencias observadas corresponden a una diferencia en el orden de 3 a 4 veces más rápido usando la prueba alpha y de profundidad para determinar estos puntos. Estos resultados pueden compararse con los expuestos por Westermann & Sevenich, y demuestra que es una alternativa viable para una implementación híbrida de un ray caster ya que la detección de los puntos no representa una sobrecarga notable en el algoritmo.

Dispersión de los Datos

La dispersión de los datos, como comenta el trabajo original, sí afecta el rendimiento del proceso de ray casting. La técnica salta solamente los espacíos vacíos externos, por lo que si la dispersión del volumen es alta, la longitud de los rayos trazados con el salto de espacio será similar a la longitud del rayo sin esta optimización. Estos resultados se aprecian en gran medida en las clases “Grasa” y “Músculo más Piel”, que son los volúmenes con mayor índice de dispersión en las pruebas.

4.7.3. Técnica de Engel, Kraus & Ertl

Reducción del Número de Planos del Volumen

Como se evidencia en los resultados correspondientes a la técnica de Engel, Kraus & Ertl, no se pudieron reproducir los resultados mostrados en el trabajo original, a pesar de disponer de un hardware superior. La posible interpretación de esta observación es que no se realizó la reducción del número de planos que componen el volumen, porque la aproximación que ellos realizan hacia

la interpolación de dos texturas no es simple. En base a esto, se tomó la decisión de hacer una implementación basada en texturas tridimensionales con los volúmenes de datos sin modificar ningún aspecto sobre ellas.

Iluminación Sobre el GPU Usando Cg

La capacidad de cómputo de la tarjetas gráficas actuales queda demostrada en estos resultados ya que la iluminación y el sombreado se hacen en el procesador de ésta, alcanzándose velocidades de 4 a 6 fotogramas por segundo, y haciendo posible la iluminación dinámica sobre el volumen. Esto representa una ventaja con respecto a la técnica de Westermann, la cual hace un cálculo de iluminación único y en la fase inicial del algoritmo, la cual se ejecuta en el CPU.

En el algoritmo implementado, el cálculo del gradiente para cada elemento del volumen es calculado previamente y almacenado en una forma comprimida dentro de una textura RGB. Al momento de hacer el render, un programa de fragmentos en Cg se encarga para cada elemento del volumen, calcular su valor de color final en base a parámetros uniformes que pueden ser modificados interactivamente desde la aplicación en C. Tomando en consideración esto, todo el proceso en CPU puede tomar una notable cantidad de tiempo de ejecución por la elevada cantidad de operaciones de multiplicación, suma y exponenciación de vectores.

Capítulo 5

Conclusiones

5.1. Trabajo Realizado

En este trabajo de grado se estudiaron e implementaron diversos algoritmos de VR basados en manejo de texturas y en técnicas híbridas. En particular, se diseñó y desarrolló software de VR con texturas bidimensionales y tridimensionales, empleando técnicas propuestas por Westermann & Sevenich, así como por Engel, Kraus & Ertl. Para las técnicas que utilizan los procesadores de las tarjetas gráficas se aprovechó la programabilidad de éstos con el lenguaje Cg. Entre estas técnicas se hicieron comparaciones de desempeño y calidad de imagen generada.

Los datos de prueba fueron obtenidos de un simulador de imágenes de resonancia magnética desarrollado en el McConnell Brain Imaging Centre ubicado en Montreal – Canadá. Este simulador permite la modificación de los parámetros de la función de transferencia para generar clases bien diferenciadas correspondientes a tejido cerebral, por consiguiente, no fue imprescindible profundizar en el proceso de clasificación de los datos.

Una de las mejores aproximaciones a VR se logra mediante ray casting. Como lo indica su nombre en inglés, la técnica consiste en lanzar rayos sobre el volumen para lograr su cometido. Cada rayo lanzado permite obtener muestras del volumen espaciadas regularmente sobre la trayectoria del

mismo. Para esas muestras se puede aplicar un operador de composición para generar el color final. En la práctica, cada rayo lanzado corresponde a un pixel de la pantalla, el cual penetra el volumen de datos, hace el muestreo espaciado y la correspondiente composición.

Los algoritmos implementados en este trabajo permiten acelerar la evaluación de la integral de VR, empleando hardware gráfico de corriente principal, ya sea para producir la imagen o para reducir el trabajo a los algoritmos implementados en el CPU. Estos algoritmos toman como entrada, el volumen de datos cargándolo a la memoria de las tarjetas gráficas como texturas, que ulteriormente son asignadas a unas primitivas de dibujo. La tarjeta se encarga de hacer la interpolación bilineal o trilineal de las texturas (según sea el caso de texturas 2D o 3D respectivamente), y hacer el proceso de blending con el framebuffer para simular el operador de composición del color final.

La técnica propuesta por Westermann & Sevenich no utiliza el framebuffer sino otra información producida por la tarjeta gráfica. En particular, emplea la información que provee el buffer de profundidad para saltar los espacios vacíos externos al volumen. Así, el algoritmo de ray casting solamente obtiene muestras de material no transparente del volumen. Por lo tanto, hay menos procesamiento de los datos del volumen, acelerando el algoritmo de ray casting.

La técnica propuesta por Engel, Kraus & Ertl indica las posibilidades de evaluación y modificación de la función de transferencia en tiempo real por medio del uso de texturas. La idea se fundamenta en almacenar el resultado de la función de transferencia evaluada para un rango de valores de entrada y cargarlo como una textura bidimensional. Con ello, dado un volumen de datos, es posible hacer la clasificación de éste y obtener el color primario y la opacidad para los voxels que lo componen. Concluyen el proceso con la iluminación de las muestras utilizando un enfoque que toma las contribuciones difusas y especulares de la iluminación.

5.2. Resultados Obtenidos

Dada la experimentación se evidencia que la técnica de ray casting mediante software ofrece siempre un nivel alto de calidad de imagen debido a que es un algoritmo enfocado en la imagen. Las optimizaciones que se hacen en el trabajo de Westermann & Sevenich reducen sustancialmente el tiempo de ejecución del ray casting en software para una gran variedad de casos, sin sacrificar la calidad de la imagen. Sin embargo, la velocidad de esta solución siempre está limitada por la velocidad del CPU, por lo que todavía se está lejos de alcanzar velocidades de tiempo real.

En general, los algoritmos que utilizan texturas bidimensionales y tridimensionales alcanzan velocidades interactivas y en tiempo real. Se ha comprobado que el alpha test proporcionado por OpenGL efectivamente reduce la cantidad de operaciones de escritura al framebuffer, ya que se descartan los fragmentos con opacidad igual a cero. Esta reducción se traduce en mejor desempeño, aunque ello depende de la cantidad de voxels que sean totalmente transparentes.

Las texturas tridimensionales ofrecen una calidad de imagen superior a la que ofrecen sus pares bidimensionales. En las primeras, se obtienen muestras de las texturas a partir de interpolaciones trilineales, mientras que en las segundas, las muestras obtenidas provienen de interpolaciones bilineales; ocasionando el efecto de bandas en el render final. Un punto a favor de las texturas tridimensionales viene dado por la sencillez de implementación y la capacidad de visualización de múltiples clases. Esto es posible gracias a las capacidades de múltiples unidades de texturas de las tarjetas gráficas recientes.

En los resultados se encontró que el uso de Cg flexibiliza la programación de los algoritmos implementados, aunque reduce un poco el desempeño de la aplicación. No obstante, Cg permite pensar en optimizaciones como reducción de memoria o post-procesamiento de la visualización sobre el GPU. Además estas técnicas se encuentran limitadas principalmente por el fill rate de las tarjetas gráficas, dada la alta cantidad de datos que se almacenan y la rapidez de acceso a la memoria de texturas en las mismas. Esto significa que los volúmenes de datos que se pueden cargar

no pueden ser arbitrariamente grandes – el límite de las texturas tridimensionales de las tarjetas utilizadas es de 512^3 voxels. De igual manera, las imágenes producidas deben mantenerse pequeñas si se quiere conservar una velocidad de fotogramas por segundo interactiva.

El trabajo de Engel, Kraus & Ertl propone una técnica que puede ser implementada en su totalidad en el procesador de la tarjeta gráfica. Nuestra implementación no reprodujo satisfactoriamente la que realizaron los autores debido a que no se hizo la reducción de primitivas a dibujar. La clasificación en nuestro caso ya estaba hecha, por lo que no hubo necesidad de implementar una función de transferencia variable. La implementación demostró qué tan veloces pueden ser los procesadores gráficos para tareas complejas como iluminación y sombreado.

5.3. Trabajo a futuro

Como investigación a futuro presentamos las siguientes propuestas:

1. Afinar y modificar el modelo de iluminación por GPU propuesto por (Engel et al., 2001 [EKE01]) para lograr mejoras en la calidad de la visualización. Esto incluye el procesamiento directo sobre GPU del gradiente para los voxels del volumen así como también la modificación dinámica de los parámetros de iluminación.

2. Uso del GPU para realizar ray casting directamente sobre él. El ideal consiste en hacer una mezcla de los resultados presentados por (Westermann et al., 2001 [WS01]) de salto de espacios vacíos junto a la implementación nativa del ray caster sobre el GPU dada la flexibilidad que ofrece Cg.

3. Uso de texturas comprimidas. En la actualidad existen mecanismos que permiten hacer compresión de los datos que se cargan en la memoria de texturas, uno de los más recientes es un conjunto de algoritmos conocidos como S3TC desarrollados por la empresa S3 Graphics Ltd.

4. Implementación de capacidades para el corte de secciones interactivo, permitiendo de este

modo capacidades de visualización interna en los volúmenes.

Bibliografía

[Aco97] Dionisio M. Acosta. Visualization of biomedical volumetric data. Technical report, University of Sussex, October 1997.

[Boa92] OpenGL Architecture Review Board. *OpenGL Reference Manual, Release 1.* Addison-Wesley Publishing Company, first edition, 1992.

[BSW+92] OpenGL Architecture Review Board, David Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide, Release 1.* Addison-Wesley Publishing Company, first edition, 1992.

[CKE97] C. Cocosco, K. Kwan, and R. Evans. Brainweb: online interface to a 3D MRI simulated brain database, May 1997.

[CN94] Timothy J. Cullip and Ulrich Neumann. Accelerating volume reconstruction with 3d texture hardware. Technical report, Chapel Hill, NC, USA, 1994.

[CZK+98] D.L. Collins, A.P. Zijdenbos, V. Kollokian, N.J. Kabani J.G. Sled, C.J. Holmes, and A.C. Evans. Design and construction of a realistic digital brain phantom. In *IEEE Transactions on Medical Imaging*, volume 17, pages 463–468, June 1998.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. *Eurographics / SIGGRAPH Workshop on Graphics Hardware*, 2001.

[FDF+94] James D. Foley, Andries Van Dam, Steven Feiner, John Hughes, and Richard Phillips. *Introduction to Computer Graphics*. Addison-Wesley Publishing Company, first edition, 1994.

[IKLH04] Milan Ikits, Joe Kniss, Aaron Lefohn, and Charles Hansen. Volume rendering techniques. In Randima Fernando, editor, *GPU Gems*, chapter 39, pages 667–692. Addison Wesley, May 2004.

[KEP96] Remi K.-S. Kwan, Alan C. Evans, and G. Bruce Pike. An extensible MRI simulator for post-processing evaluation. In *VBC*, pages 135–140, 1996.

[KEP99] R.K.-S. Kwan, A.C. Evans, and G.B. Pike. Mri simulation-based evaluation of image-processing and classification methods. In *IEEE Transactions on Medical Imaging*, volume 18, pages 1085–97, Nov 1999.

[Lac94] Philippe Lacroute. Fast volume rendering using a shear-warp factorization of the viewing transformation. Ph.d. dissertation, Stanford University, 1994.

[Lev88] Mark Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May, 1988.

[MB05] Tom McReynolds and David Blythe. *Advanced Graphics Programming using OpenGL*. Morgan Kauffman Publishers, first edition, 2005.

[MNI] McGill University Montréal Neurological Institute. Datos volumétricos tomados de `http://www.bic.mni.mcgill.ca/brainweb/`.

[RK03] Fernando Randima and Mark Kilgard. *The Cg Tutorial, The definitive guide to Programmable Real-Time Graphics*. Addison-Wesley Publishing Company, first edition, 2003.

[WS01] Rudiger Westermann and Bernd Sevenich. Accelerated volume ray-casting using texture mapping. *IEEE Visualization 2001*, 2001.

Apéndice A

Resultados de pruebas

En este apéndice se puede observar gráficamente los resultados obtenidos en las tablas del capítulo 4. Estas gráficas refuerzan las observaciones hechas en ese capítulo.

A.1. VR Directo basado en texturas

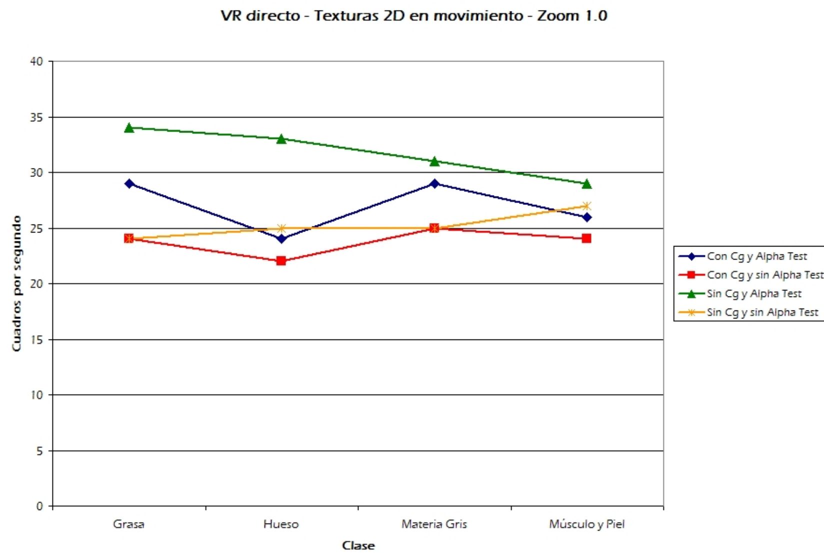

Figura A.1: Texturas 2D - Nivel de Zoom 1.0. Datos a partir de las tablas 4.3, 4.5, 4.7 y 4.9

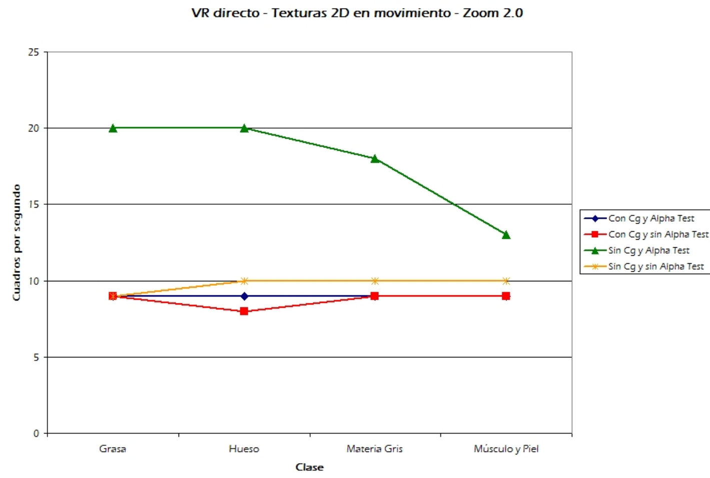

Figura A.2: Texturas 2D en movimiento - Nivel de Zoom 2.0. Datos a partir de las tablas 4.4, 4.6, 4.8 y 4.10

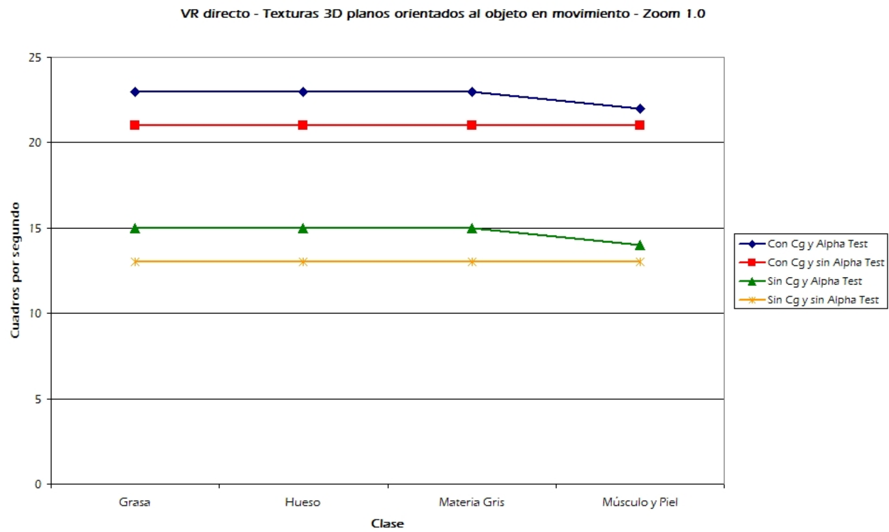

Figura A.3: Texturas movimiento - Planos orientados al objeto - Nivel de Zoom 1.0. Datos a partir de las tablas 4.3, 4.5, 4.7 y 4.9

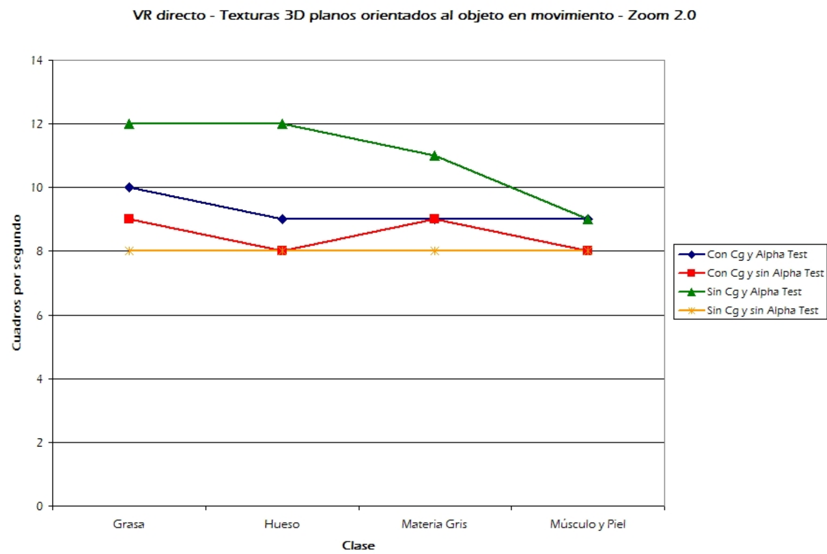

Figura A.4: Texturas movimiento - Planos orientados al objeto - Nivel de Zoom 2.0. Datos a partir de las tablas 4.4, 4.6, 4.8 y 4.10

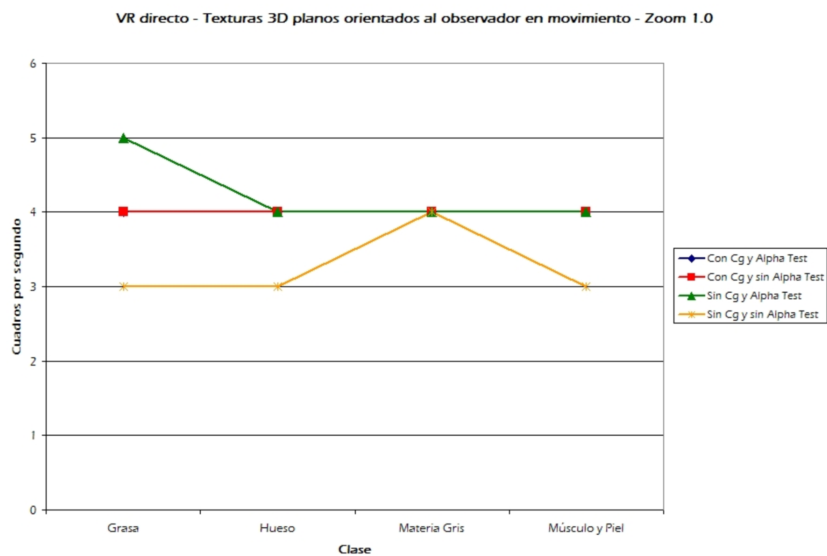

Figura A.5: Texturas movimiento - Planos orientados al observador - Nivel de Zoom 1.0. Datos a partir de las tablas 4.3, 4.5, 4.7 y 4.9

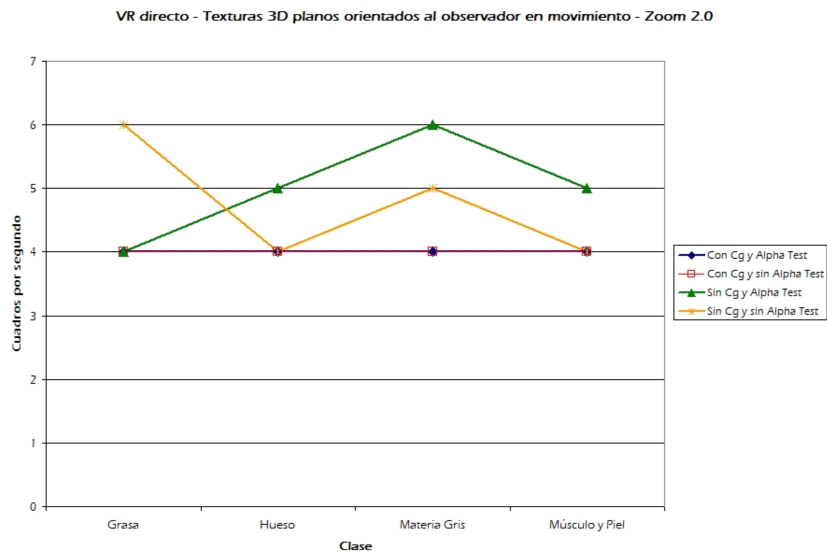

Figura A.6: Texturas movimiento - Planos orientados al observador - Nivel de Zoom 2.0. Datos a partir de las tablas 4.4, 4.6, 4.8 y 4.10

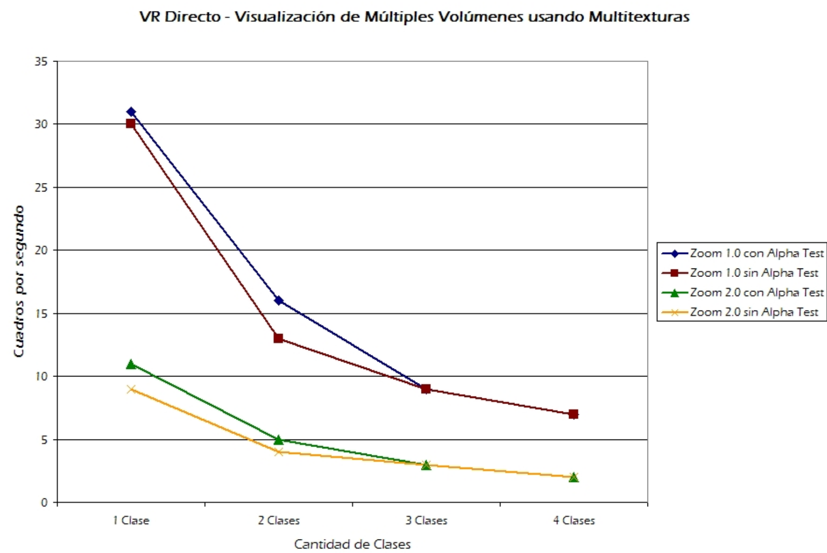

Figura A.7: Texturas 3D Multiples Volúmenes. Datos a partir de la tabla 4.11

A.2. VR Directo Preintegrado - Westermann & Sevenich

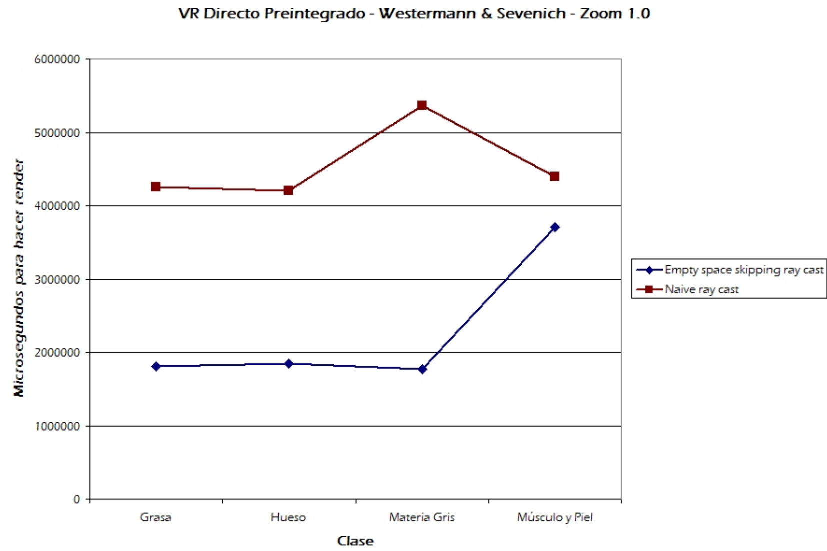

Figura A.8: Westermann & Sevenich - Nivel de Zoom 1.0. Datos a partir de la tabla 4.12

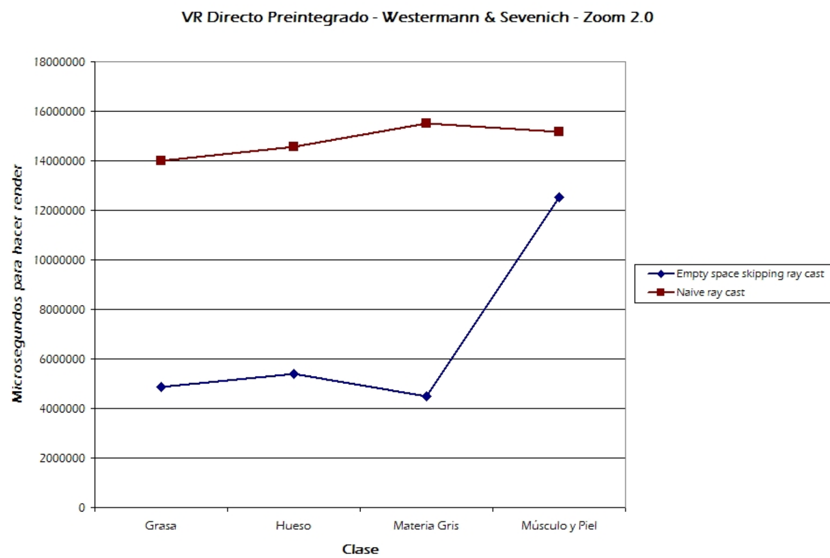

Figura A.9: Westermann & Sevenich - Nivel de Zoom 2.0. Datos a partir de la tabla 4.13

A.3. VR Directo Preintegrado - Engel, Kraus & Ertl

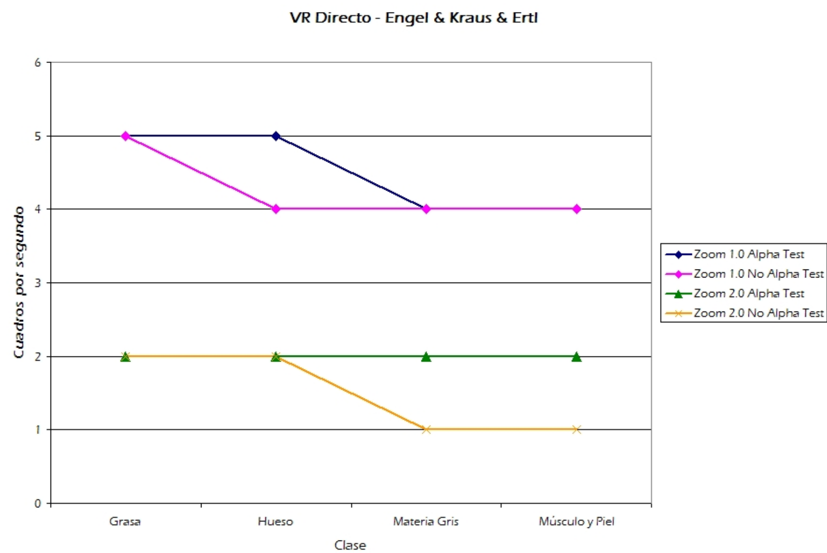

Figura A.10: Engel, Kraus & Ertl. Datos a partir de la tabla 4.14

Apéndice B

Imágenes obtenidas

B.1. Ray casting

Utilizando la implementación descrita en el capítulo 2 se obtuvieron las imágenes presentadas a continuación. Corresponden a materia blanca y músculo con piel. El tiempo de ejecución para todo el pipeline fue variable para los ejemplos presentados, el tiempo mínimo correspondió a 16 segundos, mientras que el máximo fué de unos 18 segundos.

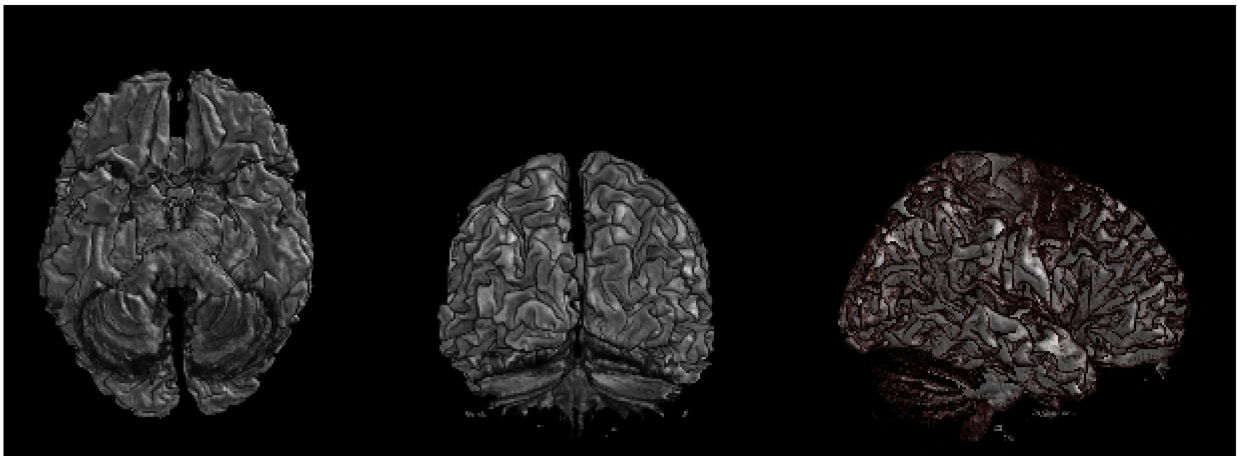

Figura B.1: Ray casting - Visualización de tejido cerebral - Materia Blanca

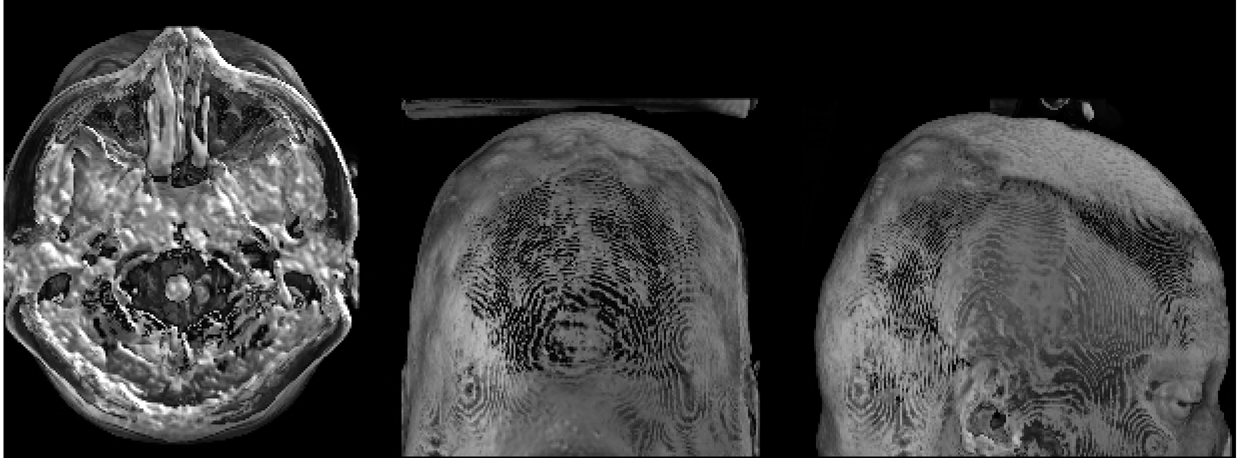

Figura B.2: Ray casting - Visualización de tejido craneal - Músculo y Piel

B.2. Basado en manejo de texturas

En esta sección se presentan imágenes obtenidas a partir de manejo de texturas utilizando lo expuesto en el capítulo 2

B.2.1. Texturas bidimensionales

En la siguiente imagen se puede apreciar como el efecto de popping modifica la visión de la imagen cuando se usan texturas 2D.

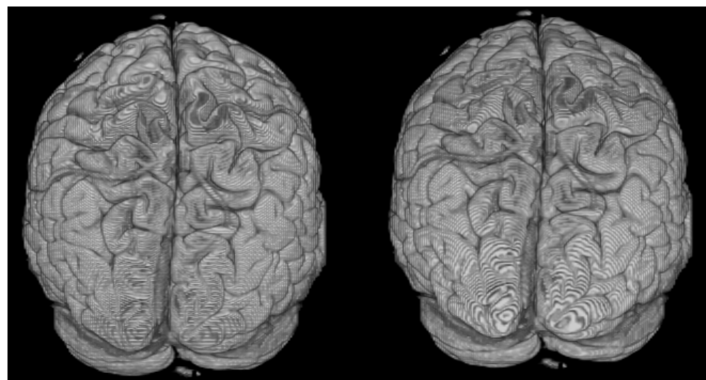

Figura B.3: Efecto de popping utilizando texturas 2D

B.2.2. Texturas tridimensionales

Para los experimentos utilizando multiples volumenes representados en texturas 3D, se presentan las siguientes imágenes correspondientes a la visualización de 3 y 4 clases.

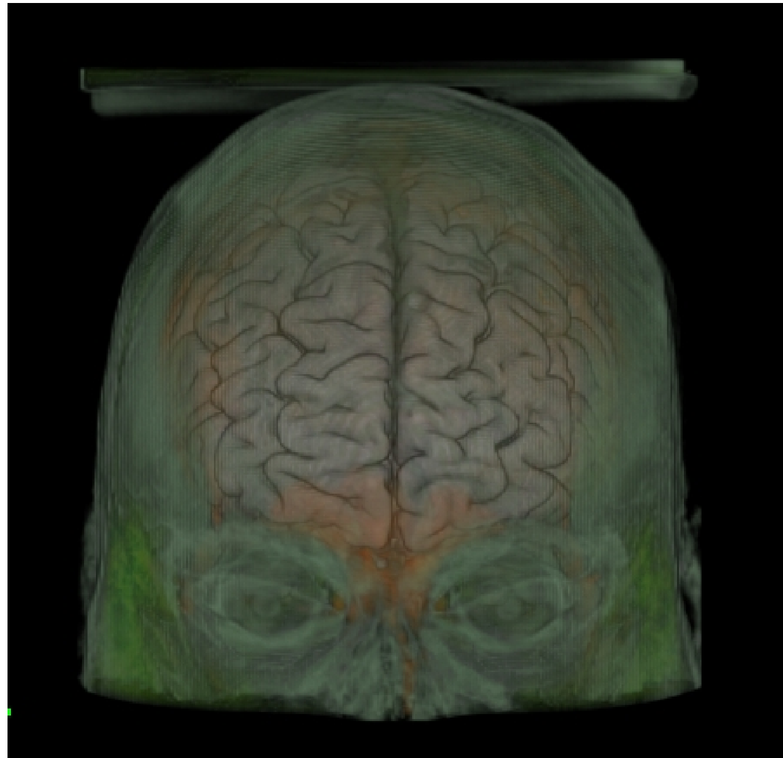

Figura B.4: Basado en texturas - Múltiples Texturas - 4 Clases

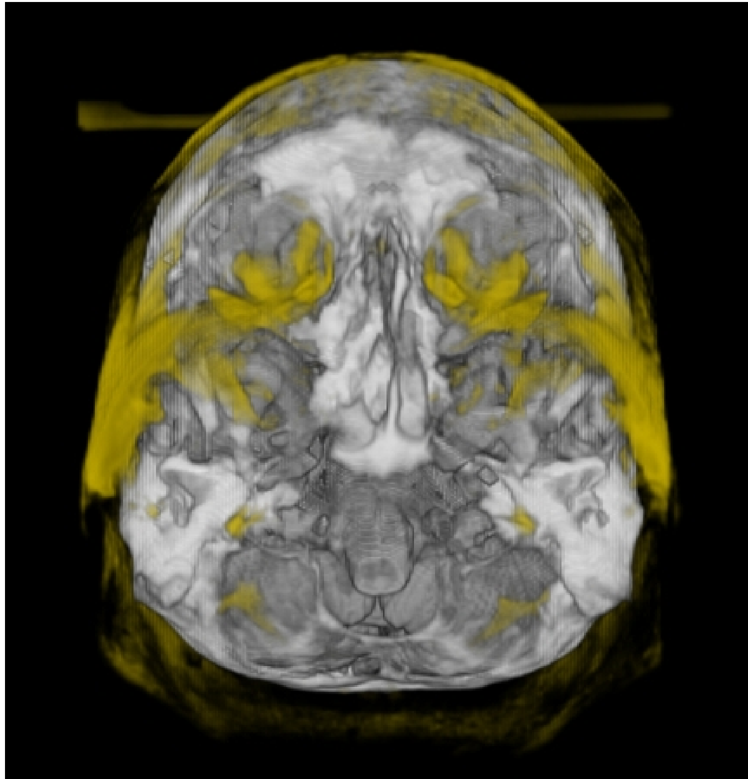

Figura B.5: Basado en texturas - Múltiples Texturas - 3 Clases

B.3. Basado en técnicas de Preintegración

B.3.1. Westermann & Sevenich

Figura B.6: Westermann & Sevenich - Clase Músculo y Piel - zoom 2.0

Figura B.7: Westermann & Sevenich - Clase Hueso - zoom 2.0

B.3.2. Engel, Kraus & Ertl

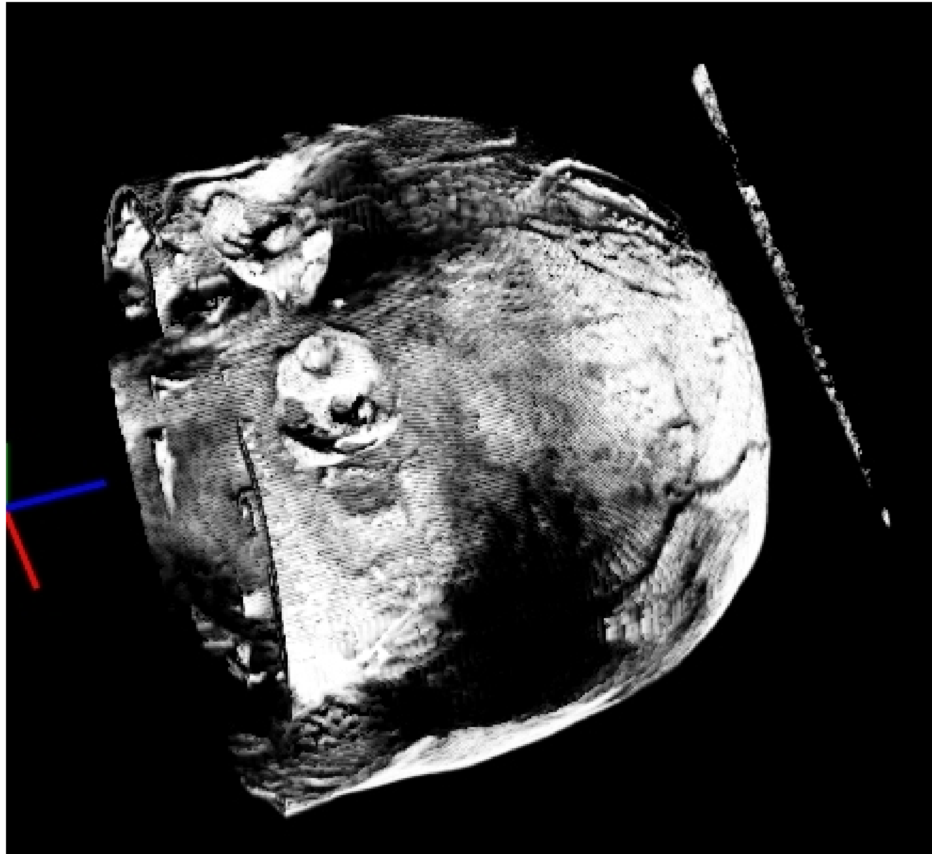

Figura B.8: Engel, Kraus & Ertl - Clase Grasa - zoom 2.0

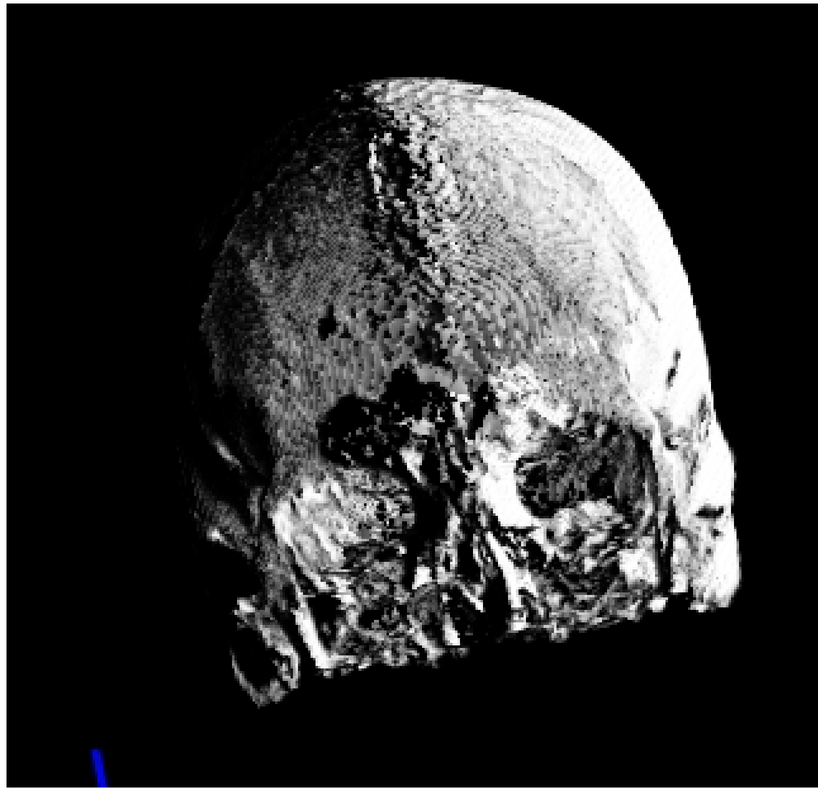

Figura B.9: Engel, Kraus & Ertl - Clase Hueso - zoom 2.0