UNIVERSIDAD NACIONAL DEL SUR

TESIS DE MAGÍSTER EN CIENCIAS DE LA COMPUTACIÓN

Visualización Interactiva De Volúmenes

Mauricio Cele López Belón

BAHÍA BLANCA ARGENTINA

2007

UNIVERSIDAD NACIONAL DEL SUR

TESIS DE MAGÍSTER EN CIENCIAS DE LA COMPUTACIÓN

Visualización Interactiva De Volúmenes

Mauricio Cele López Belón

BAHÍA BLANCA ARGENTINA

2007

PREFACIO

Esta Tesis es presentada como parte de los requisitos para optar al grado académico de Magíster en Ciencias de la Computación, de la Universidad Nacional del Sur, y no ha sido presentada previamente para obtención de otro título en esta universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el Laboratorio de Investigación y Desarrollo en Visualización y Computación Gráfica dependiente del Departamento de Ciencias e Ingeniería de la Computación, durante el período comprendido entre el 10 de mayo de 2005 al 21 de agosto de 2007, bajo la dirección de la Dra. Silvia Mabel Castro, Profesora Titular del Departamento de Ciencias e Ingeniería de la Computación.

Mauricio López Belón

Bahía Blanca, 21 de Agosto de 2007.

Departamento de Ciencias e Ingeniería de la Computación

UNIVERSIDAD NACIONAL DEL SUR

RESUMEN

Existen muchos problemas no resueltos en la visualización directa de volúmenes, uno de ellos es la visualización interactiva de enormes volúmenes de datos. Los métodos de visualización de volúmenes son de gran costo computacional y es por ello que, para lograr una visualización en tiempos interactivos, es imperativo contar con métodos adecuados para los distintos conjuntos de datos de gran tamaño.

Debido al vertiginoso desarrollo de la tecnología relacionada con Internet, la visualización remota de volúmenes se ha viabilizado. Mediante la visualización remota es posible aprovechar los recursos altamente especializados que existen en el mundo a través de una sencilla estación de trabajo. Una de las limitaciones que sufre la visualización remota es la limitada velocidad de transmisión de datos en relación a las dimensiones de los conjuntos de datos que se necesita transmitir. Sin duda, esta limitación afecta seriamente la interactividad de la visualización.

Recientemente se han planteado modelos volumétricos multirresolución basados en redes tetraédricas que permiten tanto la compresión como la transmisión progresiva de los mismos; estos modelos, se basan en dos extensiones de wavelets definidas sobre dominios tetraédricos. La transmisión de tales redes tetraédricas es muy eficiente, ya que se puede hacer por paquetes y en secuencia arbitraria.

En la presente tesis investigamos cómo aprovechar la transmisión progresiva de datos multirresolución para optimizar la visualización de volúmenes en el contexto de la visualización remota. Desarrollamos un algoritmo que permite la visualización progresiva de volúmenes en el dominio wavelet, reduciendo de esta manera el tiempo de espera del usuario remoto. Además mostramos cómo éste modelo es capaz de adaptar constantemente su resolución para mostrar la máxima cantidad de detalles posible, permite la extracción de niveles de detalle (LODs) y refina selectivamente la malla con respecto a parámetros de visión variantes en el tiempo. El refinamiento selectivo adapta constantemente la resolución del volumen a través de dos heurísticas dependientes de la posición del observador: el *Frustum de Visión* y el *Error Geométrico en el Espacio de la Pantalla*.

ÍNDICE

CAPÍTULO I
INTRODUCCIÓN

Actualmente la comunidad científica requiere analizar enormes cantidades de datos volumétricos, muchas veces multidimensionales, multivaluados y variantes en el tiempo. El proceso de análisis requiere la exploración visual de tales conjuntos de datos, la cual tiene como objetivo ayudar a ganar comprensión de los mismos y facilitar el descubrimiento de relaciones y patrones presentes en ellos. La interactividad en el proceso de visualización es indispensable para lograr la comprensión y clasificación de los datos, haciendo evidentes patrones y relaciones en los mismos mediante la modificación de parámetros visuales tales como: color, transparencia, perspectiva, etc. en respuesta a requerimientos del usuario. Sin embargo, en muchos casos, la gran densidad de datos dificulta la exploración, restringiendo la interactividad y, en ocasiones, ofuscando la visualización. La exploración visual a varios niveles de detalle tiene como objetivo permitir la interacción con cantidades enormes de datos cuya exploración es difícil o imposible de realizar dada la cantidad de información condensada en los mismos. La visualización de estos conjuntos de datos es una tarea compleja y computacionalmente intensa que requiere la utilización de técnicas de rendering especiales.

En concreto, la visualización de datos volumétricos, conocidos simplemente como *volúmenes*, requiere de técnicas de *Rendering de Volúmenes* que ofrezcan una representación cualitativa del interior del volumen. Estas técnicas se conocen como "*Rendering Directo de Volúmenes*" (DVR por sus siglas en inglés) ya que permiten visualizar datos volumétricos sin tener conocimiento previo de las características de los mismos, permitiendo descubrir e identificar características inmersas en los datos, facilitando su estudio y entendimiento. Las técnicas de DVR tienen un gran impacto en el diagnóstico médico y son de utilidad en la investigación en ciencias puras y aplicadas.

La gran mayoría de técnicas de DVR poseen un costo computacional elevado debido, principalmente, a tres factores: i) los modelos de iluminación requieren de cómputos elaborados, o en su defecto, *funciones de transferencia* elaboradas ii) los algoritmos de rendering requieren muestrear constantemente el interior del volumen en cierto orden espacial y iii) el tamaño de los conjuntos de datos, el cual suele superar la cantidad de recursos de memoria y saturar el bus de datos, aumenta el tiempo de procesamiento limitando la interactividad de la visualización.

Las técnicas de DVR hacen uso intenso de recursos computacionales, clusters, paralelismo, etc. con el objetivo de lograr visualizaciones interactivas de gran

calidad. En la actualidad las tarjetas aceleradoras gráficas en el *estado del arte* constituyen una nueva plataforma de desarrollo para todo tipo de algoritmos de graficación y cálculos científicos de propósito general. Muchos esfuerzos de investigación están avocados a aprovechar esta tecnología como soporte para las técnicas de DVR *stand alone*. Por otro lado, Internet plantea la necesidad de contar con sistemas de visualización remota y distribuida. Con la disponibilidad de la GRID y el soporte de los Web Services se han propuesto varios sistemas de visualización remota y distribuida. Mediante la visualización remota es posible aprovechar los recursos altamente especializados que existen en el mundo a través de una sencilla estación de trabajo. Sin embargo, una de las limitaciones que sufre la visualización remota es la limitada velocidad de transmisión de datos en relación a las dimensiones de los conjuntos de datos que se requieren transmitir. Sin duda, esta limitación afecta seriamente la interactividad de la visualización.

En consecuencia, al problema de exploración de volúmenes de gran tamaño se agrega el problema de transmisión de éstos por la red. Es entonces necesario contar con algoritmos que permitan el almacenamiento, la transmisión y la visualización de grandes conjuntos de datos volumétricos. Tanto la exploración como la transmisión de datos se pueden optimizar, espacial y temporalmente, a través de modelos multirresolución de los mismos. En particular los modelos que usan la transformada wavelet se han vislumbrado como una opción adecuada para la representación de datos volumétricos en múltiples resoluciones, ya que poseen un alto nivel de compresión y habilitan la transmisión progresiva de los datos por la red, de tal forma que cada resolución puede ser transmitida por separado y en secuencia arbitraria. Esto resulta de especial relevancia para sistemas de visualización remota y distribuida en los cuales el principal cuello de botella es la ineficiencia de transmisión de datos por la red.

1.1 Contribuciones

Si bien los modelos multirresolución basados en wavelets teóricamente constituyen una gran promesa para la visualización remota de grandes volúmenes representados mediante redes tetraédricas, aún no se han desarrollando algoritmos concretos que exploten sus bondades. En la presente tesis investigamos cómo aprovechar la transmisión progresiva de datos en múltiples resoluciones para optimizar la visualización directa de volúmenes representados mediante redes de tetraedros en el contexto de la visualización remota.

Tomando como punto de partida el modelado multirresolución de redes tetraédricas basado en wavelets ([Castro97], [Boscardin01]), el cual permite la compresión y la transmisión progresiva de volúmenes ([Castro05], [Castro06]), proponemos una estrategia para la visualización remota de este tipo de redes tetraédricas. Dicha estrategia abarca desde definir algoritmos concretos para realizar los procesos de análisis y síntesis wavelet con los cuales se construyen los volúmenes que son transmitidos por el servidor, hasta el rendering progresivo de los mismos en la máquina cliente.

Tomando en cuenta lo dicho previamente, las contribuciones presentadas en esta tesis son:

- *Un algoritmo de análisis multirresolución basado en wavelets definidas sobre atributos asociados a los vértices de una red semi-regular de tetraedros.* Este algoritmo aplica la transforma wavelet a una red *semi-regular* de tetraedros cuyos vértices están asociados a un campo escalar. Especifica la generación de una jerarquía de coeficientes de detalle asociados a los tetraedros de una red que posee diferentes resoluciones. Si bien el proceso de análisis está descripto en trabajos anteriores, no existía un algoritmo concreto que lo llevara a cabo.

- *Una arquitectura de visualización progresiva Cliente – Servidor.* Diseñamos una arquitectura de visualización progresiva que nos sirve como marco de referencia para definir el *pipeline* de rendering progresivo.

- *Un conjunto de algoritmos que habilitan el Rendering Progresivo de volúmenes.* Para la realización del Rendering Progresivo utilizamos técnicas de multirresolución de volúmenes así como la transformada wavelet inversa antes de realizar el Rendering Directo de Volúmenes. Se proponen dos nuevos algoritmos de Refinamiento Selectivo, uno simple y otro basado en heurísticas dependientes del punto de visión. Estos algoritmos requieren a su vez la definición de dos algoritmos básicos: un algoritmo de Refinamiento y uno de *Coarsening*.

- *El diseño de las estructuras de datos necesarias para realizar el Refinamiento Selectivo.* Se proponen estructuras de datos sencillas y compactas, las cuales no requieren del almacenamiento explícito de ningún tipo de adyacencia de tetraedros. Las estructuras están asociadas a un conjunto de índices los cuales son descritos en detalle.

- *Un algoritmo de remuestreo de volúmenes irregulares.* Este algoritmo tiene como finalidad realizar un mapeo de las redes tetraédricas no estructuradas hacia redes *semi-regulares* que poseen la propiedad de conectividad de subdivisión. Es de utilidad para construir redes que cumplen con los requisitos para la aplicación del algoritmo de análisis multirresolución basado en wavelets mencionado anteriormente.

1.2 Estructura y Contenido

La presente tesis está organizada del siguiente modo. En los capítulos II, III y IV situamos el contexto general de nuestra investigación. Resumimos las características de los datos volumétricos y revisamos de forma general la función de transferencia y las técnicas de rendering directo de volúmenes. En el Capitulo V introducimos las métricas de error para redes tetraédricas multirresolución y describimos el modelado multirresolución basado en wavelets dentro del contexto más general del modelado multirresolución de símplices. En el Capítulo VI

revisamos brevemente los trabajos previos en visualización progresiva de volúmenes, presentamos nuestra investigación en visualización progresiva de volúmenes en el dominio wavelet y mostramos los resultados obtenidos. Finalmente, en el Capítulo VII presentamos nuestras conclusiones y trabajo futuro.

- *Capítulo II Marco Teórico.* Definimos la naturaleza y origen de los datos volumétricos, sus características principales y los modelos ópticos utilizados para su visualización.

- *Capítulo III Función de Transferencia.* Definimos intuitivamente la *función de transferencia* y hacemos un resumen de los trabajos más importantes relacionados con su construcción.

- *Capítulo IV Técnicas de Rendering Directo de Volúmenes.* Hacemos una revisión de las técnicas de Rendering Directo de Volúmenes aplicables a dominios regulares, estructurados y no estructurados.

- *Capítulo V Multirresolución de Volúmenes.* Revisamos las diversas métricas de error y las técnicas para el modelado multirresolución de redes tetraédricas regulares e irregulares, haciendo énfasis en las técnicas basadas en wavelets.

- *Capítulo VI Visualización Interactiva en el Contexto de la Visualización Remota.* Presentamos la problemática relacionada a la visualización remota Cliente – Servidor. Revisamos los trabajos previos en cuanto a la visualización progresiva de volúmenes. Introducimos el *pipeline* de Visualización Progresiva en el Dominio Wavelet, la representación interna de la red *semi-regular* de tetraedros, las estructuras de datos utilizadas, los algoritmos de refinamiento selectivo en los cuales está embebida la transformada inversa wavelet y finalmente, los resultados obtenidos.

- *Capítulo VII Conclusiones y Trabajo Futuro.* Presentamos nuestras conclusiones y planes de trabajo futuro.

CAPÍTULO II
MARCO TEÓRICO

En este capitulo introducimos la terminología y definiciones básicas usadas en el presente trabajo. Muchas de las definiciones son estándar, por lo tanto no son explicadas en detalle. Los detalles pueden encontrarse en [Max95], [Sabella88], [Watt01], [Watt03], [Kniss02], [Shewchuk97], [Crawfis95].

2.1 Datos Volumétricos

Los volúmenes son cantidades densas de datos, uni o multivaluados, que están embebidos en el espacio geométrico tridimensional. En ocasiones es necesario el uso de alguna metáfora que provea de sustrato espacial a los datos; sin embargo, como veremos más adelante, la mayor parte de ellos proviene de dominios físicos que poseen un sustrato espacial intrínseco. Podemos asumir, sin perder generalidad, que el volumen de datos es extraído del dominio físico mediante muestreo y luego es procesado para conectarlo de tal manera que forme un conjunto de estructuras de *celdas* volumétricas. Una celda está formada por cuatro o más *vértices* no coplanares y las celdas contiguas suelen compartir vértices unas con otras para de esta manera formar una *malla*. Cada vértice de la *malla* posee una o varias magnitudes físicas de tipo escalar.

Los volúmenes provienen de distintos dominios de la ciencia, por ejemplo en ciencia e ingeniería se cuenta con los generados por el *Método de Elemento Finito* (FEM), el *Método de Volumen Finito* (FVM) y el *Método de la Frontera Finita* (*Boundary Finite Method* BEM), útiles para resolver *Ecuaciones Diferenciales Parciales* de límites complejos puestas en objetos o dominios que tienen forma irregular. Estas ecuaciones modelan fenómenos caóticos tales como deformación mecánica, transferencia de calor, dinámica de fluidos, propagación de ondas electromagnéticas y mecánica cuántica.

En medicina, los volúmenes de datos provienen de *Scanners* de Tomografía Computarizada (CT), Resonancia Magnética (MRI) y Ultrasonido. La mayoría de las técnicas de adquisición de imágenes médicas existentes, obtienen datos 3D a partir de la acumulación de cortes de imágenes 2D. Los diferentes cortes 2D son obtenidos por medio del desplazamiento del paciente con respecto al equipo de adquisición o viceversa.

En este trabajo nos centraremos en las *mallas* (volúmenes de datos con sustrato espacial) univaluadas, es decir, mallas que tienen asociado un solo valor escalar a cada vértice. Matemáticamente podemos definir una *malla* univaluada como un *Campo Escalar*, es decir, como una función **continua** *f(x)* cuyo dominio es un conjunto de puntos tridimensionales y cuyo rango es un conjunto de escalares:

$$f(\vec{x}) \in \mathrm{IR} \quad \text{y} \quad \vec{x} \in \mathrm{IR}^3$$

La definición del campo escalar como una función continua implica que el dominio y el rango son continuos, es decir, el dominio no sólo son los vértices de la malla, sino también cualquier punto que se encuentre dentro de las celdas formadas por ellos. De manera similar, el rango no sólo son los escalares asociados a cada vértice, sino también los generados por la interpolación de estos en el interior de las celdas de la malla.

2.2 Reconstrucción de la señal

Dado que, en una malla, los valores escalares se encuentran asociados a los vértices de cada celda, es necesario usar alguna forma de interpolación para aproximar su valor en la superficie y en el interior de la misma. En la terminología de la *Teoría del Muestreo,* un campo escalar continuo, es una *señal* y cada escalar constituye una *muestra* tomada de la *señal*. La *Teoría del Muestreo* es necesaria para definir qué clase de interpolación se debe aplicar para *reconstruir* la función original a partir de las *muestras*. Cuanto mejor se aproxime la función original, mejor calidad tendrá la imagen generada al renderizar el volumen.

Al ser la *señal* una función continua en el dominio espacial, posee un espectro continuo en el dominio frecuencia. Esto se puede comprobar usando el Teorema de Fourier al descomponer las frecuencias de la señal. Sin embargo, de hecho, solo tenemos acceso a un número discreto de *muestras*, las cuales fueron previamente obtenidas a cierta frecuencia, a la que llamaremos *frecuencia de muestreo*. Es claro que si la señal original representaba frecuencias más altas que la *frecuencia de muestreo*, ésta habría sido pobremente muestreada y por lo tanto no podrá ser reconstruida fielmente; en consecuencia se habrá perdido la alta frecuencia o detalle de los datos ([Watt01], [Watt03], [Fisher03]).

En el dominio frecuencia, la mayor parte de la señal está representada en la baja frecuencia, en tanto que la alta frecuencia representa sólo una pequeña parte de la misma. Esto se refleja en el domino espacial donde, análogamente, existe una transición suave en la mayor parte de los escalares del volumen, mientras que una pequeña parte exhibe cambios bruscos en los mismos. Estos cambios bruscos representan el detalle. En consecuencia podemos notar que perder alta frecuencia de la señal en el dominio de Fourier, equivale a perder el detalle de los datos en el dominio espacial, esto implica también perder nitidez en la imagen generada mediante DVR. Existe una frecuencia de muestreo teórica llamada *Frecuencia de Nyquist*, la cual se calcula específicamente como cota inferior, es decir, sobre ella

debe estar la *frecuencia de muestro* para muestrear la señal con suficiente detalle para crear una imagen nítida ([Watt01], [Watt03], [Fisher03]).

La forma más simple y eficiente de reconstrucción de la señal es la interpolación trilineal, la cual es aceptable, si y solo si, la función ha sido muestreada sobre la frecuencia de Nyquist. Los filtros no lineales son más precisos ya que se adaptan mejor y toman en cuenta más *muestras*, aproximando mejor la función original. Sin embargo éstos no son eficientes en términos computacionales por su complejidad y la carencia de implementaciones en hardware.

El proceso de interpolación en *volúmenes regulares* comúnmente se hace mediante *Convolución*. La Convolución es la multiplicación de dos señales en el dominio frecuencia, concretamente multiplicar la señal con una señal de filtro. En el dominio espacial, la Convolución es la suma ponderada de *muestras* con pesos específicos. Los pesos son dados como una matriz tridimensional llamada *Kernel del filtro* o *Kernel de Convolución* ([Fisher03], [Watt01]).

Durante el Rendering Directo de Volúmenes frecuentemente se aproxima numéricamente la *Ecuación de Rendering* usando la sumatoria de Riemann ([Max95]). Esto implica reconstruir la señal, usando algún tipo de interpolación, para luego tomar nuevas muestras a intervalos regulares. Esto equivale a hacer un nuevo muestreo, el cual potencialmente causa pérdida de alta frecuencia. A este proceso se lo conoce como *Remuestreo* ([Kniss02]).

2.3 Topologías

Muchos modelos matemáticos en ciencia e ingeniería se aproximan mediante el *Método de Elemento Finito*, el cual, mediante algún método de triangulación o refinamiento, produce mallas de poliedros (usualmente tetraedros o hexaedros) comúnmente conocidas como *Mallas de Elemento Finito*. Estas mallas tienen forma irregular, poseen concavidades y también celdas de diferentes tamaños y formas. Por otro lado, en el campo de la medicina, los scanners de CT, MRI y ultrasonido, producen mallas de celdas *regulares* que pueden ser representadas simplemente como un arreglo tridimensional. Es a partir de estas notorias diferencias que nace la clasificación topológica fundamental de mallas en *Estructuradas* y *No Estructuradas* ([Shewchuk97]).

Basándonos en las topologías nombradas por Schroeder *et al* ([Schroeder02]), podemos enumerar cuatro tipos: Uniforme, Rectilínea, Malla Curvilínea y No-Estructurada (figura 1).

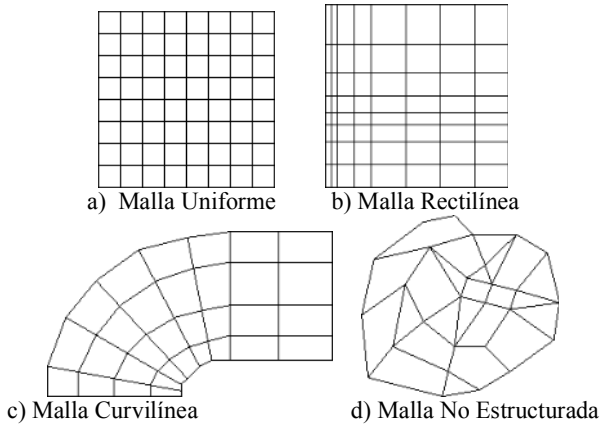

Figura 1: Topologías encontradas frecuentemente

Otros autores ([Max03], [Moreland04], [Shewchuk97]), las clasifican simplemente como *Estructuradas* y *No-Estructuradas*. Dentro de las *Estructuradas* se consideran todas las mallas que se pueden representar como un arreglo tridimensional de escalares y cuya conectividad queda implícita (es decir *a, b y c* en la Figura 1). La malla Curvilínea se puede representar como una malla Rectilínea a la cual se le aplica una función parametrizada que aplica la curvatura. Las *No-Estructuradas* engloban a las estructuradas ya que son más generales, pueden tener concavidades (por ejemplo mallas cuya *Frontera* es distinta al *Armazón Convexo*) y necesitan conectividad explícita entre celdas (Figura 1 d).

La topología *No-Estructurada* presenta una serie de problemas que aún son área activa de investigación. El ordenamiento correcto y en tiempos interactivos de este tipo de mallas es un tema parcialmente resuelto ([Moreland04]). El rendering de celdas cuyos escalares varían no linealmente en su interior aún no está resuelto; hasta el momento sólo se han hecho aproximaciones. Algunos problemas de las mallas *No-Estructuradas* pueden ser resueltos dividiendo los poliedros complejos en tetraedros, con el fin de eliminar la superposición de celdas debido a concavidades. Sin embargo el mayor problema con la subdivisión es el aumento del número de poliedros y por ende, del espacio de almacenamiento y de cálculos que deben realizarse para su visualización ([Max03]).

En contraposición, la topología Estructurada es particularmente eficiente. Cada celda suele ser de tamaño fijo, convexa y sin superposiciones. Esta celda adquiere el nombre de ***voxel*** cuando los valores dentro de ella no varían o su variación es despreciable. Es por ello que se suele definir un ***voxel*** como la unidad mínima de volumen de una imagen tridimensional, similar por analogía al píxel en las imágenes bidimensionales.

2.4 Modelos Ópticos

Primero Sabella en 1988 ([Sabella88]) y después Max en 1995 ([Max95]) derivan formalmente modelos ópticos para el *Rendering Directo de Volúmenes*.

La derivación de Max ([Max95]) parte de una ecuación diferencial de primer orden:

$$\frac{dI}{ds} = L(s)\tau(s) - I(s)\tau(s) \tag{1}$$

La ecuación expresa el cambio en intensidad de la luz I cuando atraviesa un segmento infinitesimal de volumen *s,* donde $L(s)$ es la cantidad de **luz emitida** por *s*, $\tau(s)$ es la cantidad de **luz que se espera sea absorbida** por *s* y $I(s)$ es la cantidad de luz externa que llega a *s*.

En palabras más simples, la intensidad de luz en un ***voxel*** es, entonces, la diferencia entre la luz que éste *puede* emitir y *le puede* llegar. Se deduce fácilmente que si el volumen es emisor de luz y además es denso, prácticamente no podrá ser iluminado por la luz externa.

Para calcular la intensidad producida por longitud de onda luminosa al atravesar todo el volumen se debe integrar la ecuación (1). Se multiplican ambos miembros por el factor de integración $\exp\left(\int_0^s \tau(t)dt\right)$:

$$\frac{d}{ds}\left(I(s)e^{\int_0^s \tau(t)dt} \right) = L(s)\tau(s)e^{\int_0^s \tau(t)dt}$$

Integrando ambos miembros con s = 0 al entrar al volumen y s = D cerca al ojo, se obtiene:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D L(s)\tau(s)e^{-\int_s^D \tau(t)dt} ds \tag{2}$$

siendo:

- $I(D)$ la intensidad luminosa que llega al ojo.
- I_0 la luz externa que ilumina el volumen en dirección al ojo.
- $e^{-\int_0^D \tau(t)dt}$ la cantidad de luz que se espera sea atenuada al atravesar todo el volumen.
- $L(s)$ la intensidad de luz emitida por unidad de volumen en s.
- $\tau(s)$ la cantidad de luz absorbida por unidad de volumen en s.
- $e^{-\int_s^D \tau(t)dt}$ la cantidad de luz que se espera sea atenuada en s al atravesar el volumen.

La ecuación (2) es llamada la **Ecuación de Rendering**. Su derivación completa se encuentra en [Sabella88], [Max95] y [Moreland04].

Intuitivamente podemos decir que la intensidad final (luz que llega al ojo) es proporcional a la intensidad de luz que llega al volumen (atenuada exponencialmente) y a la suma de luz que *puede* ser emitida por cada voxel.

La ecuación (2) es válida para cada longitud de onda del espectro visible ([Sabella88], [Max95], [Moreland04]). Por razones prácticas, se consideran tres regiones del espectro colorimétricamente independientes, conocidas como *primarios*, tales que, para cualquier color *f* existe una combinación lineal de *primarios* que producen el color *f*. En la mayoría de los casos la Ecuación (2) sólo se aplica una vez por cada píxel y la intensidad viaja empaquetada en un color de tres componentes *primarios* (por ejemplo RGB).

Como veremos a continuación, se han impuesto algunas restricciones a la ecuación (2), de manera que pueda ser más eficiente y simple de evaluar, pero a costa de pérdida de precisión y por consiguiente de calidad. En ocasiones, las restricciones impuestas son severas y suelen ser usadas en casos específicos; estas son, por ejemplo, los modelos *Solo Emisión* y *Solo Absorción* ([Max95]), los cuales aún son usados por algunas técnicas de DVR. A continuación mostramos ambas restricciones:

Sólo Absorción

Para modelar sólo la *absorción* se requiere que el volumen no sea autoemisivo, es decir se requiere que en la ecuación (2) $L(s) = 0$, con lo cual la única luz, si es que hay alguna, proviene de una fuente externa ([Moreland04]). De este modo la ecuación (2) se reduce a:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} \tag{3}$$

La ventaja de este modelo óptico, aparte de su simplicidad, es que se puede usar la Transformada Rápida de Fourier mediante la cual se puede hacer el Rendering de Volúmenes directamente en el dominio frecuencia. Sin embargo, como el volumen no emite luz, no se percibe la profundidad en la imagen ([Max95], [Moreland04]).

Sólo Emisión

Para modelar solo *emisión* se requiere hacer $\tau(s) = 0$, con lo cual toda la luz, tanto la proveniente de fuentes externas como la emitida por el propio volumen, llegan al ojo. Esto en general no resulta práctico ya que la luz se satura en intensidad, por lo cual Max ([Max95]) define el factor $g(s) = L(s)\tau(s)$ al que llama *source term*, el cual incluye reflexión y emisión. Con esto la ecuación (2) se reduce a:

$$I(D) = I_0 + \int_0^D g(s)ds \tag{4}$$

Este modelo es muy simple y también puede ser usado conjuntamente con la Transformada Rápida de Fourier mediante la cual se puede aplicar Rendering de Volúmenes directamente en el dominio frecuencia. Sin embargo sufre del mismo problema que el modelo de *Solo Absorción*, es decir, produce imágenes sin referencia de profundidad ([Max95]).

2.5 Calculo Aproximado de la Ecuación del Rendering

La Ecuación del Rendering es difícil de evaluar. El cálculo exacto de la ecuación, es decir la solución cerrada, algunas veces existe ([Debrin88]), sin embargo en la mayoría de casos la ecuación (2) se aproxima numéricamente usando la sumatoria de Riemann.

Max ([Max95]) muestra cómo evaluar la ecuación usando explícitamente la suma de Riemann y una tabla con el cálculo del término $e^{-\int_s^D \tau(t)dt}$ guardado en ella.

Si consideramos que el número de segmentos de la suma de Riemann es d y la partición es $\Delta x = \frac{D}{d}$, la ecuación (2) para un segmento cualquiera de la integral es:

$$I(d_i) = I_{d_{i-1}} e^{-\int_{d_{i-1}}^{d_i} \tau(t)dt} + \int_{d_{i-1}}^{d_i} L(s)\tau(s)e^{-\int_s^{d_i} \tau(t)dt} ds \tag{5}$$

Si consideramos Δx pequeño, la emisión y la absorción pueden considerarse constantes para todo el segmento, es decir $L(s) = L$ y $\tau(s) = \tau$, tal que la ecuación (5) puede escribirse así:

$$I(d_i) = I_{d_{i-1}} e^{-\int_{d_{i-1}}^{d_i} \tau\, dt} + \int_{d_{i-1}}^{d_i} L\tau\, e^{-\int_s^{d_i} \tau\, dt} ds \tag{6}$$

Al integrar la ecuación (6) tenemos:

$$I(d_i) = I_{d_{i-1}} e^{-\tau(d_i - d_{i-1})} + L\tau\left(1 - e^{-\tau(d_i - d_{i-1})}\right) \tag{7}$$

Teniendo en cuenta que $\Delta x = (d_i - d_{i-1})$ es la longitud del segmento y reemplazando en la ecuación (7) tenemos:

$$I(d_i) = I_{d_{i-1}} e^{-\tau \Delta x} + L\tau\left(1 - e^{-\tau \Delta x}\right) \tag{8}$$

Donde $I_{d_{i-1}}$ es la intensidad calculada para el segmento previo. Esto significa que cada segmento debe ser computado en un orden específico, es decir, de atrás hacia delante (*Back-To-Front*) de modo que la intensidad $I_{d_{i-1}}$ siempre esté disponible. Stein *et al* ([Stein94]) muestran cómo usar la capacidad del hardware gráfico para calcular la integral definiendo la ***opacidad*** como $\alpha = 1 - e^{-\tau \Delta x}$, y componiendo colores en orden *Back-To-Front* en el Frame Buffer usando el operador OVER de Porter *et al* ([Porter84]):

$$I_i = I_{i-1}(1 - \alpha) + L\tau\alpha \tag{9}$$

A pesar de que técnicamente es posible usar un número arbitrario de particiones, escoger un número grande aumenta el tiempo de cálculo. Otras formas más sofisticadas de evaluar la integral, sin considerar Emisión y Absorción constantes para cada segmento, pueden lograr mejor precisión con menos particiones ([Moreland04], [Engel01]) pero la complejidad aumenta considerablemente. ♦

CAPÍTULO III
FUNCIÓN DE TRANSFERENCIA

Para lograr que la visualización de volúmenes sea útil, es necesario clasificar, separar o resaltar las distintas zonas de interés que están parcial o totalmente obscurecidas dentro del volumen. En medicina, por ejemplo, se usa la Resonancia Magnética para detectar coágulos y tumores; para el especialista en tumores es importante diferenciar claramente los diversos tejidos; al mismo tiempo, un médico traumatólogo puede requerir la visualización de la estructura ósea de los mismos datos para detectar fracturas. En ciencia, dependiendo de la naturaleza de los datos, se deben resaltar las diferentes temperaturas, densidades, magnitudes de velocidad o fuerza, etc. A lo largo de décadas de investigación se ha demostrado que la Clasificación de Características no es un problema trivial sino todo lo contrario, resulta sumamente difícil y consume mucho tiempo. En este capítulo nos limitaremos tan solo a explicar los estudios principales realizados en este tema.

3.1 Noción Intuitiva de la Función de Transferencia

El concepto de Función de Transferencia nace de la necesidad concreta de traducir una cantidad física (escalar) en un color. La cantidad de energía radiante absorbida y emitida por una partícula depende de la composición de la misma, es decir de su *material* ([Kwok92], [Watt03]), por ejemplo en los volúmenes captados por medio de Tomografía Computarizada (CT) los escalares revelan la densidad del tejido. En ese caso, si el valor de un escalar es cercano a cero (el aire), el espacio que ocupa ese voxel estará vacío y por tanto será casi transparente, en cambio si un escalar es cercano a 100 (el hueso) el espacio que ocupa ese voxel estará parcialmente lleno pero será opaco con un determinado color (blanco en el caso del hueso).

Es así que las propiedades físicas de los *materiales* que conforman el volumen son, en última instancia, las que definen el color y la transparencia de los voxels. Teniendo en cuenta esto, tenemos que incluir estas propiedades en la Ecuación del Rendering.

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D L(s)\tau(s) e^{-\int_s^D \tau(t)dt} ds$$

En esta ecuación, la intensidad de la luz emitida por una partícula $L(s)\tau(s)$ corresponde al *color* del voxel y asumimos por practicidad que está representado por tres componentes (por ejemplo el modelo RGB es muy usual). El término $e^{-\int_s^D \tau(t)dt}$ corresponde a la absorción de luz proporcional a la cantidad de partículas del voxel –comúnmente llamado transparencia. De este modo todos los términos de las integrales quedan definidos para ser operados tal como vimos en el capítulo anterior. En la práctica notamos que varios voxels comparten el mismo color y otros tantos la misma transparencia, por lo tanto resulta *práctico* crear una paleta, relativamente pequeña, con aquellos colores y transparencias. Llamamos a esta *paleta* **Función de Transferencia**.

En realidad la Función de Transferencia es una función propiamente dicha, una función continua multivaluada $f(s) = (c, \tau)$ cuyo dominio son todos los escalares del volumen (una señal continua) y cuyo rango es una serie de colores y transparencias empaquetados en números de 32 bits (RGBA). En otras palabras, se trata de un *mapeo visual* que traslada los escalares del dominio físico original hacia el dominio espectral de colores y transparencias.

Al aproximar la Ecuación del Rendering con la sumatoria de Riemann, hallamos la intensidad en cada segmento de la sumatoria a través de la Ecuación (8):

$$I_i = I_{i-1}(1-\alpha) + L\tau\alpha$$

El término $\alpha = 1 - e^{-\tau\Delta x}$ representa la *opacidad*, es decir, el complemento de la transparencia. La opacidad es entonces un valor continuo que varía de cero a uno y representa el porcentaje de ocupación de espacio de un voxel.

Debido a que la sumatoria de Riemann es la forma estándar de aproximar la Ecuación del Rendering y además por razones de eficiencia, se precalcula el término α y se guarda en el canal alfa de los colores en formato RGBA. De manera similar el color RGB es precalculado como el producto $L\tau$ en una Función de Transferencia discreta.

3.2 Creación Manual de la Función de Transferencia.

Si bien los colores y opacidades de la Función de Transferencia están basados en los datos del volumen, no es útil crear una función ingenua que muestre todos los datos del volumen tal cual fueron captados originalmente. La identificación de qué características deben ser visualizadas y cómo, es una tarea iterativa que se da previa al Rendering y probablemente también posterior a éste.

El rango de la Función de Transferencia es típicamente una serie de valores (color, opacidad, emisividad, etc), pero el valor más importante de todos es la

opacidad. La opacidad determina cuan inteligible e informativo será el Rendering de Volúmenes. Si la opacidad no está bien establecida, los datos (las características) útiles para el científico pueden no aparecer en absoluto o bien pueden estar obstruidas por características más opacas.

Por definición, la creación de la Función de Transferencia no se hace en el dominio espacial, sino en el dominio de los datos. Esto dificulta la tarea de buscar características importantes. Se debe hacer uso de herramientas estadísticas, tales como histogramas, para asistir en la clasificación del volumen. El rol de los histogramas es guardar variaciones y patrones de algunas propiedades de los datos y permitir descartar datos que no interesan (asignándoles opacidad cero). El análisis de los histogramas provee además información adicional acerca del volumen de datos más allá de la creación de la Función de Transferencia.

Actualmente existen sofisticados editores que permiten modificar visualmente, a mano, la Función de Transferencia, basados en el conocimiento y experiencia que posea el científico. Dichos editores proveen histogramas y pre-visualización de los resultados. Pese a esto, muchas veces el proceso se hace demasiado lento y tedioso para los investigadores. En la Figura 2 se aprecian diferentes Funciones de Transferencia que revelan diferentes características del volumen. La opacidad es el valor más importante, ya que revela o esconde características, mientras que los colores ayudan a enfatizarlas.

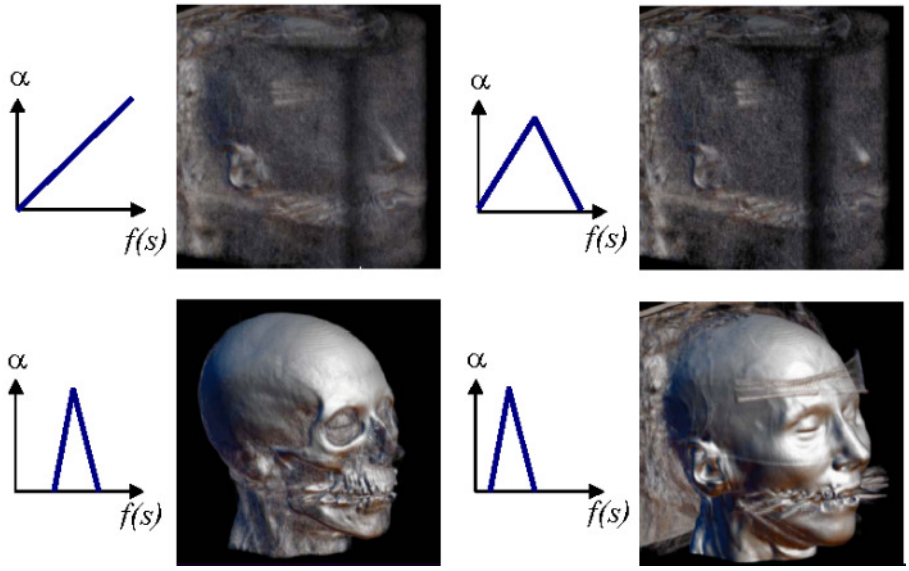

Figura 2: Diferentes Funciones de Transferencia de la opacidad editadas a mano para el "Craneo de Chapel Hill" de la Universidad de Carolina del Norte.

3.3 Creación Semi-Automática de la Función de Transferencia.

Una forma de minimizar la necesidad de que el científico de un área específica tenga conocimientos de visualización para poder crear una Función de Transferencia útil, es hacer el Rendering de Volúmenes de cientos de muestras que corresponden a un rango de Funciones de Transferencia automáticamente generadas. El usuario escoge las muestras más satisfactorias y de esta manera implícitamente escoge la Función de Transferencia más adecuada. Esta técnica se basa en el Diseño de Galerías de Marks *et al* ([Marks97]). El mayor reto consiste en crear Funciones de Transferencia que resulten en la mayor cantidad de muestras disímiles, tal como lo muestra He *et al* ([He96]) quien desarrolló un método de generación estocástico basado en Algoritmos Genéticos. Debido a que tienen que realizarse cientos de Renderings para que el usuario tenga más libertad al escoger una función de transferencia, este esquema requiere un sistema de visualización en tiempo real.

El Contorno Espectral (Spectrum Contour) de Bajaj *et al* ([Bajaj97]) consiste en calcular un conjunto de datos escalares y atributos de contorno basados en los datos del volumen y graficarlos como curvas. Cada curva representa un atributo diferente que asiste al científico al elegir iso-valores para la extracción de superficies y construir la función de transferencia.

El método de Kindlmann y Durkin ([Kindlmann98]) para la generación semi-automática de la Función de Transferencia, se basa en la premisa de que las características de interés para el científico son las fronteras que existen entre los materiales. En medicina resulta útil clasificar los datos encontrando estas fronteras. Haciendo uso de los datos del volumen y de su primera y segunda derivada direccional, el método de Kindlmann puede generar Funciones de Transferencia que hacen visibles las fronteras entre materiales en el Rendering Directo de Volúmenes.

Uno de los beneficios del método semiautomático de Kindlmann es su habilidad de encontrar funciones de transferencia que no sólo mapean escalares a colores y transparencias, sino que pueden mapear valores en el espacio bidimensional formado por los escalares en un eje y la primera derivada en el otro eje. Estas funciones de transferencia 2D pueden encontrar las fronteras entre materiales y son más difíciles de encontrar manualmente que las funciones 1D tradicionales.

3.4 Funciones de Transferencia Multidimensionales.

Las Funciones de Transferencia bidimensionales (2D), son funciones de la forma $f(s,s') = (c,\alpha)$, donde s son los escalares y s' es la primera derivada direccional (gradiente) calculada a partir de s. La primera derivada es útil para

descubrir (a través de histogramas) las fronteras entre materiales. Cuando la primera derivada alcanza un pico, estamos frente a un contorno o frontera:

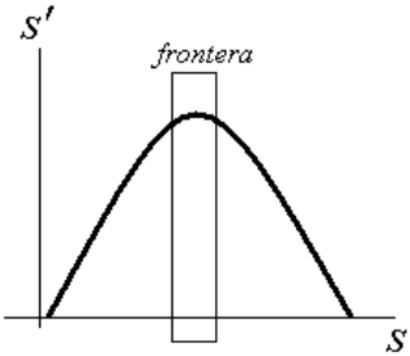

Figura 3: Los contornos (posibles fronteras) están en los máximos locales de la primera derivada.

En la práctica, sin embargo, estos arcos pueden ser más o menos pronunciados o incluso indistinguibles. Esto depende fundamentalmente de la calidad del Scanner que captó el volumen y del tipo de filtro que se usa para reconstruir los datos. Algunas veces resulta útil marcar las pendientes de los arcos con diferentes colores para poder distinguir donde están los máximos locales.

Los histogramas que usan derivadas (también llamados *Espectro*) pueden ser usados como guías. Las Curvas Espectrales de Bajaj *et al* ([Bajaj97]) corresponden a este tipo de histograma, donde el científico puede identificar los escalares de la frontera observando dónde la curva alcanza un pico. Kniss, Kindlmann y Hansen muestran una interfaz de usuario interesante para crear funciones de transferencia basándose en histogramas (Figura 4).

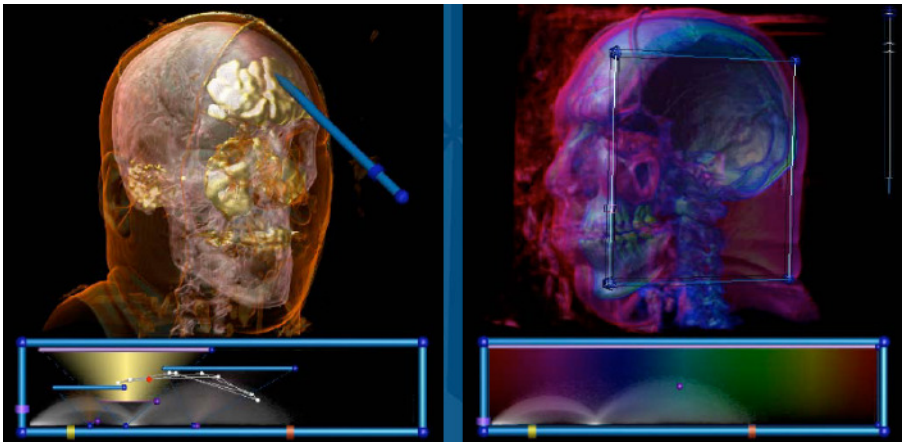

Figura 4: Interfaz para crear Funciones de Transferencia sobre un histograma.

Otro tipo de histograma tridimensional fue presentado por Kindlmann y Durkin ([Kindlmann98]) donde se observa la relación entre s, s' y s''. Observando la relación entre los escalares y su primera y segunda derivada, el científico puede identificar qué escalares representan las transiciones más importantes del tejido.

Las Funciones de Transferencia Multidimensionales aportan nuevas mejoras, ayudan a revelar características imposibles de revelar con Funciones de Transferencia 1D, ya que toman en cuenta la curvatura y transiciones presentes en los datos. Sin embargo, al ofrecer más grados de libertad, son más difíciles de manejar manualmente. Es por eso que gran parte de la investigación en Funciones de Transferencia Multidimensionales se enfoca en la generación automática o semiautomática de las mismas.

Las Funciones de Transferencia Multidimensionales no pueden encontrar las fronteras automáticamente cuando los datos son difusos. Actualmente este problema sólo puede ser resuelto a través de la segmentación del volumen, un proceso aun más largo y tedioso que la generación de la Función de Transferencia.

3.5 Pre-Integración de la Función de Transferencia

La Ecuación del Rendering puede ser aproximada numéricamente mediante la sumatoria de Riemann. Parametrizamos el volumen usando un rayo, el cual muestrea un segmento del volumen en una serie de incrementos regulares en una dirección determinada. El muestreo requiere algún tipo de filtro o interpolación para reconstruir el volumen en cualquier punto espacial. Anteriormente vimos que, para que el volumen pueda ser reconstruido exitosamente, es necesario que haya sido muestreado por encima de la frecuencia de Nyquist; de lo contrario se introducen defectos en la imagen final debido al ruido introducido por el sub-muestreo.

Las Funciones de Transferencia que poseen propiedades no lineales requieren que el volumen sea muestreado a una frecuencia más alta que la esperada. Engel *et al* ([Engel01], [Engel02]) observaron que la frecuencia de muestreo requerida para el volumen es en realidad un producto de la frecuencia de Nyquist en el espacio del volumen y la frecuencia de Nyquist en el espacio de las transparencias y colores de la Función de Transferencia.

La idea de la Pre-Integración es calcular la integración de la Función de Transferencia por separado antes de calcular la integración total del rayo usando la Ecuación del Rendering. De este modo, la Ecuación del Rendering no suma contribuciones de voxels, sino contribuciones equivalentes a pequeños bloques cilíndricos (llamados *slabs*) que poseen valores más precisos integrados previamente y guardados en la Función de Transferencia.

Un *slab* se define como el volumen que existe entre dos voxels que están separados cierta distancia. La integración de cada *slab* se precalcula usualmente usando filtros no lineales y además puede incluir voxels intermedios. Engel *et al* ([Engel01], [Engel02]), muestran que si la distancia entre voxels es constante, se puede generar una tabla Pre-Integrada bidimensional, la cual es accedida únicamente mediante dos escalares correspondientes a los voxels inicial y final. La Figura 5 ([Engel01]), muestra un *slab* comprendido entre dos voxels cuyos

escalares son *sf* y *sb*; las slices se utilizan para muestrear los escalares filtrando bi-linealmente el volumen y el rayo que parte del ojo parametriza la Ecuación del Rendering.

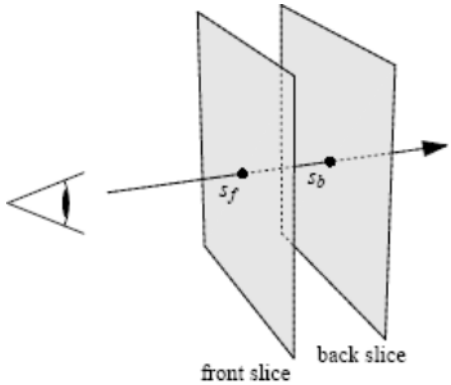

Figura 5: Un *slab* del volumen limitado por dos escalares *sf* y *sb* separados por la distancia entre slices front y back (Extraído de Engel *et al* ([Engel01]))

Si *sf* y *sb* son escalares correspondientes a voxels inicial y final respectivamente; y *d* es la distancia entre los voxels; la ecuación que aproxima la opacidad del *slab* comprendido entre los escalares *sf* y *sb* es:

$$\alpha(sf, sb, d) \approx 1 - \exp\left(-\frac{d}{sb - sf} \left(\int_0^{sb} \tau(sb) ds - \int_0^{sf} \tau(sf) ds \right) \right) \tag{10}$$

De manera similar, la ecuación que aproxima la contribución de color del *slab* comprendido entre *sf* y *sb* es:

$$c(sf, sb, d) \approx \frac{d}{sb - sf} \left(\int_0^{sb} C(sb) ds - \int_0^{sf} C(sf) ds \right) \tag{11}$$

De hecho, las ecuaciones (10) y (11) son aproximaciones de las expresiones analíticas presentadas en [Engel01] y [Engel02].

Si bien la Pre-Integración alivia los defectos causados por la linealidad de la Función de Transferencia sin disminuir la performance de los sistemas de visualización, su mayor defecto es el tiempo que toma generar la tabla Pre-Integrada. Los sistemas de visualización que utilizan la creación manual de la Función de Transferencia, necesitan modificarla interactivamente. Cada modificación de la función de transferencia implica recalcular la tabla Pre-Integrada, lo cual puede reducir significativamente la interactividad del proceso dependiendo de los cálculos de Pre-Integración que se realicen.

3.6 Discusión acerca de las Funciones de Transferencia

Actualmente, es necesario que los usuarios posean suficiente experiencia y conocimiento acerca de los datos así como de las técnicas de construcción de las Funciones de Transferencia para poder crear Funciones de Transferencia útiles. En general, los valores apropiados para las Funciones de Transferencia 1D y 2D se encuentran mediante prueba y error. Existe la necesidad de encontrar algoritmos e interfases que ayuden a los usuarios a encontrar Funciones de Transferencia adecuadas para cada conjunto de datos.

El diseño de galerías de Marks *et al* ([Marks97]) es una interfase que busca orientar al usuario mostrándole un conjunto de imágenes generadas utilizando Rendering de Volúmenes con diversas Funciones de Transferencia. No obstante es necesario contar con un sistema de rendering sumamente eficiente, el mayor reto consiste en generar funciones de transferencia que resulten en el mayor número de muestras disímiles. Otra interfase interesante pero basada en los datos es el Contorno Espectral de Bajaj *et al* ([Bajaj97]).

Kindlmann *et al* ([Kindlmann98]) propusieron una aproximación *semi-automática* para encontrar funciones de transferencia basadas en los datos las cuales resaltan las características de interés como pueden ser las fronteras entre materiales. Con este fin desarrollan las funciones de transferencia multidimensionales cuyas dimensiones están definidas por los escalares en un eje así como su primera y segunda derivada en dos ejes ortogonales entre si. Pese a su versatilidad, en muchos casos el aumento de dimensiones resulta también en el aumento de grados de libertad en la exploración dentro del espacio de la función de transferencia, lo cual puede terminar agravando el problema para los usuarios.♦

CAPÍTULO IV

TÉCNICAS DE RENDERING DIRECTO DE VOLÚMENES

Existe una cantidad importante de técnicas de *Rendering Directo de Volúmenes* que se desarrollaron en la última década. Sin embargo muchas son sólo variantes de técnicas básicas. En este capítulo, revisamos las técnicas más importantes, mostrando la formulación teórica con cierto detalle pero sin perder generalidad.

4.1 Ray Casting

Ray Casting ([Levoy90]) es una técnica simple y directa capaz de realizar DVR de alta calidad. Comparte muchas características con el Ray Tracing. En Ray Tracing miles de rayos transportadores de energía viajan a partir de una fuente luminosa hacia los objetos de la escena, haciéndolos visibles al promover una dinámica de transferencia de energía radiante. Los objetos reflejan parte de esa energía, de manera difusa y especular, y ésta finalmente llega al ojo del observador donde se percibe como colores. Análogamente, el Ray Casting emplea rayos explícitamente para transportar energía radiante en su recorrido a través del volumen de datos. A continuación revisamos dos técnicas de Ray Casting: la *Técnica Básica* que se aplica a volúmenes regulares y la *Técnica Avanzada* que se aplica a volúmenes *No Estructurados* (mallas de tetraedros).

4.1.1 Técnica Básica

El Ray Casting está basado en una entidad matemática conocida como *rayo*. Un *rayo* es un segmento de recta dirigido ([Watt01]). Un *rayo* puede ser definido por dos puntos o por un solo punto y un vector. Sea $\boldsymbol{P_0}$ el punto inicial de un rayo y $\boldsymbol{P_1}$ el final, entonces el *rayo* está definido por:

$$k\vec{V} + p_0 = p_1 \tag{12}$$

La ecuación (10) es conocida como *Ecuación del Rayo* ([Watt01]), donde k es un escalar positivo que representa la distancia euclidiana entre P_0 y P_1, y $\vec{V}$ es un vector unitario conocido como vector dirección.

La técnica consiste en situar al observador (o a la cámara) en un punto del espacio y lanzar cierta cantidad de rayos desde ese punto hacia el volumen. Se sitúa un Plano, llamado *Plano de Visión,* entre el observador y el volumen (Figura 6). Cada rayo lanzado interseca el *Plano de Visión* antes de atravesar el volumen. La intersección rayo/plano es la proyección en el plano de un segmento volumétrico. Después de lanzar cierto número de rayos, en el *Plano de Visión* se forma la proyección en perspectiva del volumen.

Durante su recorrido a través del volumen, el rayo es utilizado para resolver la Ecuación del Rendering. Es así que la Ecuación (12) es parametrizada en la Ecuación (13), en función de la distancia, donde $\boldsymbol{P_0}$ es el punto de entrada del *rayo* al volumen y $\boldsymbol{k}$ es la distancia que va incrementándose desde cero hasta que el rayo sale del volumen:

$$\vec{r}(k) = k\vec{V} + p_0 \tag{13}$$

Entonces, podemos parametrizar el volumen como el Campo Escalar $f(\vec{r}(k))$ y escribimos la Ecuación del Rendering como sigue ([Kniss02]):

$$I(D) = I_0 e^{-\int_0^D \tau(f(\vec{r}(t)))dt} + \int_0^D L(f(\vec{r}(s)))\tau(f(\vec{r}(s)))e^{-\int_s^D \tau(f(\vec{r}(t)))dt} ds \tag{14}$$

donde:

- $L(f(\vec{r}(s)))$ representa la emisión de luz de cada partícula a lo largo del rayo.
- $\tau(f(\vec{r}(s)))$ representa el *Coeficiente de Extinción* o la absorción de luz de cada partícula a lo largo del rayo.
- $e^{-\int_s^D \tau(f(\vec{r}(t)))dt}$ es la atenuación de luz debida a la obstrucción de otras partículas a lo largo del rayo.

El rayo permite muestrear el volumen a intervalos regulares y obtener los valores escalares en esas posiciones (Figura 6). Conforme el rayo atraviesa el volumen y se van muestreando los escalares se aproxima, acumulativamente, la Integral del Rendering usando la sumatoria de Riemann.

Krueger *et al* ([Krueger03]), Weiler *et al* ([Weiler03]), y Bernardon *et al* ([Bernardon04]) implementaron algoritmos de Ray Casting para hardware gráfico escritos como *shaders* que se ejecutan directamente en GPU. ***GPU Ray Casting*** es un algoritmo *multi-pasos* en el cual se trazan rayos en el primer paso y se *componen* colores en el segundo sin usar el CPU. Más recientemente, Stegmaier *et al* ([Stegmaier05]) muestran cómo aprovechar las capacidad de iteración de las tarjetas gráficas en *Estado del Arte* para codificar el *GPU Ray Casting* como un algoritmo de un solo paso.

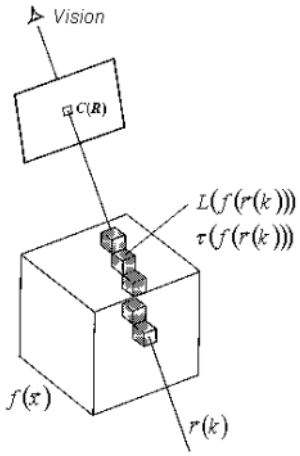

Figura 6: El rayo parte del ojo, atraviesa el Plano de Visión, luego atraviesa el volumen. Durante su recorrido, se toman muestras a intervalos regulares. Finalmente se integra la ecuación (14) y el color es puesto en el Plano de Visión.

Dos optimizaciones para Ray Casting fueron presentadas por Levoy ([Levoy90]) a principios de la década del 1990. Éstas son: *Terminación Temprana del Rayo* (Early Ray Termination) y *Saltar los Espacios Vacíos* (Empty Space Skipping). Más tarde, Yagel *et al* ([Yagel92]) proponen usar plantillas para *lanzar* rayos paralelos y en paralelo, pero su propuesta sólo se puede implementar con proyección ortogonal.

4.1.2 Técnica Avanzada

Garrity ([Garrity90]) propone un algoritmo de Ray Casting para mallas *Conformes* y *Convexas* de topología *No-Estructurada*. El algoritmo se basa en la información de conectividad disponible entre celdas para *trazar* eficientemente los rayos (Figura 7).

Garrity define una *Cara Externa* como un polígono perteneciente a una celda que no es adyacente a otra celda (es decir un polígono de la *Frontera*). Una *Celda Externa* es aquélla que posee al menos una *Cara Externa*. Así, Garrity clasifica polígonos y celdas como internos o externos y los guarda por separado en dos listas *Caras Internas* y *Caras Externas*.

Inicialmente, al lanzar los rayos, el algoritmo encuentra las primeras *Celdas Externas* que deben impactar valiéndose de la tabla *Caras Externas*, y calcula los puntos de entrada y salida de ellas: $\vec{r}(t_0)$ y $\vec{r}(t_1)$ respectivamente. Obtiene los escalares $f(\vec{r}(t_0))$ y $f(\vec{r}(t_1))$ en ambas posiciones usando interpolación. Luego integra los segmentos usando la Ecuación del Rendering (Figura 7). El algoritmo encuentra la siguiente *celda* basándose en la información de conectividad guardada en la tabla *Caras Internas* y repite el proceso hasta que encuentra nuevamente una *Celda Externa*.

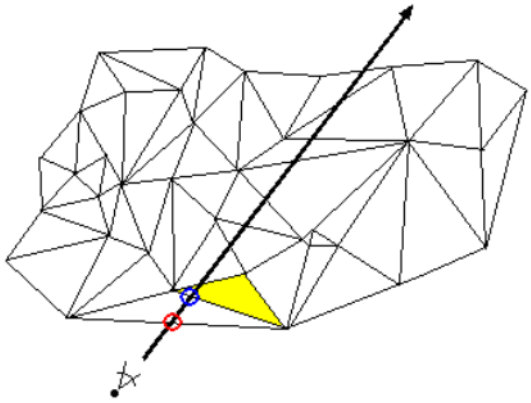

Figura 7: El Ray Casting calcula los puntos de entrada y salida del tetraedro e integra el segmento entre ellos. Luego usa la información de conectividad para encontrar el siguiente tetraedro que debe atravesar (tetraedro pintado).

Si asumimos que los escalares varían linealmente dentro de la *celda* y que los colores de la *Función de Transferencia* también varían linealmente, podemos interpolar linealmente el color y la opacidad dentro de ella ([Shirley90]). De este modo, la integración del segmento de rayo entre los puntos de entrada y salida de la celda estaría dada por:

$$I(D) = I_0 e^{-D(\tau_0+\tau_1)/2} + \frac{L_0 + L_1}{2}\left(1 - e^{-D(\tau_0+\tau_1)/2}\right) \tag{15}$$

Donde τ_0 y τ_1 son las transparencias, y L_0 y L_1 son los colores de los extremos del segmento de rayo que se integra. En muchos casos no resulta correcto asumir un cambio lineal dentro de una celda. Shirley *et al* ([Shirley90]) y Moreland ([Moreland04]) reportan graves defectos en el Rendering de Volúmenes por promediar colores y transparencias.

Weiler *et al* ([Weiler03], [Weiler04]) llevan estos conceptos a un algoritmo *multi-pasos* escrito en lenguaje de *shaders* para *GPU*. Weiler *et al* almacenan tanto tetraedros como sus conectividades en texturas volumétricas. Sin embargo Weiler utiliza una tabla *Pre Integrada,* almacenada como textura volumétrica, para evitar la interpolación lineal de colores y opacidades dentro de la *celda*.

Bernardon *et al* ([Bernardon04]) extienden el algoritmo de Weiler *et al* ([Weiler03]) a mallas *No Estructuradas* de tetraedros; esto permite trabajar con mallas que poseen concavidades (*No-Convexas*). Esta técnica se denomina *Depth Peeling* y es una técnica *multi pasos* que escrita enteramente en lenguaje de *shaders* para *GPU*. Consiste en detectar qué rayos abandonan la malla y calcular su contribución de color, luego comprobar la posibilidad que, siguiendo su recorrido, la vuelva a intersecar. Si el rayo encuentra una nueva intersección continúa su recorrido y repite el proceso. En caso de no encontrar más intersecciones retorna el color integrado. Adicionalmente, Bernardon *et al* subdividen el *Plano de Visión* en pequeños rectángulos o *tiles.* Cada *tile* lanza rayos independientemente, lo cual hace posible acelerar la terminación de rayos cortos o que no intersecan nada y eventualmente paralelizar el algoritmo.

4.2 Proyección de Celdas

La proyección de celdas es un conjunto de técnicas eficientes de Rendering Directo de Volúmenes enfocadas en la visualización de Mallas de Elemento Finito, es decir, mallas No Estructuradas que provienen de las ciencias puras y aplicadas. Estas técnicas procesan el volumen de datos como una secuencia ordenada de celdas (Stream). El orden de las celdas es dependiente de la posición del observador. Existen numerosas investigaciones que tratan solamente el problema de ordenamiento topológico de mallas de tetraedros; sin embargo, existen mallas que no poseen ordenamiento topológico alguno debido a que presentan oclusión cíclica de algunas celdas. En consecuencia la Proyección de Celdas es utilizada para visualizar volúmenes sólo cuando es posible encontrar el ordenamiento espacial de celdas.

En general, la proyección de celdas consiste en visitar cada *celda* del volumen y proyectarla sobre el *Plano de Visión*. Una vez proyectada, es dividida en polígonos a los cuales se les define color y opacidad en sus vértices. Los polígonos son entonces *rasterizados*[1] y pintados en la pantalla (Frame Buffer) donde se acumulan los colores usando el operador OVER de Porter *et al* ([Porter84]) (conocido como *Alpha Blending* o interpolación lineal de colores).

Si bien los algoritmos de Proyección de Celdas no lanzan rayos explícitamente tal como se hace en Ray Casting, éstos están implícitos en todos los cálculos y se basan en ellos para aproximar la *Ecuación del Rendering*.

Shirley *et al* ([Shirley90]) fueron los primeros en proponer un algoritmo de *proyección de celdas* para la topología No-Estructurada. Shirley propuso un algoritmo que proyecta tetraedros. Posteriormente Stein *et al* ([Stein94]) mejoraron el algoritmo de Shirley y mostraron cómo aprovechar el hardware acelerado para la proyección de tetraedros. Wilhelms *et al* ([Wilhelms91]) propusieron un algoritmo para la proyección de hexaedros y recientemente, Max *et al* ([Max03]) propusieron un algoritmo para la proyección de mallas de poliedros tales como tetraedros, cubos, prismas triangulares y prismas cuadrados.

4.2.1 Proyección de Tetraedros

El algoritmo de Rendering de Volúmenes de Shirley *et al* ([Shirley90]) trabaja sobre la base de un tetraedro a la vez, es decir, en secuencia. La secuencia debe estar previamente ordenada en *Back-To-Front* con respecto a la posición espacial del observador ya que los tetraedros se deben superponer y mezclar en la pantalla usando el operador OVER de Porter *et al* ([Porter84]). Algoritmos de

[1] *Rasterizar* significa "dividir en pedazos pequeños", en éste caso píxeles. El término *Raster* también es usado por algunos autores para referirse a la topología de Malla Uniforme, ya que ésta se asemeja a una Reja o a un Enrejado.

ordenamiento topológico de mallas de tetraedros, como por ejemplo el MPVO (Meshed Polyhedra Visibility Ordering) de Williams *et al,* son de orden lineal y trabajan en el espacio del objeto; sin embargo se ha probado que ciertas mallas no poseen orden topológico alguno; para ese tipo de mallas es necesario usar técnicas de ordenamiento en el espacio de pantalla (Ray Casting) o bien técnicas hibridas, como veremos más adelante.

El algoritmo PT recorre la secuencia ordenada de tetraedros y primero, desde la perspectiva del observador, clasifica el tetraedro según su clase (ver Figura 8). Luego proyecta el tetraedro sobre el *Plano de Visión* y computa ciertos parámetros en coordenadas de pantalla. Finalmente subdivide el tetraedro (proyectado) en tres o cuatro polígonos según su clase (Figura 8), asigna color y opacidad a sus vértices y envía los polígonos al motor gráfico para ser rasterizados y mezclados en el Frame Buffer usando el operador OVER (*Alpha Blending* ([Porter84])).

Como en cualquier técnica de Rendering Directo de Volúmenes se requiere aproximar la *Ecuación del Rendering* para cada longitud de onda luminosa que atraviesa cada celda (tetraedro) de la malla. Shirley *et al* reemplazan el trabajo de hacer un Ray Casting completo por tetraedro por lanzar sólo unos pocos rayos significativos, calcular su aporte de color usando la *Ecuación del Rendering* y luego interpolar los colores dentro del tetraedro con ayuda del hardware grafico Standard. El algoritmo PT lanza cinco rayos por tetraedro, cuatro rayos que deben pasar por los vértices del tetraedro y el quinto debe atravesar el tetraedro en su *parte más densa*. El algoritmo calcula el color (intensidad luminosa) y la transparencia integrando cada rayo, con la Ecuación (14) (Ecuación del Rendering Paramétrica). Asumiendo que los colores varían linealmente dentro del tetraedro, éste es proyectado y subdividido en triángulos que son rasterizados con *Sombreado de Gouraud* (interpolación lineal de color en la superficie del polígono) aprovechando así las capacidades de interpolación lineal del hardware grafico actual.

El punto más denso de un tetraedro corresponde a la parte percibida como más profunda y por lo tanto varía según la posición donde se ubique el observador; se trata entonces de un punto perceptual que coincide con un punto del tetraedro proyectado sobre el *Plano de Visión*. Ese punto está destacado con un círculo en la Figura 8. Es así que Shirley primero proyecta cada tetraedro con el fin de encontrar su *Punto Más Denso* (*Thickest Point*) sobre el plano para luego trasladar los cálculos nuevamente a coordenadas espaciales donde puede calcular la longitud del rayo.

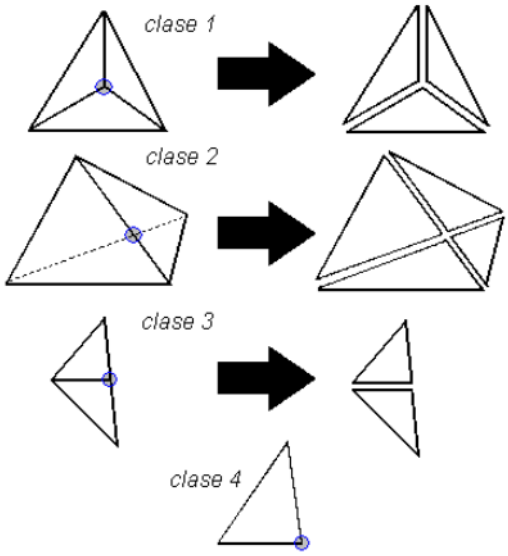

Figura 8: Clasificación de tetraedros según su proyección.

La *clase 1* de la Figura 8 presenta tres vértices en el contorno del tetraedro proyectado y el cuarto vértice coincide con el *Punto Más Denso* que es por donde el rayo principal atravesará el volumen. En contraste, la *clase 2* presenta los cuatro vértices en el contorno del tetraedro proyectado y el *Punto Más Denso* es calculado como la intersección de dos líneas. Calcular esa intersección es la razón principal de proyectar el tetraedro antes de lanzar el rayo. Las *clases 3 y 4* son casos degenerados de las *clases 1 y 2* como se ve en la Figura 8.

Shirley *et al* asumen que la densidad varía linealmente desde la parte más densa (*Punto Más Denso* en la Figura 8) hacia los límites del tetraedro. Debido a ello, los vértices y el contorno mismo del tetraedro poseen densidad cero. Esto implica que no existe oclusión de luz en el contorno pero sí una mínima emisión de luz necesaria para la interpolación, en otras palabras esos vértices poseen color pero sus opacidades son cero ($\alpha = 0$).

Para integrar la intensidad luminosa del segmento volumétrico que atraviesa el tetraedro por el *Punto Más Denso,* el algoritmo PT se basa en la parametrización del rayo tal como se hace en Ray Casting tradicionalmente. Es necesario calcular las intersecciones de entrada y salida del rayo al atravesar el tetraedro. Para la *clase 1* una intersección coincide con un vértice y sólo hay que calcular la otra. Para la *clase 2* la intersección de entrada se calcula sobre el *Plano de Visión,* en coordenadas de pantalla, como la intersección de 2 líneas (Figura 8) y la de salida se calcula en el espacio *lanzando* un rayo. Una vez calculadas ambas intersecciones se puede usar la Ecuación (15) para integrar el segmento ([Moreland04]):

$$I(D) = I_0 e^{-D(\tau_0+\tau_1)/2} + \frac{L_0 + L_1}{2}\left(1 - e^{-D(\tau_0+\tau_1)/2}\right)$$

Debido a que se instruye al hardware gráfico para que realice el Sombreado de Gouraud en la superficie de cada triángulo, se define el color RGB (llamado C) y la opacidad (llamada α) como:

$$C = (L_0 + L_1)/2 \quad \text{y} \quad \alpha = 1 - e^{-\Delta x(\tau_0 + \tau_1)/2} \tag{16}$$

donde Δx es la distancia cartesiana entre el punto de entrada y el de salida del rayo que atraviesa el tetraedro. La integración o *composición* final de color la hace el hardware gráfico. Debido a que procesamos los tetraedros en orden *Back-To-Front*, se instruye al hardware para que haga la composición de la siguiente manera:

$$C_i = C_{i-1}(1-\alpha) + C\alpha \tag{17}$$

siendo C_i el color acumulado (color final), C_{i-1} el color que estaba previamente en el Frame Buffer, C y α son el color y opacidad definidos.

Recientemente, Wylie *et al* ([Wylie02]) programaron el algoritmo de Shirley *et al* ([Shirley90]) en lenguaje de *shaders* para ejecutar enteramente en el *GPU*. Sin embargo debido a las limitaciones de ese momento en los lenguajes de *shaders*, tuvieron que definir un Grafo Base para clasificar los tetraedros y tratar con más casos que los presentados por Shirley (16 casos y las permutaciones de los mismos). De existir un aumento en la performance, no es significativo.

Weiler *et al* ([Weiler02]), en cambio, proponen un algoritmo diferente, también escrito enteramente en lenguaje de *shaders* para ejecutar en *GPU*, con performance superior. Weiler *et al* instruyen al hardware para descartar los polígonos *Back-Facing* y utilizar el *Procesador de Vértices* (*Vertex Shader*) para calcular los puntos de intersección de entrada y salida de cada rayo, los escalares en esos puntos y la distancia entre ellos. El hardware es capaz de interpolar tanto Colores como Coordenadas de Textura. Weiler *et al* usan las Coordenadas de Textura para interpolar los escalares y la distancia, en la superficie de los polígonos; luego utiliza el *Procesador de Pixels* (*Pixel Shader*) para asignar el color y la opacidad de cada pixel. Esto implica obtener los colores y opacidades de la *Función de Transferencia* e interpolarlos según la ecuación (16). El hardware se encarga de *componer* los colores usando la Ecuación (17). Weiler *et al* proponen el uso de tablas *Pre-Integradas* para no interpolar colores y opacidades.

En general, los algoritmos de proyección de tetraedros escritos en *shaders* usan tablas *Pre-Integradas* para obtener los colores; esto aumenta la calidad de las imágenes generadas ya que usualmente las tablas *Pre-Integradas* pueden precalcular mejores aproximaciones que la simple interpolación (promedio) propuesta por Shirley *et al* (Ecuación 16).

Moreland ([Moreland04]) extiende el algoritmo de Weiler *et al* ([Weiler02]) esencialmente para usar mejores aproximaciones de integración del color y la opacidad que la ecuaciones (15) y (16) sin usar tablas *Pre-Integradas*. Adicionalmente, propone una variación de su algoritmo que usa puntos de control

definidos en la *Función de Transferencia* para aproximar celdas con cambios no lineales dentro de ellas.

4.2.2 Proyección de Cubos, Prismas y otros poliedros.

Max *et al* ([Max00], [Max03]) generalizan el algoritmo **PT** de Shirley y Tuchman ([Shirley90]) para proyectar poliedros convexos cuyas caras son planas. Al igual que el algoritmo PT, Max *et al* asumen que la densidad varía linealmente dentro del poliedro proyectado y que es cero en los vértices que forman el contorno. Es así que su algoritmo se basa en *lanzar* rayos que atraviesen el poliedro en sus partes más densas, calcular sus contribuciones de luz e interpolarlas linealmente en el interior del poliedro.

Roettger y Ertl ([Roettger03]) también desarrollaron un algoritmo que proyecta poliedros convexos usando hardware gráfico. Se basan en restringir el modelo óptico a *Solo Emisión* y *Solo Absorción*. Por esta razón, su algoritmo sólo es útil para visualizar fenómenos gaseosos tales como fuego o niebla.

Al igual que Shirley y Tuchman ([Shirley90]), Max *et al* proyectan el poliedro sobre el *Plano de Visión* con el fin de calcular sus *Puntos Densos* en coordenadas de pantalla. Una vez calculados son trasladados a coordenadas espaciales con el fin de encontrar los puntos de entrada y salida de los rayos. Por ejemplo, consideremos la proyección del prisma triangular P=ABCDEF como se muestra en la **Figura 9**.

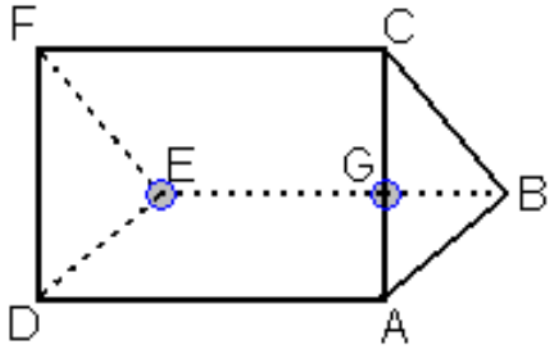

Figura 9: Proyección de un Prisma Triangular. Los *Puntos Densos* se destacan con círculos.

En la Figura 9, los vértices A, B, C, D y F pertenecen al contorno, por consiguiente su densidad es cero. Los puntos *E* y *G* son los *Puntos Densos* del poliedro y es necesario trazar rayos que pasen por ellos. El vértice E coincide con uno de esos puntos, mientras que el punto G es calculado como la intersección de dos líneas. Ambos están en coordenadas de pantalla y deben ser trasladados a coordenadas espaciales.

Así como en el algoritmo PT, los vértices que conforman el contorno sólo poseen color pero la opacidad es cero. Finalmente el poliedro es dividido en polígonos

tales como EFD, DEGA, etc. los cuales son rasterizados y sombreados con *Sombreado Gouraud* por el hardware gráfico. El hardware se encarga de componer los colores usando la Ecuación (15).

El mayor inconveniente de la Proyección de Celdas es no conocer de antemano cuántos *Puntos Densos* posee un poliedro, éstos pueden estar en una intersección o coincidir con un vértice. La solución de Max *et al* es compleja, lenta y sufre de errores debido al punto flotante; ésta consiste en ir añadiendo líneas a una lista y comprobar si existe una intersección entre ellas. Al encontrar una intersección se subdividen ambas líneas y se añade un nuevo vértice. En el siguiente paso se definen cuáles son vértices del contorno y cuáles vértices internos. Luego se encuentran los polígonos cerrados que se han formado.

4.3 Técnicas de DVR basadas en Mapeo de Texturas

Las técnicas basadas en Mapeo de Texturas derivan directamente del algoritmo Shear Warp concebido por Lacroute ([Lacroute95]). Shear Warp es un algoritmo muy eficiente que fue concebido para acelerar el Rendering De Volúmenes de datos provenientes de la ciencia médica que sólo podían ser visualizados con Ray Casting. Debido a esto, pierde generalidad para lidiar con mallas No Estructuradas. Las técnicas de Rendering de Volúmenes Basadas en Mapeo de Textura son la evolución natural del algoritmo Shear Warp, es decir, una adaptación que aprovecha el mapeo de texturas existente en el hardware gráfico actual.

4.3.1 Slices Alineadas al Objeto

Ray Casting lanza rayos desde el observador hacia el volumen; en cambio, las técnicas basadas en textura hacen el proceso inverso, es decir, proyectan el volumen en la pantalla. El principio es simple, se basa en dividir el volumen en *Rebanadas* conocidas como *Slices* y formar una pila con ellas (Figura 10), cada *Slice* es un polígono que tiene como textura los voxels del volumen mapeados y traducidos a colores y opacidades RGBA. El hardware gráfico proyecta y rasteriza los polígonos en orden *Back-To-Front* con respecto a la posición del observador y compone los colores en el Frame Buffer usando el operador OVER. Este tipo de geometría (pila se *Slices*) es llamada en la literatura *Geometría Proxy*.

El *muestreo* que ocurre al rasterizar los polígonos es en realidad un *remuestreo* ya que, como sabemos, el mapeo de texturas ya muestrea el volumen. El Mapeo de Texturas es, entonces, una forma de *remuestreo* del volumen por hardware. El hardware reconstruye el volumen mediante un filtro bilineal y luego lo vuelve a muestrear al mapear la textura en el polígono y pintarlo en la pantalla.

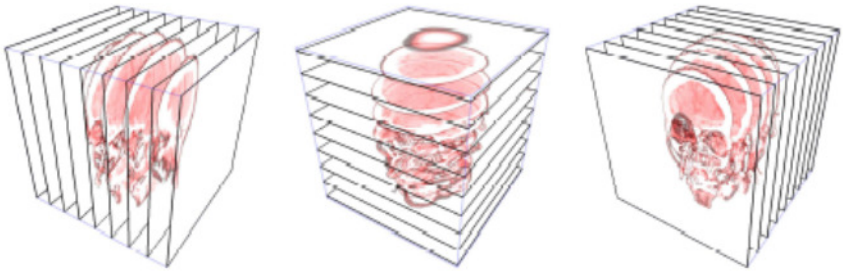

Figura 10: Volumen Regular dividido en *Slices*. El volumen de datos es un arreglo tridimensional de 256x256x256 generado por un *Scanner* de Tomografía Computarizada

Si usamos la proyección ortogonal y asumimos que la pila de *Slices* es paralela al Plano de Visión como se ve en la Figura 11, notamos que cada *Slice* se proyecta exactamente en la misma posición y todas ocupan la misma cantidad de píxeles en la pantalla. Es decir que los píxeles de cada *Slice* ubicados en la misma posición relativa se *mezclan* exactamente uno encima del otro. Esto equivale a lanzar rayos perpendiculares al Plano de Visión (Figura 11). Conforme el rayo virtual interseca cada *Slice*, calcula el punto de intersección y obtiene el escalar en ese punto mediante la interpolación bilineal de los escalares. Después mapea el escalar a color y opacidad y lo mezcla en la pantalla con *Alpha Blending*. Bajo estas condiciones las técnicas basadas en Mapeo de Textura, equivalen a hacer Ray Casting.

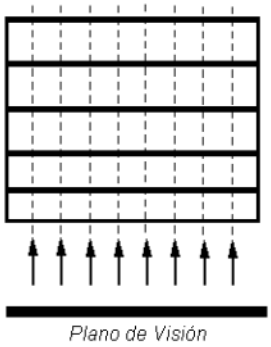

Figura 11: Al proyectar los *Slices* en el Plano de Visión implícitamente se lanzan rayos

Al dividir el volumen en una pila de *Slices* formada por un conjunto polígonos planos, estamos creando una estructura llamada *Object Aligned Slices* (*Rodajas Alineadas con el Volumen*) debido a que las *Slices* siempre están alineadas con el volumen pero no siempre con el Plano de Visión. Consideremos la rotación del volumen. Conforme aumenta el ángulo de intersección de los rayos con el volumen, se hace más evidente para el observador la separación que existe entre *Slices* (Figura 12).

En la Figura 12 se aprecia cómo intersecan los rayos al volumen rotado 45 grados. Se hace evidente entonces que si usamos una pila de *Rodajas Alineadas al Objeto* y rotamos el volumen 90 grados los rayos serían paralelos a las *Slices* y no existiría intersección alguna. Éste problema lo resolvió Lacroute ([Lacroute95]) creando tres pilas de *Slices* (Figura 10). Cada pila está alineada a una cara del volumen. Conforme varía el ángulo de intersección de los rayos se

debe escoger la pila que sea más perpendicular al plano de visión. Esto se hace examinando las componentes del *Vector de Visión*, por ejemplo: si la componente Z es la mayor, se escoge la pila alineada al plano XY. Sin embargo el cambio de pilas es frecuentemente visible para el usuario ([Kniss02]) debido a que el hardware no interpola los mismos valores en las mismas posiciones después del cambio de pila.

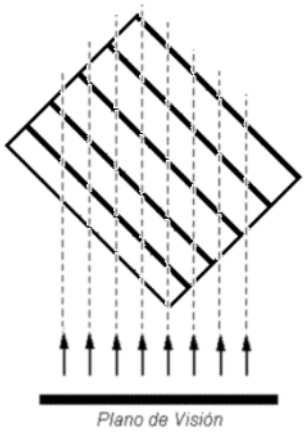

Figura 12: Se producen defectos en la imagen al rotar el volumen.

Los defectos producidos a causa del ángulo de incidencia con que el rayo interseca el volumen pueden ser mitigados si se aumenta el número de *Slices* en la pila ([Lacroute95], [Rezk-Salama00], [Engel01], [Kniss02]). Es posible aumentar el número de *Slices* interpolando *Slices* intermedias. Rezk-Salama *et al* ([Rezk-Salama00]) proponen un algoritmo para calcularlas *On-The-Fly* usando *Pixel Shaders* y *Multi-Texturas*.

Si asumimos que la distancia que existe entre *Slices* es constante y está dada por Δx podemos asumir que las intersecciones de los rayos con las *Slices* también son equidistantes como se muestra en la Figura 11. Según esto podemos calcular la opacidad (α) de cada voxel mapeado en la *Slice* y resolver la Ecuación de Rendering usando la Suma de Riemann (ecuaciones 8 y 9), tal que:

$$\alpha = 1 - e^{-\tau \Delta x} \tag{18}$$

donde la Absorción τ (también llamada *Coeficiente de Extinción*) la define el usuario, basándose en los datos del volumen, durante la construcción de la *Función de Transferencia*.

Sin embargo notamos en la Figura 12 que si los rayos intersecan el volumen con un ánguloθ, las distancias entre las intersecciones de los rayos con las *Slices* varían en proporción de *Sec(θ)* ([Rezk-Salama00]). El problema se agrava aún más si usamos Proyección en Perspectiva debido a que los rayos nunca son paralelos y la distancia Δx tiene variaciones aún si el ángulo θ es cero. No obstante, se puede corregir la opacidad dependiendo del ángulo de visión (sólo de manera aproximada) usando la siguiente expresión:

$$\alpha = \begin{cases} \theta <> 0 & \dfrac{1}{\cos(\theta)}(1 - e^{-\tau \Delta x}) \\ \theta = 0 & 1 - e^{-\tau \Delta x} \end{cases} \tag{19}$$

La corrección de opacidad (Ecuación (19)) es muy costosa ya que, por depender del ángulo de visión, no es posible precalcular la opacidad, lo cual implica calcular la Ecuación (19) varias veces por cada pixel pintado. Pese a esto, Rezk-Salama *et al* ([Rezk-Salama00]) reportan que los defectos causados por la variación de la distancia Δx son despreciables si se incrementa el número de *Slices* y se usa la proyección ortogonal.

Otros investigadores proponen usar *Cascarones de Esferas Concéntricas* (*Concentric Spherical Shells*) en lugar de *Slices* para el *muestreo* del volumen (Figura 13). Esto elimina completamente el problema para la Proyección en Perspectiva pero añade complejidad al algoritmo y sólo es posible implementarlo efectivamente usando Texturas Volumétricas.

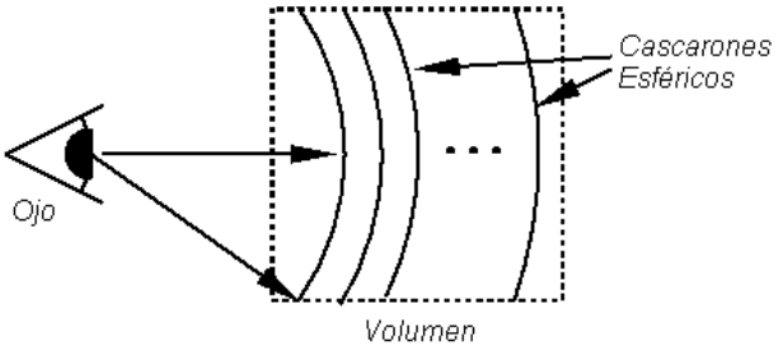

Figura 13: Volumen dividido en Cascarones Esféricos.

4.3.2 Slices Alineadas al Plano de Visión – Texturas Volumétricas

Cabral *et al* ([Cabral94]) propusieron el uso de texturas volumétricas para el remuestreo del volumen. Sin embargo, el rendering de volúmenes de Cabral *et al* no usaba hardware gráfico. Hoy en día las tarjetas gráficas en el *Estado del Arte* proveen texturas volumétricas.

El volumen necesita ser cargado íntegramente en la textura volumétrica. Similar a las texturas convencionales, cada escalar del volumen puede ser accedido usando coordenadas de textura (u,v,w). El volumen de datos es un Campo Escalar continuo dentro de la textura volumétrica y es accedido por coordenadas de textura continuas, es decir, cada componente está en el rango de cero a uno [0,1]. Para hacer esto el hardware emplea interpolación trilineal. Por esta razón las texturas volumétricas mejoran el muestreo del volumen. Si asignamos a una *Slice* coordenadas de textura (u,v,w) arbitrarias en cada uno de sus vértices, el

hardware interpola trilinealmente los escalares del volumen para mapearlos sobre la *Slice*. Esto permite orientar las *Slices* arbitrariamente con respecto al volumen.

Cabral *et al* orientan las *Slices* de manera que siempre estén alineadas con el Plano de Visión, es decir, paralelas a él (Figura 14). Si reorientamos el Plano de Visión es necesario modificar las *Coordenadas de Textura* de cada *Slice* para mantener la coherencia.

Después de reorientarlas, las *Slices* deben ser recortadas por los planos del AABB (*Axis Aligned Bounding Box*) que encierra al volumen. Este es un proceso de *clipping* en el espacio similar al que se hace con los planos del *Frustum de Visión* ([Watt01]). Luego del recorte, las *Slices* pueden tener más vértices; debido a esto las *Coordenadas de Textura* se calculan, para cada vértice, después del recorte.

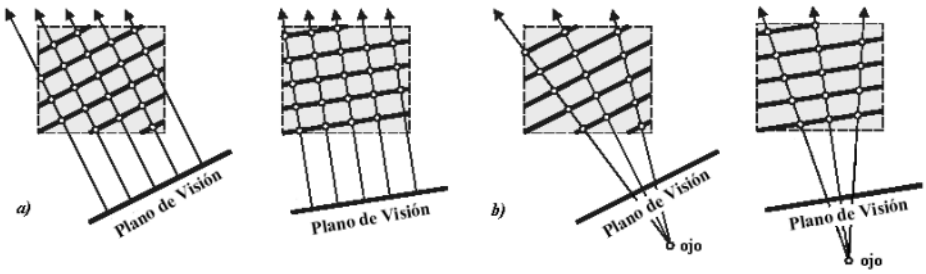

Figura 14: *Slices* alineadas al *Plano de Visión*. a) Proyección Ortogonal. b) Proyección en Perspectiva.

En la Figura 14 notamos que, usando *Slices Alineadas al Plano de Visión* y la Proyección Ortogonal, la distancia entre *Slices* se mantiene siempre constante, por tanto las ecuaciones de la Suma de Riemann (Ecuaciones (8) y (9)) son siempre correctas. Para la Proyección en Perspectiva, en cambio, la distancia no puede ser constante; sin embargo las Ecuaciones (8) y (9) aún son una buena aproximación. Otra opción es usar *Cascarones de Esferas Concéntricas* tal como se muestra en la Figura 13, lo cual elimina por completo el problema, asumiendo que las ecuaciones de la Suma de Riemann siempre sean correctas. Usar *Esferas Concéntricas* como geometría proxy suele ser más complicado y requiere procesar más polígonos ya que las esferas deben ser clipeadas por la caja que encierra al volumen y por el *Frustum de Visión* ([Kniss02]).

Teóricamente, el aumentar el número de *Slices* no implica ningún problema usando Texturas Volumétricas. Sin embargo, en la práctica, el hardware ejecuta un poco más lento con texturas 3D que con texturas 2D debido a que la interpolación trilineal es más costosa. Por tanto, se cumple la relación inversamente proporcional calidad / rendimiento ya que siempre que la calidad aumenta el rendimiento del sistema disminuye.

4.4 Splatting.

Splatting fue propuesta por Westover en 1989 ([Westover89]) para el Rendering Directo de Volúmenes de datos provenientes de la ciencia médica, es decir,

conformados en topologías Uniforme y Rectilínea. El remuestreo de los escalares se hace eficientemente por medio de un filtro de Gauss. Como sabemos, un filtro es representado por una matriz n-dimensional llamada *kernel* y se aplica mediante convolución ([Fisher03]). El kernel del filtro se centra en un voxel pero envuelve a varios voxels vecinos. Cuando se realiza el remuestreo, los valores intermedios entre voxels se calculan como la suma de las superposiciones de kernels centrados en voxels vecinos. El algoritmo recorre cada voxel del volumen en orden *Back-To-Front* y calcula la "huella" o *splat* bidimensional que surge de aplicar el kernel de convolución centrado en cada voxel y proyecta la huella en el *Plano de Visión*. Es en el *Plano de Visión* donde las huellas finalmente se superponen y se combinan para formar una imagen.

Debido a que la superposición de huellas ocurre en el *Plano de Visión*, la aplicación de propiedades ópticas a la señal mediante la *Función de Transferencia*, debe hacerse antes de proyectar la huella en el *Plano de Visión*. La superposición de colores produce algunos defectos visuales; la imagen, por ejemplo, se ve borrosa y, en algunas partes, el color se esparce demasiado produciendo saturación del mismo.

En el transcurso de los años se han realizado muchas mejoras a la técnica original de Westover, se han suprimido la mayoría de los defectos visuales y se ha mejorado mucho su eficiencia. Crawfis y Max ([Crawfis93]) propusieron acelerar la técnica usando polígonos texturizados como huella y más recientemente, Neophytou y Mueller ([Neophytou05]) lograron programar la técnica usando lenguaje de *shaders* para *GPU*.

4.5 Técnicas Hibridas

Consideramos el algoritmo HAVS (Hardware-Assisted Visibility Sorting) una técnica híbrida, ya que es una técnica en el espacio del objeto en su primera fase y una técnica en el espacio de la imagen en la segunda. Se ocupa del ordenamiento topológico aproximado de celdas como parte inicial del problema para luego centrarse en el uso del K-buffer para completar el Rendering de Volúmenes.

En su primera fase, el algoritmo trabaja con una secuencia de triángulos ordenada con respecto a la posición espacial del observador. El orden de la secuencia de triángulos sólo garantiza un orden aproximado de los tetraedros a los que pertenecen. El orden total y perfecto se logra con la segunda fase, en el espacio de pantalla, usando el K-buffer. Después de rasterizar k triángulos semi ordenados, en el K-buffer quedan guardados k escalares por píxel y k distancias Z por píxel. Cuando el buffer se encuentra lleno, antes de insertarle una nueva entrada, se extraen dos escalares, los que poseen la menor distancia y, se calcula la intensidad del segmento mapeando colores y opacidades mediante la Función de Transferencia. Luego se integran esas intensidades en el Frame Buffer componiendo colores con *Alpha Blending*.

4.5.1 Hardware-Assisted Visibility Sorting

Un antiguo problema en Computación Gráfica es producir primitivas (generalmente polígonos) ordenadas para representar correctamente una escena. Inicialmente se usaba el *Algoritmo del Pintor* para componer primitivas ordenadas en *Back-To-Front*, sin embargo tal algoritmo no funciona bien cuando existen polígonos que se superponen cíclicamente. Tradicionalmente los *Buffers* resuelven este problema. Estos son usados en Computación Gráfica para remover superficies ocultas y ordenar las primitivas gráficas luego que son rasterizadas; de este modo se pueden enviar primitivas en orden arbitrario a la librería gráfica. Se han investigado varias maneras de usar eficientemente los *Buffers* dando lugar a implementaciones tales como *Z-Buffer para* remover superficies no visibles, el *A-Buffer* para ordenar y componer primitivas semitransparentes en el espacio de pantalla y el *K-Buffer* derivado del *A-Buffer*, entre otros.

Debido a su flexibilidad y simplicidad, los fabricantes de hardware han incorporado algunas técnicas basadas en *Buffers* en sus productos. Se destacan comúnmente el *Stencil Buffer* y el *Z-Buffer*.

Algunos investigadores (i.e. [Farias01]) propusieron algoritmos que usan el *A-Buffer* en el contexto del *Rendering Directo de Volúmenes* con el fin de ordenar celdas que ya estaban "casi" ordenadas. Tales algoritmos son híbridos que primero ordenan celdas en el espacio pero sin obtener un orden exacto (por ejemplo usando la distancia desde el centroide al ojo) y luego obtienen el orden exacto en coordenadas de pantalla usando *A-Buffer* o *R-Buffer*.

Recientemente Callahan *et al* ([Callahan05]) desarrollaron el algoritmo HAVS (Hardware Assisted Visibility Sorting) que usa el *K-Buffer* no sólo para ordenar fragmentos, sino también para efectuar el *Rendering Directo de Volúmenes*. Callahan *et al* usan el hardware casero en el *Estado del Arte* para implementar una versión del *K-Buffer* usando lenguajes de *shaders* y reporta una eficiencia superior a otras técnicas como *Proyección de Celdas* y *Ray Casting*.

La implementación Callahan *et al* es sólo para tetraedros. Ordena los polígonos de cada tetraedro en *Front-To-Back* usando la distancia desde el centroide del polígono a la cámara. Para este primer ordenamiento usan una versión modificada del algoritmo Radix Sort que puede trabajar con números en punto flotante. Después de este primer ordenamiento obtienen un conjunto de polígonos "casi" ordenados, es decir, con un porcentaje de error debido al uso del centroide como representación de todo el polígono. Luego la librería gráfica procesa los polígonos. El proceso que hace la librería grafica se divide en dos fases, el proceso de vértices y el proceso de fragmentos.

En el procesador de vértices (*Vertex Shader*) se transforma cada vértice a coordenadas de visión y se calcula la distancia desde el ojo hasta él. Luego se transforma a coordenadas de pantalla y se envía al *Rasterizador* junto con la distancia calculada y el escalar (densidad) asociados a él. El *Rasterizador* interpola linealmente esos valores en la superficie del polígono. Finalmente un procesador de fragmentos (*Pixel Shader*) accede a cada pixel y es ahí donde se

implementa el *K-Buffer* para ordenar los fragmentos. El *K-Buffer* mantiene, por cada pixel, una lista de k distancias, de manera que, cuando un fragmento es procesado, primero es insertado en el *K-Buffer*. Luego, dos fragmentos (los de menor distancia) son extraídos del *KBuffer*. Se extraen también los escalares de cada fragmento y se transforman a Colores y Atenuaciones mediante la *Función de Transferencia.* Tanto el Color como la Opacidad finales son calculados usando la Ecuación (16), donde Δx es la diferencia entre las distancias de ambos fragmentos. Finalmente son integrados en el Frame Buffer con la ecuación (17) (*Alpha Blending*).

$$C = (L_0 + L_1)/2 \quad \text{y} \quad \alpha = 1 - e^{-\Delta x(\tau_0 + \tau_1)/2} \tag{16}$$

$$C_i = C_{i-1}(1-\alpha) + C\alpha \tag{17}$$

Callahan *et al* usan una tabla *Pre-Integrada* en su implementación, lo cual mejora la calidad y reemplazan el calculo de la ecuación (16) por un acceso a la tabla.

Callahan *et al* reportan ciertos defectos en las imágenes, principalmente producidos por problemas con la tecnología con la que se implementa el *K-Buffer* (actualmente su algoritmo sólo funciona en las tarjetas ATI Radeon9800 y sus resultados son experimentales) y también por defectos que se deben a las limitaciones en la implementación del *K-Buffer* ya que solo puede guardar un número pequeño de fragmentos (k=6). Además, no funciona correctamente cuando la malla posee concavidades (huecos). Estos problemas solo se superarán totalmente con la siguiente generación de hardware gráfico.

4.6 Discusión de los métodos revisados

La Proyección de Celdas es la técnica de rendering más popular. En particular, la Proyección de Tetraedros. Debido a que aproxima linealmente el valor de los atributos dentro de los tetraedros es posible utilizar la capacidad de interpolación de las placas de video modernas. Aún cuando posee limitaciones de precisión y sobretodo estabilidad numérica al calcular los escalares sobre las proyecciones de los tetraedros y además requiere que éstos posean un orden espacial perfecto, su implementación posibilita el rendering de volúmenes de gran tamaño en tiempos interactivos.

Ray Casting es la técnica de rendering directo de volúmenes más general, no requiere de ningún tipo de ordenamiento espacial de celdas y por lo tanto se puede aplicar a redes no convexas de cualquier topología. Sin embargo al ser una técnica en el espacio de la imagen, su eficiencia depende no sólo del número de tetraedros, sino también de la resolución de la pantalla. Recientemente ha sido implementada en lenguaje de shaders para GPU. Esta nueva implementación, si bien sufre de imprecisiones al encontrar las intersecciones de los rayos con el volumen, permite realizar el rendering en tiempos interactivos, no obstante está

limitada a redes tetraédricas que no exceden la capacidad de memoria de las placas de video modernas.

Los algoritmos híbridos como son, por ejemplo HAVS y Z-SWEEP requieren, en primera instancia, el ordenamiento espacial aproximado de los polígonos que componen la red volumétrica, luego utilizan una implementación particular del A-Buffer para realizar el ordenamiento total de los fragmentos en el espacio de la imagen. Teniendo los fragmentos almacenados en el A-Buffer, éstos se integran de forma idéntica a como se hace con los rayos del algoritmo Ray Casting. Estos algoritmos son más generales que los algoritmos de Proyección de Celdas y a la vez son más eficientes que Ray Casting. El algoritmo HAVS, es el primero que utiliza una versión del A-Buffer programada en shaders para GPU, llamada k-Buffer. Esta implementación permite realizar el rendering de volúmenes de gran tamaño en tiempos interactivos, gozando de la generalidad y calidad de la técnica Ray Casting pero sin las limitaciones de memoria que ésta técnica posee.

Las técnicas basadas en Mapeo de Texturas son aplicables sólo a volúmenes de topología regular representados como un conjunto de voxels. Debido a que utilizan la capacidad interpoladora de las placas de video, tanto las aproximaciones que usan texturas bidimensionales como las que usan texturas volumétricas permiten el rendering de volúmenes en tiempos interactivos. A pesar de las imprecisiones producidas cuando se utiliza proyección en perspectiva suelen producir imágenes de buena calidad. La mayor limitación de estas técnicas es su dependencia de la cantidad de memoria que poseen las placas de video modernas.

Splatting es también una técnica aplicable sólo a volúmenes de topología regular representados como un conjunto de voxels. Es similar a las técnicas de Proyección de Celdas en el sentido de que los voxels del volumen son proyectados en el plano de visión. Recientemente las técnicas basadas en puntos tales como Point Splatting, que usan un filtro elíptico gaussiano EWA (Elliptical Weighted Average), pueden usarse tanto en el rendering de volúmenes regulares como en el de irregulares. Point Splatting es una técnica que usa elipses semi-transparentes como representación de los vértices de cualquier tipo de triangulación. Esto habilita su uso para el rendering tanto de superficies como de volúmenes. Debido a la generalidad que brinda una primitiva basada tan solo en puntos, la cual no requiere el almacenamiento de ningún tipo de adyacencia, esta clase de rendering es actualmente un campo muy activo de investigación. ♦

CAPÍTULO V
MULTIRRESOLUCIÓN DE VOLÚMENES

Se han planteado varios modelos matemáticos, geométricos y de datos que dan soporte a la multirresolución de volúmenes; algunos de ellos son extensiones de modelos exitosos planteados para la multirresolución de superficies. Muchas representaciones multirresolución se crearon en base a un método de simplificación / refinamiento geométrico subyacente, dando lugar a familias de representaciones multirresolución. Podemos clasificar las técnicas multirresolución de diversas formas. Nosotros adoptamos la clasificación de Borgo *et al* ([Borgo04]) que consiste en sólo dos clases: *Regular* e *Irregular*, según el tipo de simplificación / refinamiento que se utilice. A su vez las técnicas de simplificación / refinamiento se basan en una variedad de métricas de error para generar los modelos entre las que destacan: *Error Geométrico (Geometric Error o Domain Error)*, *Error de Campo (Field Error)* y *Error en Espacio de Pantalla (Screen Space Error)*. El *Error Geométrico* mide el cambio en la forma geométrica del volumen, el *Error de Campo* mide el cambio en los escalares del dominio y el *Error en Espacio de Pantalla* mide directamente las variaciones que existen a nivel de píxeles de la imagen. En este capítulo introducimos los conceptos teóricos relacionados con la representación y manejo de volúmenes multirresolución *Regulares* e *Irregulares*, así como las métricas de error Geométrico y de Campo frecuentemente utilizadas en volúmenes. Para una revisión completa remitimos al lector a [DeFloriani97], [Borgo04], [Garland97], [Cignoni00], [Cignoni03], [Cignoni94], [Castro06], [Bank83], [Bey95], [Pascucci02], [Hoppe96], [Popovic97].

5.1 Métricas de Error Geométrico

5.1.1 Cuádricas de Error

La métrica de Cuádricas de Error ([Garland97]) mide el *Error Geométrico* de una malla. Se basa en la métrica de Ronfard y Rossignac ([Ronfard96]) que consiste en asociar una serie de planos a cada vértice de la malla y calcular el error basado en la distancia euclidiana que existe desde cada vértice a los planos. En la Figura 15 vemos una configuración clásica: un vértice compartido por varios triángulos, los cuales forman una *variedad*.

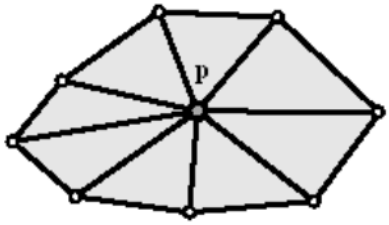

Figura 15: El vértice *p* está en una *variedad*. El error por defecto es cero, ya que todos los planos pasan por *p*.

La siguiente ecuación calcula el error en el punto ***p*** con respecto a un plano cualquiera cuya normal es ***N*** y cuya distancia ortogonal al origen es ***D***.

$$e = (N \bullet p + D)^2 \tag{20}$$

En la Figura 15 el error total del vértice ***p*** está dado por la suma de los errores que tiene con respecto a cada plano. Esto se ve reflejado en la siguiente ecuación:

$$e(p) = \sum_{i=0}^{np} (N_i \bullet p + D_i)^2 \tag{21}$$

El error del vértice ***p*,** mostrado en la Figura 15, es cero ya que los planos incidentes en ***p*** lo contienen y por tanto la distancia a cada uno de ellos es cero.

Garland y Heckbert ([Garland97]), transforman la Ecuación (21) en una forma compacta que hace realmente práctica la métrica. En la notación de Garland *et al* debemos asumir que el vector **n** (la normal del plano) es un vector columna (matriz de 3x1) y que el vector $\mathbf{n}^T$ es un vector fila **(**matriz de 1x3). Con todo esto, la Ecuación (20) se puede reescribir de la siguiente manera:

$$\begin{aligned} e &= \left(n^T p + d\right)^2 = p^T (nn^T) p + 2(dn)^T p + d^2 \\ e &= p^T (nn^T) p + 2(dn)^T p + d^2 \end{aligned} \tag{22}$$

Los términos de la Ecuación (22) se pueden reemplazar en la notación de Garland *et al* por:

$$\begin{aligned} A &= nn^T = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{bmatrix} \\ b &= dn \\ c &= d^2 \end{aligned} \tag{23}$$

Donde la matriz **A** es simétrica y todos sus coeficientes pueden almacenarse en **10** números en punto flotante. Según esto, Garland *et al* definen la *Cuádrica Fundamental* Q como la 3-tupla:

$$Q = (A, b, c) \tag{24}$$

La Ecuación (22) puede escribirse como:

$$Q(p) = p^T Ap + 2b^T p + c \tag{25}$$

La propiedad fundamental de la *Cuádrica* de Garland *et al* es la adición cerrada, es decir, la suma de dos *Cuádricas* es otra *Cuádrica*. La siguiente ecuación corresponde a la adición de *Cuádricas*:

$$Q_i + Q_j = (A_i + A_j, b_i + b_j, c_i + c_j) \tag{26}$$

Con esta propiedad podemos reescribir la Ecuación (21) de la siguiente manera:

$$e(p) = \sum_{i=0}^{np} \left(N_i \bullet p + D_i\right)^2 = \sum_{i=0}^{np} Q_i(p) \tag{27}$$

Como ejemplo del uso de esta métrica podemos ver el caso sencillo de la simplificación de superficies mediante el *Colapso de Arista*. La Figura 16 ilustra el colapso de la arista (*a*, *b*) en el vértice *p* (el punto medio). El error introducido por el colapso de arista es proporcional a la suma de las distancias de *p* a los planos incidentes en *a* y *b*.

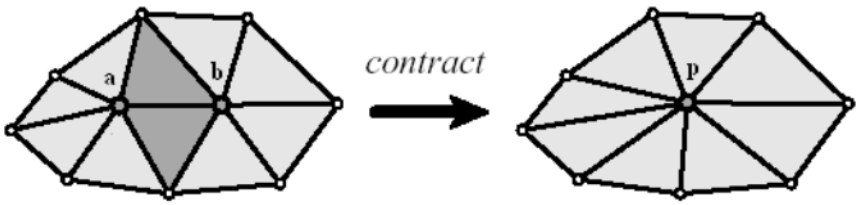

Figura 16: Colapso de la arista (*a*, *b*) en el vértice *p*.

En la métrica de *Cuádricas de Error*, al colapsar la arista (*a*, *b*), la *Cuádrica* resultante es $Q = Q_a + Q_b$ y el error en el vértice *p* es $Q(p) = Q_a(p) + Q_b(p)$. Los planos de los triángulos que se ven oscuros en la Figura 16 se duplican en la *Cuádrica* resultante.

5.1.1.1 Cuádricas Ponderados por Área (Area Weighted Quadrics)

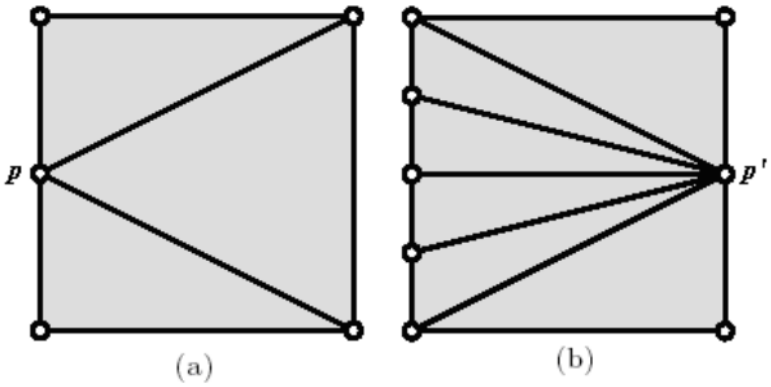

Figura 17: a) cuadrilátero plano formado por tres triángulos. b) cuadrilátero plano formado por seis triángulos. A pesar que ambos cuadriláteros tienen la misma área, la Cuádrica resultante para el vértice *p* difiere de la Cuádrica resultante para el vértice *p'*.

En la Figura 17 vemos dos cuadriláteros planos que tienen la misma forma y área pero tienen diferente número de triángulos, por lo tanto las *Cuádricas* resultantes para el punto ***p*** y para el punto ***p'*** difieren. Para uniformizar las *Cuádricas*, Garland sugiere ponderarlas por un peso ***w*** que, por defecto, es el área de cada triangulo:

$$\begin{aligned} Q &= (wA, wb, wc) \\ Q(p) &= p^T (wA) p + 2(wb^T) p + (wc) \end{aligned} \tag{28}$$

Más generalmente, Garland escribe la Ecuación (28) como:

$$e(p) = \sum_{i=0}^{np} w_i Q_i(p) = \left(\sum_{i=0}^{np} w_i Q_i \right)(p) \tag{29}$$

5.1.1.2 Cuádricas Homogéneas

Se trata simplemente de expresar las *Cuádricas* como una sola matriz extendida de 4x4, la cual pueda aplicarse directamente a los vértices en coordenadas homogéneas.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^{\mathsf{T}} & c \end{bmatrix} \tag{30}$$

Dado un vértice en coordenadas homogéneas $\bar{v}$, el error en dicho vértice estaría dado por:

$$Q(\bar{v}) = \bar{v}^T Q \bar{v} \tag{31}$$

La *Cuádrica Pesada Por Área* se calcula escalando la matriz extendida por un peso ***w***: wQ.

5.1.1.3 Posición Óptima del Vértice.

La métrica de *Cuádricas de Error* es usada frecuentemente para *optimizar* la simplificación geométrica de superficies y volúmenes. La optimización consiste en minimizar el error introducido por una operación de simplificación, por ejemplo el *Colapso de Arista*. La Figura 18 muestra el colapso de la arista (*a*, *b*) en el vértice *p*, tal que la posición de *p* minimiza el error geométrico introducido por el *Colapso de Arista*.

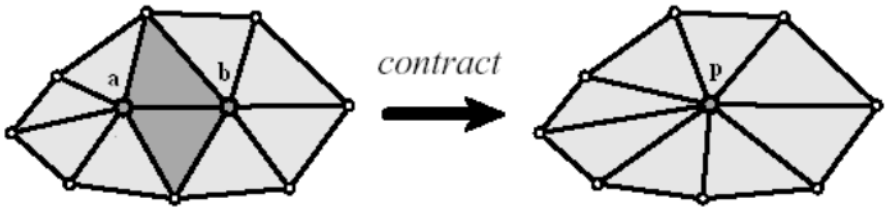

Figura 18: Colapso de la arista (*a*, *b*) en el vértice *p*.

La optimización consiste en calcular la posición óptima del vértice **p** tal que minimice el error dado por la *Cuádrica* resultante en $Q(p)$. Dado que $Q(p)$ es una expresión cuadrática, hallar el mínimo es un problema lineal: El mínimo ocurre cuando las derivadas parciales son iguales a cero.

$$\frac{\partial Q}{\partial x} = \frac{\partial Q}{\partial y} = \frac{\partial Q}{\partial z} = 0 \tag{32}$$

Calculando las derivadas parciales, el gradiente de Q es:

$$\nabla Q(p) = 2Ap + 2b \tag{33}$$

Resolviendo para $\nabla Q(p) = 0$

$$\begin{aligned} 2Ap + 2b &= 0 \\ p &= -A^{-1}b \end{aligned} \tag{34}$$

En la Ecuación (34), la matriz **A** puede ser singular, es decir, la inversa de **A** puede no existir. Otro problema con la optimización es que el punto ***p*** óptimo no necesariamente resulta estar en la superficie simplificada, sino que, debido a errores numéricos, puede ubicarse en cualquier lado. Los algoritmos de simplificación deben tener en cuenta estas anomalías para alcanzar la robustez.

5.1.2 Error Global

La *Distancia de Hausdorff* es una de las métricas más conocidas para comparar geométricamente dos triangulaciones, ya sea de superficies o volúmenes. La métrica se basa en medir la distancia euclidiana entre un punto ***p*** y un conjunto de puntos ***M***:

$$d_p(M) = \min_{w \in M} \| p - w \| \tag{35}$$

Donde $\| \cdot \|$ es la "distancia" o norma del vector. El conjunto de puntos *M* puede ser la parametrización de una superficie o un volumen.

La *Distancia de Hausdorff* entre dos conjuntos de puntos M_1 y M_2 (i.e. dos superficies o volúmenes) es la desviación máxima entre ambos conjuntos. Dados los conjuntos M_1 y M_2, hallamos la distancia del punto más alejado de M1 a M2 y viceversa y de ellas dos, la mayor:

$$hausdorff(M_1, M_2) = \max\left(\max_{v \in M_1}(d_v(M_2)), \max_{v \in M_2}(d_v(M_1)) \right) \tag{36}$$

Esta métrica es útil para comparar dos mallas y medir el máximo error geométrico de una con respecto a la otra, por ejemplo, de una malla a máxima resolución versus otra con menor resolución. En la práctica, evaluar la *Distancia de Hausdorff* es una tarea computacionalmente intensa. Como es de esperar, se usan las representaciones discretas de los volúmenes M_1 y M_2 interpolando linealmente los puntos que no coinciden con algún vértice.

El error promedio de Hausdorff, es otro cálculo útil que nos brinda información de error global aproximado de la malla:

$$hausdorffAvg(M_1, M_2) = \frac{1}{w_1 + w_2}\left(\int_{M_1} d_v^2(M_2)dv + \int_{M_2} d_v^2(M_1)dv \right) \tag{37}$$

donde w_1 y w_2 son los volúmenes de las mallas M_1 y M_2. De igual modo que con el error máximo de Hausdorff, en la práctica, las integrales de la Ecuación (37) se evalúan como sumas discretas de las distancias euclidianas.

Existen diferentes formas de tomar las distancias entre las mallas; entre las mas usuales se encuentra la correspondencia vértice – vértice, cuya ventaja es su simplicidad y eficiencia, y también la correspondencia vértice – superficie, que es la proyección de un vértice sobre una superficie. Ésta proyección se encuentra intersecando un rayo, que se inicia en el vértice y sigue la dirección del vector normal, con la superficie. Es posible utilizar otras correspondencias como, por ejemplo, la correspondencia superficie – superficie; pero en estos casos, se hace a través de un mapeo que ayuda a determinar la correspondencia de los puntos de una superficie con los puntos de la otra superficie.

5.2 Métricas de Error de Campo

5.2.1 Diferencia de Gradientes

Para calcular el error introducido en el campo escalar por una modificación de simplificación al volumen, Cignoni *et al* ([Cignoni00]) proponen pre-calcular el gradiente para cada vértice del campo escalar. Por ejemplo, el gradiente en el vértice $\boldsymbol{x}$ se calcula como el promedio ponderado de los gradientes de los tetraedros incidentes a $\boldsymbol{x}$. El peso asociado a la contribución de cada tetraedro está dado por el ángulo sólido α del tetraedro en el vértice $\boldsymbol{x}$. El error introducido en el campo escalar por un Colapso de la arista (a, b) en el vértice p esta dado por la diferencia de los gradientes en los vértices a y b multiplicados por la distancia euclidiana entre ambos vértices:

$$e(p) = \|a - b\| \times [\nabla f(a) - \nabla f(b)] \tag{38}$$

Cignoni *et al* ([Cignoni00]) utilizan esta métrica para minimizar el error de campo. Su metodología consiste en escoger el Colapso de Arista que introduce el menor error. A esto lo denomina *predicción*.

5.2.2 Cuádricas de Error

Cignoni *et al* ([Cignoni00]) proponen usar *Cuádricas* para medir el error de campo en una malla de tetraedros. La idea fundamental en este caso es, en lugar de asociar a cada vértice de la malla un conjunto de planos, se le asocia un conjunto de funciones lineales. Las funciones lineales corresponden a parametrizaciones de las celdas que son incidentes en cada vértice. Cignoni *et al* notan que podemos representar la función lineal φ usando la siguiente notación:

$$\varphi(v) = n^T v + d \tag{39}$$

Donde n es un vector 3D no unitario que contiene el gradiente de la celda y d es una constante escalar que denota el valor del campo en el origen de la celda. Con esta representación, análoga al caso geométrico, se puede construir una *Cuádrica* para cada función lineal. De ese modo cada vértice acumula *Cuádricas* de igual manera a como se describe en [Garland97] (Sección 5.1.1).

La *Cuádrica* planteada por Cignoni *et al* representa la diferencia entre los escalares medidos y las funciones lineales establecidas para la malla a mayor resolución. Esta diferencia es cero para la malla original y aumenta conforme se pierde resolución en los datos que representa el volumen, es decir, el campo escalar.

5.3 Modelado Multirresolución Para Símplices

Representar datos en varias resoluciones no es simple. Es necesario organizarlos de manera eficiente, flexible y compacta para poder extraer información coherente en tiempos interactivos. La información es recolectada durante el proceso de simplificación / refinamiento del volumen; por ejemplo, un volumen que es simplificado iterativamente reemplaza constantemente un conjunto de celdas detalladas por un conjunto más simple. El conjunto de celdas detalladas se guarda como una modificación atómica dentro de una jerarquía. La jerarquía de modificaciones se usa posteriormente para reconstruir el modelo original, dando lugar a modelos intermedios con niveles de detalle adaptados a ciertas cotas de error. Este mismo esquema es válido para las estructuras multirresolución basadas en refinamiento.

De Floriani *et al* ([DeFloriani97]) introducen el MSM (Multiresolution Simplicial Model), un modelo teórico formal para representar modelos multirresolución. Dado el nivel de abstracción que provee el MSM, todos los modelos multirresolución actuales se pueden expresar en términos de éste. Nosotros adoptaremos este modelo como marco conceptual para introducir de forma comprensiva los modelos reales.

5.3.1 Multiresolution Simplicial Model

De Floriani *et al* ([DeFloriani97]) definen la multiresolución como el conjunto $\mathcal{M}=\{\Gamma_0,M_0\ldots M_h,\lambda\}$, donde Γ_0 es la malla base, es decir la malla a menor resolución, $M_0\ldots M_h$ es el conjunto de modificaciones que se aplican consecutivamente a Γ_0 para refinarla y, eventualmente, obtener la malla original Γ y λ es una condición o conjunto de condiciones (reglas) que se usan para elegir qué modificaciones se aplicarán durante una reconstrucción.

Las modificaciones $M_0\ldots M_h$ aplicadas consecutivamente y en cierto orden dan lugar a la malla de máxima resolución Γ_n llamada malla de referencia. Dado que, generalmente, existen relaciones de dependencia entre modificaciones, se pueden representar mediante un grafo dirigido acíclico (DAG) el cual las modela. Dado que el grafo es dirigido, la dirección de los arcos significa precedencia. Es decir, para que una modificación M_x pueda aplicarse, es preciso que se apliquen antes todas las que la preceden, a esto se le llama transitividad cerrada. Como veremos más adelante, al aplicar métodos de refinamiento regular, el DAG se reduce a un árbol o foresta de árboles.

Un Nivel De Detalle (Level Of Detail LOD) es el resultado de aplicar un conjunto de modificaciones (en un orden determinado) a la malla base. Es posible refinar la malla base solo usando un subconjunto de las modificaciones; dicho subconjunto debe cumplir con las restricciones impuestas por λ y debe generar una malla válida, es decir, que no tenga *auto intersecciones* y que sea *conforme*.

Para asegurar que una malla sea *conforme*, todas las modificaciones aplicables deben ser *conformes*. Si existen modificaciones no *conformes* se pueden agrupar en *clusters* tal que la *frontera* combinatoria del cluster sea conforme. De esta manera, cuando hablamos de modificaciones, nos referimos a modificaciones *conformes* o a clusters de modificaciones que son *conformes*.

5.3.2 Multirresolución Basada en Refinamiento Irregular

Cignoni *et al* ([Cignoni94]) proponen refinar una malla mediante inserción iterativa de vértices. Después de insertar un vértice, se eliminan los tetraedros afectados y se re-triangula una parte de la malla en tiempo de ejecución mediante la *Tetraedrización de Delaunay*. El criterio usado para insertar un vértice es refinar el tetraedro que posee el máximo error geométrico y de campo. Esta técnica de refinamiento sólo produce mallas convexas, ya que las triangulaciones de Delaunay sólo pueden aplicarse a dominios convexos.

Por el contrario, la simplificación por *Diezmo* (Decimation), se basa en la operación inversa a la inserción iterativa, es decir, la eliminación iterativa de vértices. En este caso se parte de la malla de referencia (a máxima resolución) y

se elimina el vértice que posee menor error geométrico y de campo, re-triangulando luego de cada eliminación. Sin embargo, esta aproximación sí puede usarse en mallas definidas sobre dominios irregulares ([Hoppe96], [Garland97]).

La re-triangulación de la malla después de una operación de *Diezmado* puede aplicarse implícitamente si se utiliza (cíclicamente) el *Colapso de Arista* ([Hoppe96], [Garland97]). Cuando se aplica el *Colapso de Arista*, la arista puede colapsar en un punto arbitrario, en el punto medio o en uno de los vértices que la forman. Estas sutiles variaciones dan lugar a algoritmos con diferente grado de exactitud y calidad en la simplificación. Uno de los primeros algoritmos de multirresolución de volúmenes basado en la simplificación iterativa usando *Colapsos de Arista* fue el propuesto por Popovic y Hoppe ([Popovic97]), llamado *Progressive Simplicial Complexes* (PSC), el cual intenta extender el algoritmo *Progressive Meshes* (PM) de Hoppe ([Hoppe96]) para superficies triangulares. Sin embargo la propuesta de Popovic *et al* es demasiado general para ser realmente práctica.

Los métodos propuestos para refinamiento y simplificación son la base de modelos multirresolución de mallas tetraédricas. Cada modificación de refinamiento M_i añade detalle a la malla Γ_i. El conjunto de modificaciones $M_0 \ldots M_n$ son guardadas como un registro histórico del proceso de refinamiento. Dado que los métodos de refinamiento iterativos introducen cambios locales en la malla, las modificaciones pueden aplicarse selectivamente para refinar sólo una parte de la malla. En el modelo MSM, al aplicar la modificación M_i a la malla Γ_i se produce un cambio de estado hacia la malla Γ_{i+1}. Por ejemplo, la operación de insertar un vértice a la malla base Γ_0 da como resultado otra malla Γ_1. Sin embargo, es necesario definir el orden en que las modificaciones pueden ser aplicadas, es decir, el conjunto de reglas λ. Todas las modificaciones son insertadas en un DAG (Grafo Acíclico Dirigido), de tal modo que una modificación se relaciona sólo con las modificaciones de las cuales depende. Un ejemplo práctico de DAG es la jerarquía de vértices (foresta de árboles de vértices) propuesta por Hoppe ([Hoppe96]), donde los vértices de la malla de referencia están en las hojas y los vértices de la malla base están en las raíces. Las jerarquías de vértices son adecuadas para codificar la dependencia de *Colapsos de Arista* o *Vertex Splits*.

5.3.3 Multirresolución Basada en Refinamiento Regular

Algunos volúmenes presentan una estructura interna regular, es decir que, partiendo de una malla base de forma regular, mediante reglas de refinamiento basadas en patrones (generalmente recursivos), y mediante subdivisión, se aproxima la forma real del volumen. Las mallas más generales, cuyas topologías son irregulares o no estructuradas pueden ser trasladadas a un dominio regular, donde pueden ser refinadas regularmente, para luego ser devueltas a su dominio original. Esto requiere parametrizar el volumen mediante un proceso conocido como *Remeshing*. Lamentablemente todavía no existe un método de *Remeshing* general.

Algunos métodos de refinamiento regular son extensiones de casos bidimensionales. El uso de Octrees para subdividir el dominio tridimensional, por ejemplo, requiere la voxelizacion previa del volumen, sin contar con los problemas de discontinuidad entre nodos de distinta resolución, es una adaptación de métodos de refinamiento para terrenos en dos dimensiones.

Otra aproximación consiste en descomponer el volumen en símplices (tetraedros) y, a partir de una malla base, aplicar técnicas de refinamiento regular. Generalmente la subdivisión comienza por subdividir la caja que encierra al volumen (Axis Aligned Bounding Box) en tetraedros. Luego se añaden vértices o bien se subdividen los tetraedros recursivamente usando algún patrón o formula matemática bien definida. La ventaja de este método es su simplicidad. Sólo es necesario usar un pequeño conjunto de reglas para producir una malla de máxima resolución, cuyas resoluciones de transición son a su vez regulares. Esto habilita una representación compacta de los datos, ya que lo esencial de la multirresolución recae más en las reglas de generación y menos en los datos.

Rivara ([Rivara84]) propuso el refinamiento de redes triangulares mediante la bisección de triángulos tomando como criterio subdividir por la arista más larga. Posteriormente Plaza y Carey extendieron el método de Rivara a *n* dimensiones. Para el caso tridimensional, Plaza y Carey formulan un algoritmo de tres etapas. En la primera etapa, se selecciona el subconjunto de tetraedros a ser refinado de acuerdo a cierto criterio y a cada uno se le añade un nodo en el punto medio de sus aristas (Figura 19b). En la segunda etapa se comprueba la *Conformidad* de la malla. Una malla es conforme, si y sólo si, la intersección de dos tetraedros es una cara en común, una arista en común, un vértice en común o no existe intersección. En esta etapa, a cada tetraedro no Conforme, se le agrega un nodo en cada arista no Conforme así como en la arista más larga (Figura 19c). En la tercera etapa, cada tetraedro que posee nodos en sus aristas es subdividido según un patrón (Figura 19d).

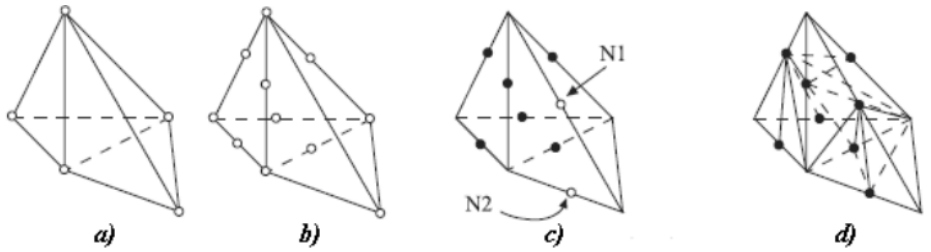

Figura 19: Aplicación del algoritmo de subdivisión de Plaza y Carey: a) Malla inicial, b) Pasos 1 y 2, c) Paso 3: chequeo de conformidad, d) Paso 5: división en tetraedros. Extraído de Borgo *et al* ([Borgo04]).

La adición de nodos extra para mantener la malla Conforme (nodos N1 y N2 en la Figura 19c) es una solución empírica para prevenir que se formen tetraedros de formas irregulares. La subdivisión de los tetraedros seleccionados en la primera etapa del algoritmo, provoca la subdivisión de otros tetraedros con el fin de mantener la conformidad de la malla. Esto introduce cierto error en el dominio.

El Multiresolution Tetrahedral Framework (MTF) de Zhou *et al* ([Zhou97]) usa la subdivisión recursiva para refinar volúmenes de topología regular. Zhou *et al* definen tres configuraciones básicas de tetraedros, llamadas Clase 1, Clase 2 y Clase 3. La definición y los detalles de las clases del MTF las encontramos en [Zhou97]. El MTF comienza por descomponer el dominio hexaédrico en 12 tetraedros de Clase 1. La siguiente subdivisión se basa en bisección de tetraedros por la arista más larga y da lugar a tetraedros de Clase 2. La subdivisión regular de tetraedros de Clase 2 produce tetraedros de Clase 3. Con la siguiente subdivisión de los tetraedros de Clase 3 se vuelven a generar tetraedros de Clase 1 pero en una escala más pequeña, con lo cual volvemos a la configuración inicial. La aplicación recursiva de las reglas de subdivisión refina la malla en tres pasos; cada paso produce tetraedros en una configuración especial. En el MTF el refinamiento se da rápidamente y el modelo multirresolución necesita almacenar la malla de mayor resolución explícitamente. Esto conlleva un alto costo de almacenamiento.

El Slow Growing Subdivision (SGS) introducido por Pascucci ([Pascucci02]) generaliza el criterio de subdivisión base del MTF a *n* dimensiones; además su método está diseñado para manejar mayores cantidades de datos. Similarmente al método MTF, SGS requiere descomponer el dominio hexaédrico en tetraedros. SGS define tres tipos diferentes de entidades llamadas *diamantes*. La malla se refina mediante la unión y división de diamantes. En la primera subdivisión se divide el hexaedro en 6 pirámides, esas pirámides no sólo son subdivididas en tetraedros, sino que además se requiere formar diamantes uniendo las pirámides que colindan con otra celda hexaédrica. Estos diamantes se denominan *Diamantes Octaédricos*, cuyo centro es el centro de la cara que une ambas celdas hexaédricas. El siguiente paso consiste en subdividir los *Diamantes Octaédricos* en ocho tetraedros. Uniendo de cierta forma los tetraedros resultantes se forma otro diamante llamado *Diamante Hexaédrico*, el cual se subdivide en tetraedros. Finalmente la recombinación de los tetraedros originados a partir del *Diamante Hexaédrico* da lugar al *Diamante Cúbico* el cual posee aristas de la mitad de tamaño del cubo original. El ciclo de refinamiento se repite tomando como base al *Diamante Cúbico* y así sucesivamente hasta alcanzar el máximo nivel de detalle. De manera similar a MTF, la formación de cada diamante en SGS corresponde a una etapa de refinamiento.

Refinamiento Rojo/Verde

El Refinamiento Rojo / Verde fue propuesto por Bank *et al* ([Bank83]), para refinar superficies triangulares. El método de Bank *et al* consiste en una combinación de refinamiento regular (rojo) e irregular (verde). El refinamiento regular consiste en subdividir un triángulo en cuatro subtriángulos congruentes mediante la unión de los puntos medios de sus aristas (Figura 20a). El refinamiento irregular solamente se usa para mantener la *Conformidad* de la triangulación y consiste en la bisección y en la trisección de triángulos como se muestra en la Figura 20b y Figura 20c.

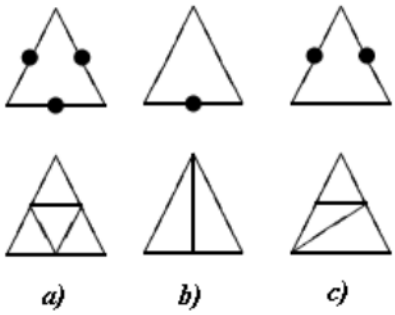

Figura 20: a) subdivisión regular (rojo) de un triangulo en cuatro. b) y c) subdivisión irregular de triángulos (verde), de acuerdo al número de aristas marcadas que poseen.

Como ejemplo, podemos analizar el refinamiento incremental de un triángulo como el de la Figura 21a. El refinamiento regular del triángulo genera cuatro subtriángulos regulares como se muestra en la Figura 21b. En el subsiguiente paso seleccionamos el triángulo *i* para ser refinado. El refinamiento regular del triángulo *i* inserta cuatro nuevos triángulos regulares a la malla y causa que la misma quede no *Conforme*. Es necesario aplicar un patrón de refinamiento irregular (bisección) al triángulo *iv* para lograr la *Conformidad*. Esta operación (llamada C*lausura Verde*) inserta dos triángulos irregulares en la malla. El resultado se muestra en la Figura 21c. En el siguiente paso seleccionamos el triángulo *ii* para ser refinado. Nuevamente el triángulo *ii* introduce cuatro nuevos triángulos regulares en el dominio, quedando la malla no *Conforme*. En este caso, la subdivisión irregular aplicada al triángulo *iv* debe desaparecer (ambos triángulos deben ser eliminados) y se aplica otro patrón de subdivisión irregular (trisección) al triángulo *iv* para lograr la *Conformidad*. Esta operación inserta tres triángulos irregulares en la malla. El resultado se muestra en la Figura 21d.

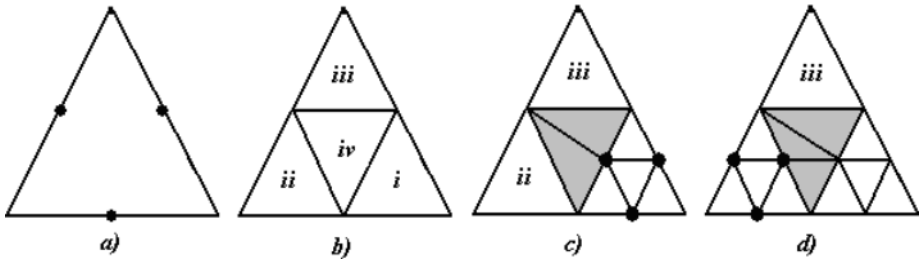

Figura 21: Primeros pasos del refinamiento rojo/verde de un triangulo según Bank *et al.* a) Triangulo con sus aristas marcadas para subdivisión, b) División en cuatro triángulos aplicando un patrón regular, c) La subdivisión regular del triangulo *i* causa la subdivisión irregular del triangulo *iv.* d) la subdivisión regular del triangulo *ii* causa nuevamente la subdivisión irregular del triangulo *iv* con otro patrón irregular.

Del ejemplo anterior podemos notar que existe anidamiento entre elementos de distintos niveles, es decir, cada triángulo producido en cierto nivel de subdivisión proviene de la subdivisión de otro elemento de un nivel superior. Es posible definir explícitamente la relación Padre – Hijo de cada elemento. Por ejemplo la estructura de datos que modela las relaciones de anidamiento del ejemplo anterior es un árbol cuaternario.

El refinamiento de Bank *et al*, genera triangulaciones *Estables*. Existen diversas métricas para medir la estabilidad de una triangulación, la más usual se basa en

medir los ángulos de cada elemento. La estabilidad asegura que la forma de los triángulos sea regular, es decir, que los ángulos sean mayores a cero, no sean demasiado agudos ni obtusos. Por ejemplo, la forma regular de los triángulos asegura la estabilidad numérica del sistema cuando la malla es usada para la resolución de ecuaciones por el método de elemento finito.

Para generar triangulaciones estables, Bank *et al* establecen que los triángulos irregulares (que provienen de refinamiento irregular como por ejemplo los triángulos sombreados en la Figura 21c y Figura 21d) no deben ser refinados. Es decir que, si deseamos refinar algún triángulo irregular, es necesario primero eliminar el refinamiento irregular existente para luego refinar regularmente al padre.

Bey ([Bey95]) extiende el refinamiento de Bank *et al* ([Bank83]) a tres dimensiones, es decir a mallas de tetraedros (volúmenes). El método de Bey mantiene las mismas propiedades que el algoritmo de Bank *et al* (*Conformidad, Anidamiento y Estabilidad*). El refinamiento regular consiste en subdividir un tetraedro T en ocho subtetraedros $T_1 \ldots T_8$ de tal forma que los vértices de T_i coincidan con un vértice de T o con el punto medio de alguna arista de T.

Bey conecta los puntos medios de cada arista de cada triángulo de T de la misma forma que se hace en el refinamiento regular de triángulos planteado por Bank *et al*. Esto permite cortar cuatro tetraedros de las esquinas, los cuales son congruentes con T. En el interior queda un octaedro (Figura 22). El octaedro se puede dividir en cuatro tetraedros (Figura 23), los cuales generalmente no son congruentes con T, dependiendo de la diagonal que se use para subdividirlo. Bey introdujo un algoritmo que, para cualquier elemento inicial, genera subtetraedros de a lo sumo 3 clases de congruencia, no importando cuántos refinamientos sucesivos se realicen, lo que garantiza la estabilidad de las triangulaciones generadas.

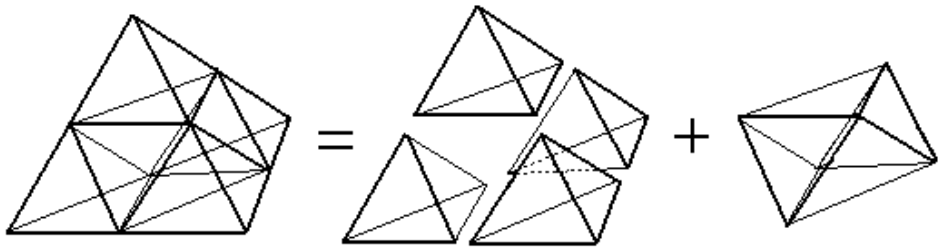

Figura 22: Primera parte de la subdivisión regular de Bey, un tetraedro T es subdividido en cuatro subtetraedros congruentes con T y un octaedro.

Bey plantea la congruencia en términos del conjunto de transformaciones (traslados y escalas) que hacen que un tetraedro T_1 coincida con un tetraedro T_2. Si existen tales transformaciones, decimos que T_1 es congruente con T_2.

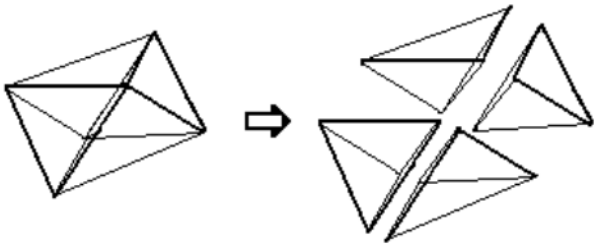

Figura 23: Segunda parte de la subdivisión regular de Bey, el octaedro es dividido en cuatro tetraedros.

Al igual que el algoritmo de Bank *et al*, para el caso de Refinamiento Selectivo de la malla, el método de Bey requiere introducir refinamiento irregular para mantener la *Conformidad* de la malla. Dado que cada tetraedro posee seis aristas, existen $2^6 = 64$ posibles patrones de refinamiento irregular. El caso en que las seis aristas están marcadas corresponde al refinamiento regular; así mismo, el caso en que ninguna arista está marcada no corresponde a ningún refinamiento. Usando argumentos de simetría, los otros 62 casos pueden ser divididos en 9 tipos diferentes. Por razones prácticas varios autores restringen el refinamiento irregular a sólo a 4 tipos (Figura 24), forzando a los otros 5 tipos a realizar refinamiento regular.

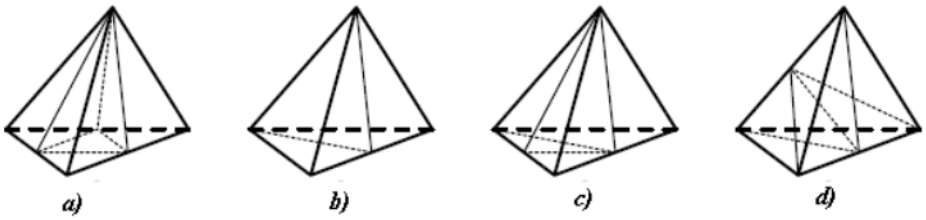

Figura 24: Tipos de subdivisión irregular: a) tres aristas marcadas

En la Figura 24a, mostramos el refinamiento irregular cuando el tetraedro posee marcas en tres aristas que pertenecen a una misma cara. La Figura 24b corresponde al caso en que el tetraedro posee una arista marcada. La Figura 24c corresponde al caso en que el tetraedro posee dos aristas marcadas que pertenecen a la misma cara. Finalmente, la Figura 24d corresponde al caso en que el tetraedro posee dos aristas marcadas que no corresponden a la misma cara.

El caso mostrado en la Figura 24c necesita un tratamiento especial para preservar la *conformidad* de la malla, ya que la triangulación de la cara cuyas aristas están marcadas puede hacerse de dos formas posibles. Notemos también que si un tetraedro posee tres aristas marcadas que no pertenecen a la misma cara o bien tiene, en general, más de tres aristas marcadas, se deben marcar todas las aristas faltantes del tetraedro para poder aplicar refinamiento regular. Esto puede causar un efecto dominó ya que la introducción de nuevas marcas potencialmente induce el refinamiento de otros tetraedros o bien induce el cambio de tipo de refinamiento irregular que se debe aplicar para lograr la *Conformidad* de la malla.

Bey ([Bey95]) introduce un algoritmo de refinamiento global, el cual es guiado por la exactitud de una solución numérica. De acuerdo a cierto criterio se escogen los tetraedros que se deben refinar regularmente. Estos tetraedros son marcados como regulares y sus seis aristas son marcadas para refinamiento. Luego se

aplican las reglas de refinamiento irregular para refinar los tetraedros adyacentes que mantienen la Conformidad (*Clausura Verde*). Después de este refinamiento es posible que la malla aún no sea conforme, debido al efecto dominó mencionado anteriormente. Es así que Bey plantea evaluar los tetraedros nuevamente, y de ser necesario, eliminar el refinamiento anterior (*Coarsening*) y volver a refinar con un nuevo patrón de refinamiento según corresponda. Bey no plantea un algoritmo de *Coarsening* global, sino tan solo local.

5.3.4 Modelado Multirresolución para Símplices con Wavelets

Castro ([Castro97], [Castro05]), Boscardín ([Boscardin01]) y Castro *et al* ([Castro06]), introducen la construcción de wavelets sobre volúmenes de topología arbitraria, siempre y cuando sea posible la descomposición del domino en símplices y además la red cumpla con la propiedad de conectividad de subdivisión.

La propiedad de conectividad de subdivisión se da en mallas de triángulos y tetraedros que provienen del refinamiento iterativo de una malla base mediante subdivisión regular. En el contexto de la multirresolución podemos definir una malla con esta propiedad como $\mathcal{M}=\{\Gamma_0,M_0\ldots M_h,\lambda\}$, donde M_i es el refinamiento de M_{i-1} y por lo tanto la malla Γ_i corresponde a un nivel de refinamiento de la malla Γ_{i-1}. Además, los vértices de la malla Γ_{i-1} deben estar incluidos en la malla Γ_i.

La compresión y transmisión progresiva de volúmenes multirresolución de topología arbitraria en el dominio Wavelet fue introducida por Castro ([Castro05]) y Castro *et al* ([Castro06]). En su trabajo desarrollan un nuevo método de modelado de volúmenes para dominios no estructurados. Éste se caracteriza por una alta ganancia en el nivel de compresión que puede alcanzarse, un control preciso del error en la norma L^2, el refinamiento progresivo de la secuencia comprimida y una descompresión rápida y progresiva. También cabe señalar que, debido a que la red inicial captura el *Genus* del volumen final, se garantiza que el mismo evolucionará manteniendo la topología del volumen deseado.

Para extender el modelado de volúmenes con Wavelets a dominios tetraédricos no estructurados, es necesario definir la construcción de Wavelets sobre tetraedros y usar un esquema multirresolución basado en subdivisión. El análisis en base de wavelets tiene la ventaja de generar directamente una estructura de datos multiresolución con cotas de error garantizadas. El ingrediente básico necesario para el análisis de wavelets es la construcción de espacios anidados de funciones por lo que se requiere que el volumen tenga la propiedad de conectividad de subdivisión.

El esquema de subdivisión adoptado por Castro y Boscardín es el de Bey ([Bey95]), el cual subdivide un tetraedro en ocho subtetraedros de igual volumen.

Mediante un proceso de *análisis* descomponen la función, definida sobre el dominio del volumen, en dos conjuntos según sus frecuencias: Coeficientes y Detalles. El conjunto de Coeficientes está conformado por las frecuencias importantes de la función, es decir, las bajas frecuencias, mientras que las frecuencias altas conforman el conjunto de Detalles. Geométricamente, el proceso de análisis wavelet de una malla Γ_i produce una red de menor resolución Γ_{i-1} y un conjunto de detalles. La malla de menor resolución Γ_{i-1} es adecuada para representar el conjunto de Coeficientes, mientras que el conjunto de detalles se guardan por separado (Figura 25). La red original Γ_i puede reconstruirse mediante un proceso de *síntesis*, teniendo la red de menor resolución Γ_{i-1} podemos reconstruir la red aumentando su resolución (mediante el refinamiento regular de Bey) y añadiendo los detalles que fueron sustraídos en la etapa de análisis.

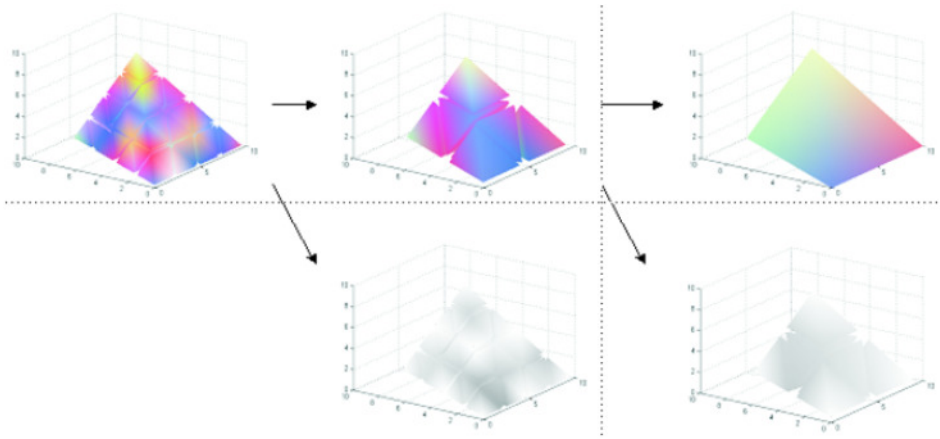

Figura 25: Modelado multirresolución con wavelets. En la fase de análisis, la malla de resolución *i* es engrosada y filtrada dando como resultado una malla de resolución *i*-1. Los detalles que resultan del filtrado (mostrados en gris sobre una malla virtual de resolución *i*) se guardan aparte. (Cortesía de Silvia Castro).

En la Figura 25 se muestra el procedimiento de análisis aplicado recursivamente hasta alcanzar la menor resolución posible, en este caso la malla base es un tetraedro. La función definida sobre el volumen está representada por colores, mientras que los detalles resultantes del análisis están representados en escala de grises.

Construcción de Wavelets sobre el tetraedro: Esquema basado en vértices

Para formular el análisis multirresolución para un volumen de topología arbitraria, es necesario definir los cuatro filtros A_j, B_j, P_j y Q_j; donde A_j, B_j son los filtros usados en el análisis y P_j, Q_j son sus inversas usadas en la reconstrucción (síntesis). Debido a que no existen expresiones analíticas para las funciones de escala, ni sus inversas, estas funciones son definidas como el límite de un algoritmo llamado Algoritmo Cascada. El Algoritmo Cascada consiste de dos fases: Particionamiento y Perturbación. El particionamiento se obtiene

mediante el refinamiento del volumen y las perturbaciones se dan al aplicar las funciones de escala (interpolantes) de wavelet a los escalares del domino.

Está probado que si $f(T_j)$ representa la función definida sobre el tetraedro T_j, y además $f(T_j)$ es lineal y continua, cualquier función f definida sobre T_{j+1} (el refinamiento regular de T_j) también será lineal y continua dentro de T_{j+1}.

En la fase de particionamiento se refina un tetraedro T_j cuyos índices de vértices pertenecen al conjunto $K_j=\{1,2,3,4\}$, agregándole seis nuevos vértices cuyos índices pertenecen al conjunto $M_j=\{5,6,7,8,9,10\}$. De tal forma que, en la siguiente resolución, el tetraedro T_{j+1} posee el conjunto de vértices total, es decir, la unión de K_j y M_j, formalmente: $k_{j+1} = k_j \cup M_j$

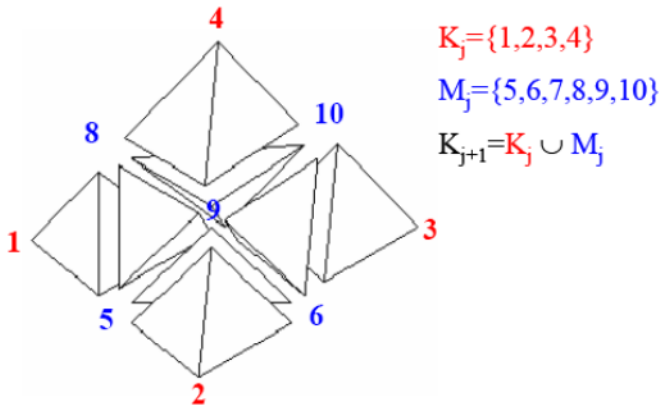

Figura 26: Indexado de un tetraedro. (Cortesía de Silvia Castro).

Mediante un proceso de Lifting, se definen las siguientes expresiones para el cálculo de los coeficientes de la transformación rápida de wavelet con base de HAAR para las fases de análisis y síntesis:

Sean (u,v) los índices de las seis aristas del tetraedro T y sea m el punto medio de la arista (u,v):

Análisis:

Calcular los Detalles:

$$\forall m \in M_j \qquad \left\{ d_{j,m} := c_{j+1,m} - \frac{1}{2}\left(c_{j+1,u} + c_{j+1,v}\right) \right\}$$

Calcular los Coeficientes:

$$\forall k \in Kj \qquad c_{j,k} = c_{j+1,k}$$

$$\forall m \in M_j, \qquad u,v \in K_j \begin{cases} c_{j,u} = c_{j,u} + s_{j,u,m} d_{j,m} \\ c_{j,v} = c_{j,v} + s_{j,v,m} d_{j,m} \end{cases}$$

Síntesis:

Calcular los Coeficientes:

$$\forall k \in Kj \qquad c_{j+1,k} = c_{j,k}$$

$$\forall m \in M_j, \qquad u,v \in K_j \begin{cases} c_{j,u} = c_{j,u} - s_{j,u,m} d_{j,m} \\ c_{j,v} = c_{j,v} - s_{j,v,m} d_{j,m} \end{cases}$$

Reconstruir los coeficientes que faltan, a partir de los que se tiene y los detalles:

$$\forall m \in M_j \qquad \left\{ c_{j+1,m} := d_{j,m} + \frac{1}{2}\left(c_{j+1,u} + c_{j+1,v}\right) \right\}$$

Las expresiones para el cálculo de los pesos $S_{j,*,m}$ se encuentran en [Castro05]. Sin embargo no son de relevancia práctica en la presente tesis.

En la fase de análisis, los coeficientes del tetraedro T_j son calculados por submuestreo del tetraedro de mayor resolución T_{j+1}. Los detalles que pierde el tetraedro T_{j+1} son calculados como la diferencia de cada escalar en M_j (punto medio de la arista (u, v)) menos la interpolación de los escalares de la arista (u, v). En la fase de síntesis, los escalares K_j del tetraedro T_{j+1} son calculados por *supersampling*, mientras que los escalares M_j (puntos medios de las aristas (u, v)) son reconstruidos añadiéndole los detalles perdidos a cada interpolación de los escalares de las aristas (u, v).

Construcción de Wavelets sobre el tetraedro: Esquema basado en tetraedros

Este modelo es adecuado para representar funciones definidas sobre tetraedros. Análogamente a las funciones definidas sobre voxels, en este tipo de volúmenes, cada tetraedro posee un valor uniforme, por lo tanto, sólo es necesario almacenar un escalar por cada celda.

En el proceso de síntesis se introducen nuevos tetraedros $T_{\beta i}$ obtenidos por subdivisión del tetraedro padre T_α de acuerdo al esquema de subdivisión de Bey ([Bey95]). Esto significa que, al pasar de un nivel de resolución j a un nivel j + 1, se generarán 8 tetraedros $T_{\beta i}$ que reemplazarán al tetraedro de menor resolución T_α (Figura 27).

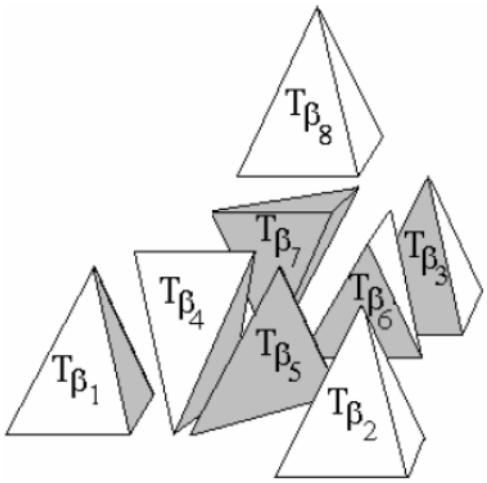

Figura 27: El tetraedro padre T_α es reemplazado por 8 tetraedros $T_{\beta i}$. (Cortesía de Silvia Castro)

Sea $\boldsymbol{\varphi}$ un conjunto de ecuaciones lineales definidas sobre un tetraedro, y sea $B(\alpha)$ el conjunto de tetraedros $T_{\beta i}$ que resultan de refinar el tetraedro $T_{\alpha.}$ Podemos expresar el refinamiento como:

$$\varphi_\alpha = \sum_{\beta \in B(\alpha)} \frac{1}{\sqrt{8}} \varphi_\beta \tag{40}$$

La Ecuación (40) denota que la función definida sobre tetraedro T_α se expresa en el análisis como una combinación lineal de las funciones definidas en los tetraedros de mayor resolución $T_{\beta i}$.

La ecuación de escala de wavelet es:

$$\psi_\gamma = \sum_\beta g_{\gamma,\beta} \varphi_\beta \tag{41}$$

Donde $g_{\gamma,\beta}$ es un conjunto de pesos y $\gamma \in G(\beta)$, donde $G(\beta)$ es el conjunto de detalles obtenidos al pasar de una resolución j a una resolución j-1. Por lo tanto la Ecuación (41) es un filtro que extrae los detalles del conjunto de tetraedros $T_{\beta i}$.

La derivación completa de las ecuaciones propuestas para el análisis y síntesis se puede encontrar en [Castro05] y [Castro06]. Las ecuaciones son:

Análisis:

Calcular el Coeficiente:

$$c_{j,\alpha} = \sum_{\forall \beta \in C_j(\alpha)} \frac{1}{\sqrt{8}} c_{j+1,\beta}$$

Calcular los Detalles:

$$d_{j,\gamma} = \sum_{\beta \in C_j(\alpha)} g_{\gamma,\beta} c_{j+1,\beta}$$

Para los cuales:

$$d_{j,\gamma_1} = \frac{1}{\sqrt{8}}\left(c_{j+1,\beta_1} + c_{j+1,\beta_2} + c_{j+1,\beta_3} + c_{j+1,\beta_4}\right) - \frac{1}{\sqrt{8}}\left(c_{j+1,\beta_5} + c_{j+1,\beta_6} + c_{j+1,\beta_7} + c_{j+1,\beta_8}\right)$$

$$d_{j,\gamma_2} = \frac{1}{2}\left(c_{j+1,\beta_1} + c_{j+1,\beta_2}\right) - \frac{1}{2}\left(c_{j+1,\beta_3} + c_{j+1,\beta_4}\right)$$

$$d_{j,\gamma_3} = \frac{1}{2}\left(c_{j+1,\beta_5} + c_{j+1,\beta_6}\right) - \frac{1}{2}\left(c_{j+1,\beta_7} + c_{j+1,\beta_8}\right)$$

$$d_{j,\gamma_4} = \frac{1}{\sqrt{2}}\left(c_{j+1,\beta_1} - c_{j+1,\beta_2}\right)$$

$$d_{j,\gamma_5} = \frac{1}{\sqrt{2}}\left(c_{j+1,\beta_3} - c_{j+1,\beta_4}\right)$$

$$d_{j,\gamma_6} = \frac{1}{\sqrt{2}}\left(c_{j+1,\beta_5} - c_{j+1,\beta_6}\right)$$

$$d_{j,\gamma_7} = \frac{1}{\sqrt{2}}\left(c_{j+1,\beta_7} - c_{j+1,\beta_8}\right)$$

Síntesis:

Calcular los Coeficientes:

$$c_{j+1,\beta} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \sum_{\gamma \in G_j(\beta)} g_{\gamma,\beta} d_{j,\gamma}$$

Para los cuales:

$$c_{j+1,\beta_1} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(\frac{1}{\sqrt{8}} d_{j,\gamma_1} + \frac{1}{2} d_{j,\gamma_2} + \frac{1}{\sqrt{2}} d_{j,\gamma_4}\right)$$

$$c_{j+1,\beta_2} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(\frac{1}{\sqrt{8}} d_{j,\gamma_1} + \frac{1}{2} d_{j,\gamma_2} - \frac{1}{\sqrt{2}} d_{j,\gamma_4}\right)$$

$$c_{j+1,\beta_3} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(\frac{1}{\sqrt{8}} d_{j,\gamma_1} - \frac{1}{2} d_{j,\gamma_2} + \frac{1}{\sqrt{2}} d_{j,\gamma_5}\right)$$

$$c_{j+1,\beta_4} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(\frac{1}{\sqrt{8}} d_{j,\gamma_1} - \frac{1}{2} d_{j,\gamma_2} - \frac{1}{\sqrt{2}} d_{j,\gamma_5}\right)$$

$$c_{j+1,\beta_5} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(-\frac{1}{\sqrt{8}} d_{j,\gamma_1} + \frac{1}{2} d_{j,\gamma_3} + \frac{1}{\sqrt{2}} d_{j,\gamma_6} \right)$$

$$c_{j+1,\beta_6} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(-\frac{1}{\sqrt{8}} d_{j,\gamma_1} + \frac{1}{2} d_{j,\gamma_3} - \frac{1}{\sqrt{2}} d_{j,\gamma_6} \right)$$

$$c_{j+1,\beta_7} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(-\frac{1}{\sqrt{8}} d_{j,\gamma_1} - \frac{1}{2} d_{j,\gamma_3} + \frac{1}{\sqrt{2}} d_{j,\gamma_7} \right)$$

$$c_{j+1,\beta_8} = \frac{1}{\sqrt{8}} c_{j,\alpha} + \left(-\frac{1}{\sqrt{8}} d_{j,\gamma_1} - \frac{1}{2} d_{j,\gamma_3} - \frac{1}{\sqrt{2}} d_{j,\gamma_7} \right)$$

En la Figura 28 vemos dos pasos en el análisis para funciones escalares definidas sobre los vértices del tetraedro (arriba) y dos pasos en el análisis que corresponden a funciones escalares definidas sobre celdas tetraédricas (abajo); en ambos casos el atributo escalar se ha asociado con un color.

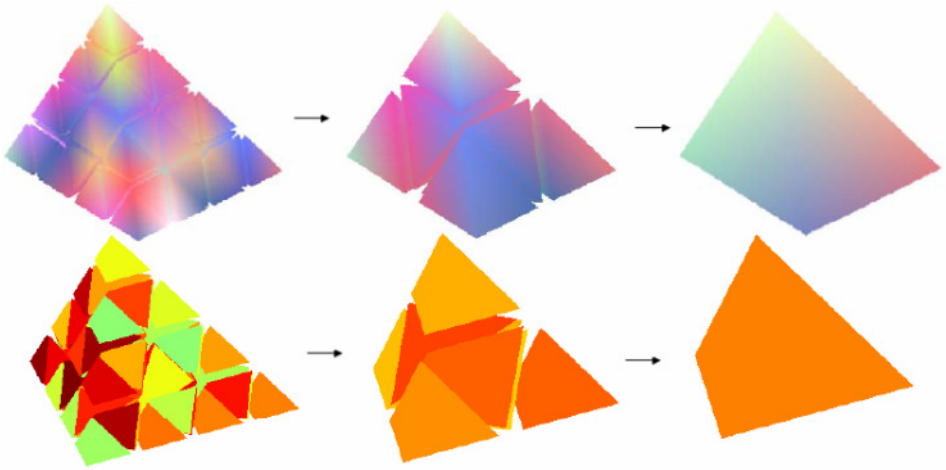

Figura 28: Vértices vs. Celdas. Arriba, dos pasos del análisis wavelet a una malla con el atributo definido en los vértices. Abajo, dos pasos del análisis wavelet a la misma malla pero con el atributo definido sobre los tetraedros. (Cortesía de Silvia Castro)

Castro ([Castro05]) y Castro *et al* ([Castro06]) proponen usar Octrees para representar, tanto la jerarquía de detalles como la de tetraedros. La malla base se representa como una malla indexada de tetraedros. Cada tetraedro de la malla base es la raíz de una jerarquía de tetraedros codificada en un Octree. Las relaciones de dependencia que existen entre los tetraedros de las diferentes resoluciones vienen dadas por el esquema de subdivisión Rojo / Verde de Bey ([Bey95]). La representación compacta de una malla modelada con wavelets consiste en la malla base y una jerarquía de detalles codificada como una foresta de Octrees. Esta representación además de ser compacta, permite un alto nivel de compresión en los detalles.

5.3.5 Refinamiento Selectivo

El proceso de extraer una resolución explícita del modelo multirresolución se puede expresar en términos de una función booleana $C(\sigma)$ que evalúa el error de cada tetraedro σ con respecto a cierta tolerancia. Haciendo un recorrido en amplitud del DAG, se aplican todas las modificaciones que cumplen con la función $C(\sigma)$, respetando las dependencias funcionales implícitas en el DAG. Por supuesto, el recorrido comienza por la malla base Γ_0 que es la raíz del DAG. La función $C(\sigma)$ puede medir el error geométrico, error de campo o el error en el espacio de pantalla. La función $C(\sigma)$ toma el nombre de *View-Dependant LOD*, cuando mide el error en el espacio de pantalla.

Llamamos *Refinamiento Selectivo* a la tarea de extraer una resolución explícita del modelo, usando un criterio de error dependiente del punto de vista. Es usual utilizar una región de interés o ROI como criterio de refinamiento. Por ejemplo, el *Frustum de Visión* puede ser utilizado como ROI, definiendo una función $C(\sigma)$ que permita refinar sólo los tetraedros que están dentro del *Frustum*. Los nodos del DAG que cumplen con el criterio $C(\sigma)$ son seleccionados para ser parte de la malla requerida, así como todos sus ancestros transitivos en el grafo.

5.4 Discusión de los Modelos Revisados

Las técnicas multirresolución irregulares, a pesar de tener una estructura compleja, permiten mejores y más precisas aproximaciones de volúmenes en cualquier topología. Las técnicas regulares, por otro lado, permiten el refinamiento eficiente de volúmenes regulares y son más adecuadas para trabajar en memoria secundaria (Out-Of-Core). Si bien las técnicas basadas en wavelets sobre tetraedros pueden utilizarse para modelar volúmenes no estructurados, es necesario que éstos posean la propiedad de conectividad de subdivisión. Ciertos volúmenes pueden parametrizarse a través de un remeshing y así tener conectividad de subdivisión; sin embargo, aún no existe un método de remeshing general para este propósito.

Las técnicas basadas en wavelets sobre tetraedros introducen un modelo de transmisión progresiva de redes tetraédricas, en el cual primero debe transmitirse la red base, que el receptor puede mostrar inmediatamente, y luego se deben transmitir los detalles. Éstos pueden transmitirse en secuencia arbitraria. Si cada detalle o conjunto de detalles especifican a qué tetraedro pertenecen en la multirresolución, la transmisión ordenada de los mismos no es importante. De hecho, la pérdida de algunos detalles durante la transmisión no impide la visualización de la malla y sólo se introducen algunos errores locales.

El uso de patrones regulares de subdivisión permite un almacenamiento eficiente en disco y una mejor organización de los datos en el sentido que mejoran el uso

de memoria y las políticas de acceso a memoria secundaria, lo cual es necesario para la visualización eficiente de grandes cantidades de datos. Las visualizaciones producto de ambas técnicas (regulares e irregulares) son comparables, aún cuando las técnicas irregulares se adaptan mejor a cualquier dominio. Ambas aproximaciones son buenas para modelar problemas específicos y todavía no es posible formular un único juicio de calidad que prefiera una sobre la otra. ♦

CAPÍTULO VI

VISUALIZACIÓN INTERACTIVA DE VOLÚMENES EN EL CONTEXTO DE LA VISUALIZACIÓN REMOTA

Actualmente, uno de los desafíos para la comunidad de visualización es proveer sistemas de visualización que estén naturalmente integrados a la computación en la GRID. Uno de los objetivos de estos sistemas es que el usuario pueda visualizar enormes cantidades de datos desde un computador común con acceso a Internet. Por ejemplo, el *Rendering Directo de Volúmenes* representa un caso típico de visualización que requiere gran cantidad de recursos computacionales. Se han desarrollado clusters gráficos especializados para visualizar grandes conjuntos de datos en paralelo y en grandes displays de alta resolución. Sin embargo, la disponibilidad de esos clusters a menudo es limitada. Si bien se han desarrollado muchas técnicas de Rendering de Volúmenes que trabajan en computadores comunes con hardware de video en el estado del arte ([Kniss02], [Hadwiger01], [Max00], [Moreland04], [Stein94], [Krueger03], [Weiler03], [Bernardon04], [Stegmaier05], [Wylie02], [Weiler04], [Rezk-Salama00], [Engel01], [Engel02], [Roettger03], [Leven02], [Weiler00], [Neophytou05], [Callahan05]), estas técnicas proveen una solución que permite a los investigadores visualizar datos localmente cuando no hay disponibilidad de recursos especializados. Los limitados recursos de memoria y almacenamiento de los computadores personales no permiten la visualización de grandes conjuntos de datos. La disponibilidad de usar recursos alejados geográficamente mediante la GRID permite a sistemas heterogéneos compartir datos y poder de procesamiento; la arquitectura cliente/servidor de los Servicios Web permite al usuario promedio ser cliente de estos sistemas. Dentro del esquema cliente/servidor, la visualización llevada a cabo completamente del lado del servidor resta interactividad al proceso de visualización, mientras que, por otro lado, la visualización sólo del lado del cliente no siempre es posible (Figura 29). Es pues necesario encontrar el balance adecuado entre la visualización del lado del servidor y del cliente.

En este contexto, a través de la Visualización Progresiva de volúmenes, introducimos el concepto de visualización *ad-hoc*. La visualización llevada a cabo del lado del cliente tiene como ventaja mantener la interactividad del proceso de visualización. Es deseable explotar la visualización del lado del cliente al máximo sin sobrepasar los recursos y capacidad de procesamiento locales y al mismo tiempo lograr la visualización más fina y precisa posible. La visualización de datos en múltiples resoluciones tiene la ventaja de adaptar la resolución (i.e.

cantidad) de los datos a diversos criterios (desde cotas de error en el dominio hasta la resolución del monitor). El usuario puede obtener visualizaciones *ad-hoc* de manera progresiva, usando sus propios recursos, y así explorar interactivamente el conjunto de datos hasta que en cierto momento desea visualizar una parte (o todo el conjunto) a máximo detalle, momento en que, de ser necesario, se corta la transmisión de datos y el proceso de rendering pasa a realizarse del lado del servidor; luego éste transmite el resultado y el usuario recibe una imagen estática del rendering de volúmenes (Figura 30). La combinación de Visualización Progresiva, Multirresolución y Visualización del lado del Servidor, permiten un balance entre el uso exclusivo de los recursos remotos y el sólo uso de la capacidad de procesamiento local.

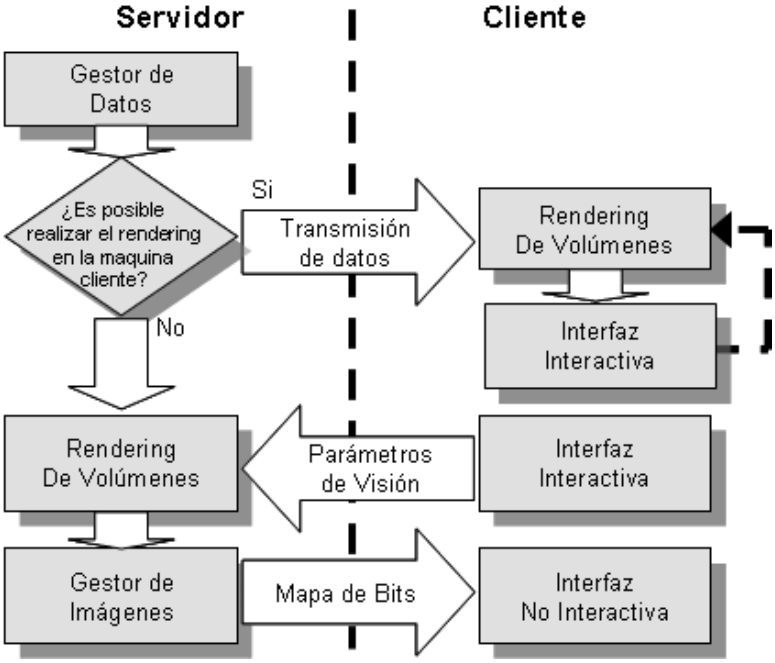

Figura 29: Diagrama de flujo de datos del típico proceso de visualización Cliente / Servidor.

Para hacer viable un sistema de visualización basado en la GRID, es imperativo hacer uso eficiente de la red; muchas estrategias consisten en transmitir sólo las porciones de datos que serán procesadas y visualizadas; sin embargo, la transmisión completa de dichas porciones es necesaria antes de realizar la visualización. La transmisión de LODs (niveles de detalle) permite visualizar los datos en una resolución específica y agiliza el proceso de exploración. La transmisión de un LOD debe completarse antes de poder visualizarlo y además la transmisión de varios LODs implica una tasa cada vez mayor de datos redundantes. En ese sentido la Transmisión Progresiva, junto con la Visualización Progresiva de datos, posibilitan hacer un uso óptimo de la red en el sentido que los datos son transmitidos sólo una vez al cliente y además son visualizados de inmediato.

Esta investigación se enmarca en el contexto general de la visualización remota Cliente – Servidor. Se enfoca en aprovechar la transmisión progresiva de datos en

múltiples resoluciones para poder llevar a cabo el Rendering Progresivo en la máquina cliente. Es necesario crear algoritmos y estructuras de datos que permitan reconstruir el volumen progresivamente, de tal modo que sea posible realizar el Rendering de Volúmenes de los datos en resoluciones intermedias.

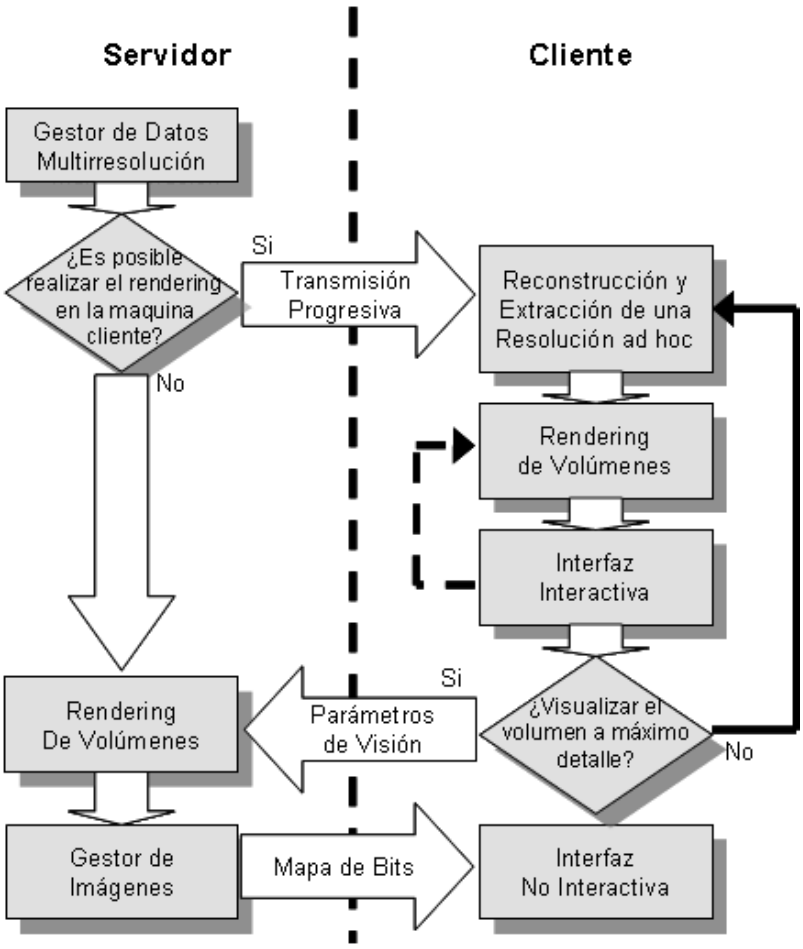

Figura 30: Diagrama de flujo de datos del proceso de visualización ad hoc.

6.1 Visualización Remota

En 1987, McCormick, de Fanti y Brown ([McCormick87]) establecen los fundamentos para la visualización distribuida, a la que llamaron *televisualización*, sugiriendo el uso de una supercomputadora para ejecutar cálculos científicos y de estaciones de trabajo para realizar el rendering; entre ambas está el concepto de *"image computer"*, la cual prepara los datos para ser visualizados. Este concepto es el antecesor del modelo del flujo de datos ([Haber90], [Martig03]). McCormick *et al* [McCormick87] plantean los Ambientes de Visualización Modulares (MVE - Modular Visualization Environments), constituidos por módulos independientes que realizan algún tipo de transformación sobre los datos. Típicamente, los datos pasan de un módulo a otro siguiendo un patrón de flujo de datos. El conjunto de módulos incluye un módulo inicial responsable de

la lectura de los datos y uno final encargado de realizar el rendering. El conjunto puede contener varios módulos intermedios, cada uno con una funcionalidad específica; por ejemplo, un módulo responsable de extraer una iso-superficie. La ejecución de los módulos puede realizarse en secuencia o en paralelo. En el caso en que los módulos se ejecuten en paralelo, resulta sencilla su implementación en base a un modelo distribuido.

Este modelo incluye un sofisticado editor gráfico que permite construir las aplicaciones seleccionando los módulos a utilizar y el enlace entre los mismos. De este modo el usuario diseña su propia línea de producción o *pipe line*. En este sentido, el *diseño* de la visualización es independiente del software de visualización. Ejemplos de estas aplicaciones son IRIS Explorer ([IRIS]) e IBM Data Explorer ([OVDE]). La Figura 31 ilustra el enlace de tres módulos que se ejecutarán para lograr la visualización de iso-superficies poligonales secuencialmente; también se detallan las correspondientes interacciones con el cliente.

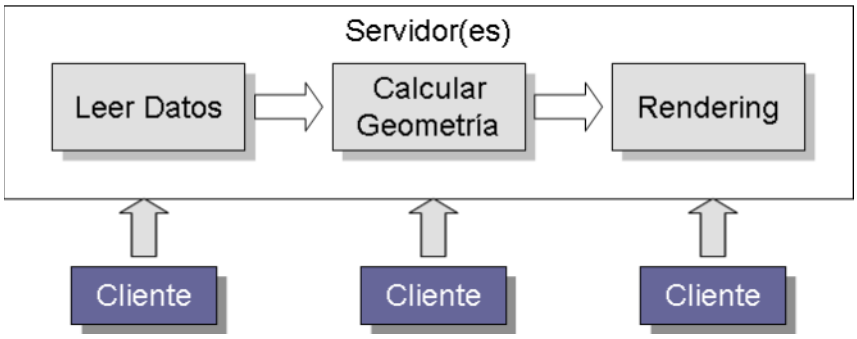

Figura 31: Visualización de Iso-Superficies como Flujo de Datos (McCormick *et al*)

El modelo de flujo de datos continúa utilizándose y generalmente es la base de todos los sistemas de visualización distribuida así como de varios sistemas *stand alone*.

En las últimas décadas, Internet se ha constituido en un ambiente de computación distribuida. Los Servicios Web representan una herramienta esencial aportando la interoperabilidad necesaria entre plataformas, así como la independencia de lenguajes de programación. Uno de los desafíos en el desarrollo actual es identificar los componentes adecuados para visualización distribuida usando Web Services o Grid Services basados en OGSA (Open Grid Services Architecture). Para analizar el modelo de servicios necesitamos determinar cuáles componentes son provistos por el Servidor y cuáles por el Cliente.

6.1.1 Visualización del lado de Cliente

Para determinar dónde tiene lugar la ejecución de la visualización, distinguimos los sistemas de visualización del lado del Cliente de los del lado del Servidor. Primero consideramos el caso en el que la visualización se ejecuta del lado del Cliente.

Un aspecto clave es la ubicación de los datos a ser visualizados; estos pueden estar centralizados en un servidor de acceso público o pueden estar directamente en la máquina cliente. La dimensión de los datos, en términos de tamaño, es un aspecto a considerar si tienen que ser transmitidos para su visualización. De manera general, la provisión de datos puede ser vista como un Servicio. Existen diferentes aproximaciones para modelar la visualización del lado del Cliente, dependiendo de cuánto software se asume que tiene instalada la máquina cliente y de cuánto es transferido por la red según se requiera.

(i) Diseño de la Visualización y Software de Visualización Presentes en el Cliente.
En este modelo los datos residen remotamente y el software de visualización reside en el cliente y se ejecuta localmente. Típicamente, los datos son recogidos de una URL como un tipo particular de datos MIME. Los archivos remotos de gran tamaño son difíciles de manejar, especialmente si sólo una pequeña parte de ellos es requerida; un desarrollo posterior de este modelo permite extraer un subconjunto del dataset remoto.

(ii) Diseño de la Visualización en el Servidor y Software de Visualización Presente en el Cliente.
Como mencionamos, podemos distinguir entre el diseño de la visualización y el software de visualización. Por ejemplo para un sistema MVE el diseño consiste en el enlace de módulos que componen el *pipeline* de visualización. Este diseño puede residir remotamente. Se puede ver al sistema MVE como un espacio de trabajo vacío dentro del cual un diseño puede ser ejecutado. Tanto los datos como el diseño pueden ser recogidos del servidor.

(iii) Diseño de la Visualización y Software de Visualización en el Servidor.
En este modelo, tanto el diseño y como el software de visualización están ubicados remotamente. El software es descargado a la máquina Cliente donde se ejecuta la visualización. Típicamente, el software descargado es una Applet de Java que se ejecuta dentro del framework de la máquina virtual de Java.

6.1.2 Visualización del lado del Servidor

Consideramos arquitecturas en las que la ejecución de la visualización se lleva a cabo en el servidor.

(i) Mostrar Imágenes en el Cliente.
En este modelo todos los componentes de la visualización tienen lugar en el Servidor. El cliente puede especificar los parámetros del proceso de visualización y eventualmente recibe el resultado de la visualización como una imagen, con lo cual se pierde interactividad.

(ii) Rendering de una Representación Intermedia en el Cliente.

Este modelo plantea llevar a cabo parte de la visualización remotamente y transferir una representación intermedia a la máquina cliente. El cliente se encarga de realizar el rendering y mostrar el resultado interactivamente.

6.1.3 Visualización del lado de Cliente vs. Visualización del lado de Servidor

Los modelos de Visualización del lado del Cliente asumen que el cliente tiene poder de procesamiento, recursos de memoria y espacio de almacenamiento secundario suficientes para realizar el rendering; también se considera que el tamaño de los datasets es mediano debido a que tienen que ser transmitidos por la red desde un servidor remoto. En muchos casos los datos a visualizar son un subconjunto de los datos originales. Este subconjunto es generado por el servidor como respuesta a una interacción del usuario ([Martig03]). El poder de procesamiento de la máquina cliente es un requerimiento crucial en estos enfoques, ya que los mismos permiten realizar la visualización en forma interactiva.

La Visualización del lado del Servidor, llevada al extremo de transmitir sólo una imagen al cliente, parece adecuada para visualizaciones de datos cuyo volumen sobrepasa la capacidad de procesamiento local, pero disminuye considerablemente, sino totalmente, la interactividad en la exploración. En el caso en que el rendering se lleve a cabo en el cliente, éste debe poseer la suficiente capacidad de procesamiento; por otro lado, también debe considerarse que deben transmitirse los datos al cliente, lo que implica que debe tenerse en cuenta cuán eficientemente puede hacerse esto.

Como veremos más adelante la Visualización Progresiva (Rendering Progresivo) intenta solucionar los problemas planteados en el caso de la Visualización del lado del Servidor, permitiendo el rendering interactivo en el cliente al explotar los recursos que éste posee, que por lo general son moderados. Además se basa en un modelo de transmisión progresiva de datos que es muy eficiente.

6.1.4 Visualización Progresiva Cliente – Servidor

El rendering progresivo es un mecanismo que computa el rendering en baja resolución, luego incrementa gradualmente la resolución para mejorar la calidad. La Figura 32 muestra tres pasos del rendering progresivo de un volumen.

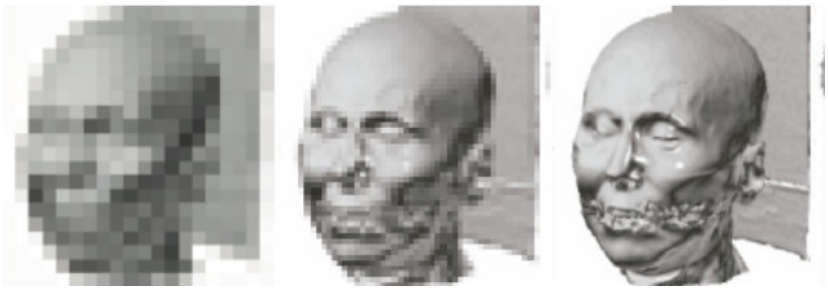

Figura 32: Rendering Progresivo

El rendering progresivo es un mecanismo que permite al usuario tener rápidamente una idea anticipada de cuál será el resultado final del rendering e interactuar con el pipe de visualización brindando la realimentación necesaria sin tener que esperar hasta que se realice el rendering a máxima resolución. Además, el rendering a baja resolución puede ser suficientemente eficiente para llevarse a cabo en tiempos interactivos.

Rorad y Jones ([Rorad06]) plantean un modelo de rendering progresivo de volúmenes para un ambiente distribuido. Ellos proponen una arquitectura general de visualización basada en agentes, en la cual un agente se encarga de administrar y ejecutar cada módulo de la visualización. Cada agente (componente) puede estar ubicado en diferentes computadores, lo cual habilita la distribución de la visualización. En esta arquitectura, el proceso de rendering es en sí mismo un agente que usa su ambiente para generar una imagen (Figura 33)

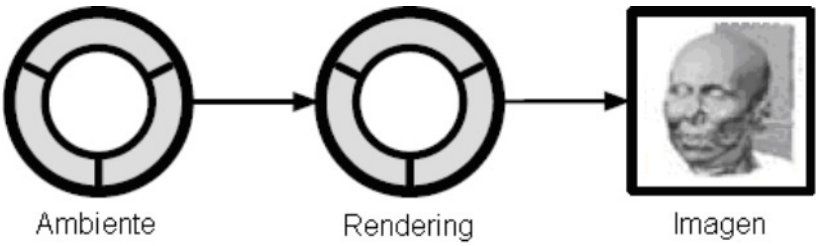

Figura 33: Arquitectura de rendering. El proceso de rendering es en si mismo un agente que toma decisiones según el ambiente en que se desenvuelve.

Para crear una estrategia de rendering progresivo, Rorad y Jones usan un agente, al cual se podrá conectar el cliente (de hecho, el rendering se realiza en el cliente), que obtiene su ambiente al momento de responder al protocolo de rendering. Este agente implementa el rendering progresivo en tres pasos; para esto requiere conectarse con tres agentes de rendering (Figura 34). Cuando recibe un pedido de rendering, el agente modifica la resolución en el *ambiente de rendering* y pasa el nuevo ambiente a los agentes de rendering. Cuando obtiene el resultado, lo transmite al cliente de la visualización, el cual es otro agente. La Figura 34 muestra la arquitectura del rendering progresivo basado en agentes.

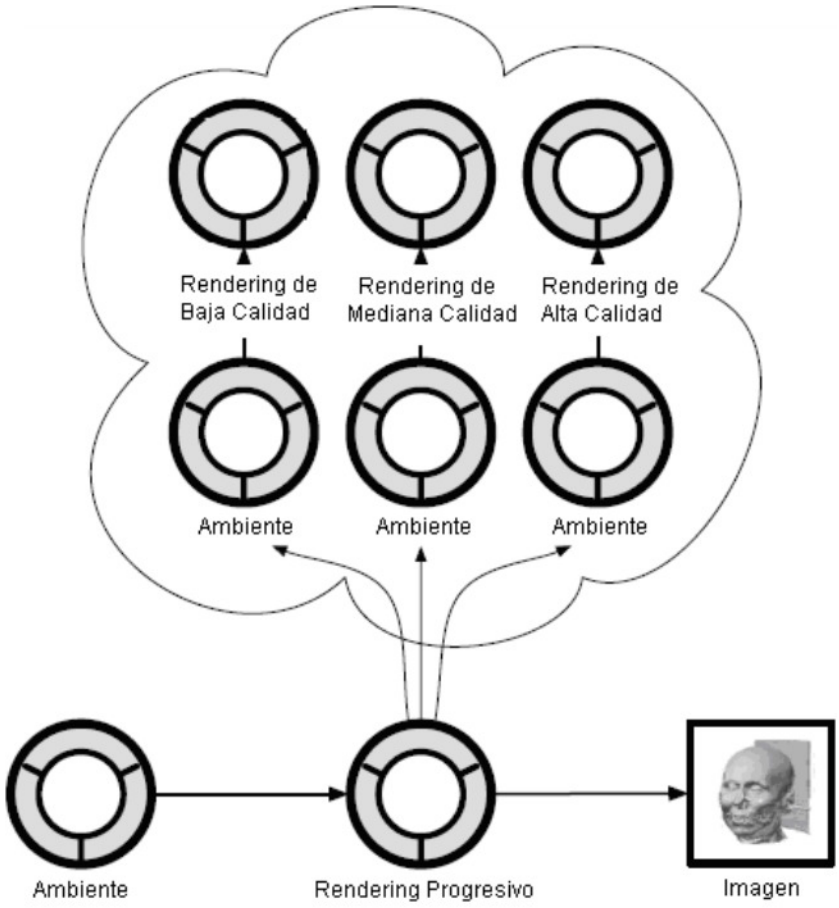

Figura 34: Agente de Rendering Progresivo.

La arquitectura propuesta por Rorad y Jones ([Rorad06]) no implica tener una representación especial de los datos ni un algoritmo especial para realizar el rendering progresivo; en cambio, se basa en una modificación interna del pipe de visualización que ejecuta en tres oportunidades el rendering tradicional pero con diferentes calidades. La propuesta de Rorad y Jones es general y no proveen más detalles acerca del rendering progresivo.

Callahan *et al* ([Callahan06]), proponen un sistema de visualización de redes tetraédricas no estructuradas de gran tamaño basado en la arquitectura cliente/servidor. Plantean el uso de la Transmisión Progresiva junto con la Visualización Progresiva para minimizar el tráfico de datos y obtener visualizaciones interactivas. Similar a nuestra propuesta, Callahan *et al* proponen realizar el rendering progresivo del lado del cliente y derivar el resto de los módulos al servidor. El servidor gestiona las porciones de datos a enviar, realizando ciertos procesos como Depth Culling, Frustum Culling, ordenamiento aproximado con respecto al punto de vista, y compresión de datos. Al no trabajar con multirresolución de datos, toda la geometría debe ser transmitida al cliente, en orden Front To Back, por porciones. Sin embargo el cliente en todo momento sólo mantiene una porción de la geometría en memoria y no es necesario el almacenamiento externo de la misma. El sistema de Callahan *et al* trabaja en tres modos: *Modo Interactivo*, *Modo Progresivo* y *Modo Completo*.

Modo Interactivo: El servidor transmite la frontera del volumen al cliente. El cliente puede mostrar una vista previa interactiva del volumen, visualizando el Rendering de Volúmenes de la frontera del volumen; si ésta es demasiado grande para ser procesada por el cliente, el servidor envía una simplificación de la misma.

Modo Progresivo: El servidor transmite los datos por porciones. Cada porción corresponde a un rango de profundidad del volumen, comparable por analogía a una *Slice* o *Corte* usados en el rendering de volúmenes regulares. La geometría correspondiente a un rango de profundidad es ordenada en orden *Front-To-Back* y transmitida al cliente. El cliente procesa cada porción de geometría usando el algoritmo de rendering HAVS ([Callahan05]), el cual implementa una variante del *A-Buffer* en lenguaje de shaders para GPU, llamado *k-Buffer*. La mezcla coherente de los buffers permite realizar el Rendering de Volúmenes por partes, refinando la aproximación anterior en cada paso. En esta etapa, la visualización no es interactiva; el cliente refina la vista del rendering de volúmenes de forma similar al refinamiento de imágenes JPEG transmitidas vía Web. El proceso de refinamiento se puede cortar en cualquier momento si se desea.

Modo Completo: Se interrumpe la transmisión de datos. Esto puede suceder debido a una petición del usuario o a que el servidor termina la transmisión de todo el volumen, luego se genera un archivo *mapa de bits* con la imagen final producida por el Rendering de Volúmenes, la cual se guarda localmente para su análisis.

A diferencia de nuestra propuesta, la Visualización Progresiva de Callahan *et al* no permite interactividad durante la transmisión de datos, es decir, las interacciones ocurren, previamente a la transmisión, con una vista de la frontera del volumen. La implementación de esta propuesta está ligada a un método de DVR en particular (el método HAVS). Sin embargo, puede tratar con volúmenes de topologías arbitrarias y no restringe la calidad del rendering progresivo a la cantidad de recursos de la máquina cliente, por lo que no es necesario el rendering del lado del servidor.

6.2 Visualización Progresiva en el Dominio Wavelet

Recientemente se han planteado modelos volumétricos multirresolución basados en redes tetraédricas que permiten tanto la compresión como la transmisión progresiva de los mismos; estos modelos se basan en dos extensiones de wavelets sobre dominios tetraédricos ([Castro97], [Boscardin01], [Castro05], [Castro06]). La transmisión de tales redes tetraédricas es muy eficiente, ya que se puede hacer por paquetes, siendo la red base la única geometría que se envía (Figura 35a). Posteriormente, sólo es necesario transmitir los coeficientes de detalle extraídos en la fase de análisis, los cuales adicionalmente pueden estar comprimidos

(Figura 35b). Este modelo de transmisión progresiva de redes tetraédricas es nuevo y constituye el punto de partida de la presente tesis.

En este apartado nos centramos en la Visualización Progresiva de tales redes tetraédricas (Figura 36). Nuestro algoritmo es apropiado para visualizar datos transmitidos progresivamente por la red, ya que no se necesita la presencia de todo el detalle para realizar el rendering. De hecho, la única geometría necesaria es la de la malla base, ya que el resto de la geometría se va generando conforme sea necesario, permitiendo que la visualización se adapte a la cantidad de detalles disponibles. La única restricción que deben cumplir los volúmenes es tener la propiedad de conectividad de subdivisión; esta propiedad debe verificarse para todo el volumen excepto para la red base.

Figura 35: Transmisión progresiva de datos en el dominio wavelet. Primero se transmite la malla base; luego se transmiten los detalles en secuencia. El orden en que llegan a la máquina cliente no está garantizado.

Nuestro algoritmo no impone ninguna restricción en cuanto a la cantidad de detalle que se visualizará en el Cliente. Simplemente se usan los detalles que se tienen disponibles y se adapta la resolución del modelo, mediante refinamiento selectivo, para mostrarlos. Por lo tanto, en el contexto de un sistema de visualización remoto, el proceso del lado del Servidor es el encargado de transmitir parte o toda la jerarquía de detalles según sea conveniente. Sin embargo, la disponibilidad de recursos para la visualización, en términos de memoria y almacenamiento secundario, no está garantizada para todos los Clientes. Para solucionar este inconveniente, planteamos dos alternativas para estimar la cantidad de detalle que será transmitida:

i) El Cliente puede cortar la recepción de detalles si carece de recursos para el rendering de los mismos. En este caso, la decisión de cuánto detalle mostrar será tomada por el cliente, de acuerdo a sus propios recursos.

ii) El Servidor, basándose en información estadística provista por el cliente, estima la cantidad de detalle que el cliente podrá procesar y simplemente los envía progresivamente.

Los detalles son agregados progresivamente al volumen mediante un esquema de refinamiento selectivo y síntesis de wavelet. En la Figura 36 detallamos el proceso de visualización progresiva. Conforme llegan los coeficientes de detalle al cliente, son insertados ordenadamente en un Buffer de Detalles. Este Buffer será consultado oportunamente por el algoritmo, ya que en base a los coeficientes de detalle disponibles se deberá decidir la resolución que tendrá el volumen.

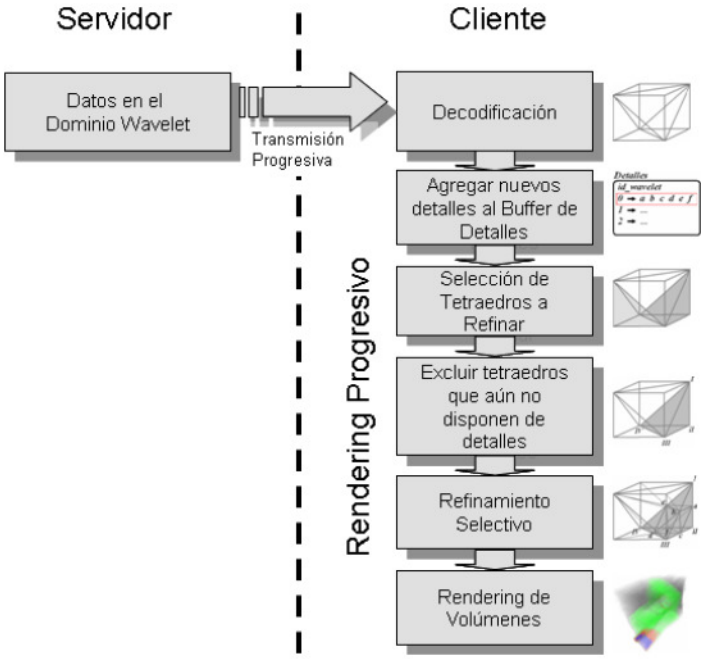

Figura 36: Diagrama de flujo del proceso de Visualización Progresiva.

Cada nivel de refinamiento regular (uniforme) aumenta exponencialmente la cantidad de tetraedros de la malla. En la Figura 37 se puede apreciar el aumento de detalles en la malla al pasar de un nivel de refinamiento al siguiente. Es por ello que, para lograr la visualización en tiempos interactivos, nuestro algoritmo refina selectivamente la malla de acuerdo a dos heurísticas, ambas dependientes de la ubicación del observador. La primera busca refinar aquellos tetraedros cuya proyección en perspectiva no cumpla cierta tolerancia de error medida en píxeles y la segunda busca realizar un *coarsening* de aquellos tetraedros que estén fuera de la pirámide de visión.

Figura 37: Rendering de Volúmenes del dataset SPX. De izquierda a derecha se muestran distintos LODs uniformes extraídos mediante el refinamiento de la malla.

El algoritmo desarrollado es simple y robusto; se enfoca en realizar los refinamientos y *coarsening*[2] sobre conjuntos grandes de tetraedros para aumentar su eficiencia. Debido a que adoptamos la estrategia de refinamiento rojo/verde de Bey ([Bey95]), generamos mallas tetraédricas de buena calidad; esto redunda en la calidad de nuestra visualización.

En las siguientes secciones revisaremos las características y la estructura de los volúmenes a los cuales aplicaremos el algoritmo de Visualización Progresiva. Luego, detallaremos las estructuras de datos que se utilizarán para representar estos datos. Posteriormente explicaremos los algoritmos de Refinamiento y *Coarsening* necesarios para realizar el Refinamiento Selectivo del volumen. Finalmente, explicaremos el algoritmo de Refinamiento Selectivo así como las heurísticas utilizadas para extraer las resoluciones *ad hoc* de los datos.

6.2.1 Representación de la Red Base y de los Coeficientes de Detalle

Partimos del modelo multirresolución propuesto por Castro ([Castro05]), que se basa en una *red semi-regular* de tetraedros que consiste en una Red Base de topología *irregular* la cual es refinada recursivamente de forma *regular*. El esquema de refinamiento adoptado por Castro es el de Bey ([Bey95]), que consiste en subdividir cada tetraedro recursivamente en ocho subtetraedros regulares. La red resultante debe poseer la propiedad de *conectividad de subdivisión*; la misma asegura que cada tetraedro pertenece a un espacio anidado de funciones. Esta propiedad implica que los tetraedros que pertenecen a resoluciones consecutivas poseen una relación padre/hijo;

La red tetraédrica tiene asociado un conjunto de atributos. La interpolación de estos define una función continua de la cual la red es el dominio. Algunas redes tienen los atributos asociados a los tetraedros y otras a los vértices. Podemos asumir, sin perder generalidad, que la red posee atributos asociados a los vértices. En la Figura 38 se aprecia el diagrama de clases de una representación compacta para una malla de tetraedros *semi-regular* con atributos asociados a los vértices. En esta representación los tetraedros comparten vértices y aristas, es decir, poseen adyacencia en ellos. Posteriormente tomaremos como base esta arquitectura para representar la malla multirresolución.

[2] Dado que no hay palabra en castellano que sea el antónimo de refinar, se utiliza el verbo en inglés *to coarsen*, que significa pasar de una mayor a una menor resolución.

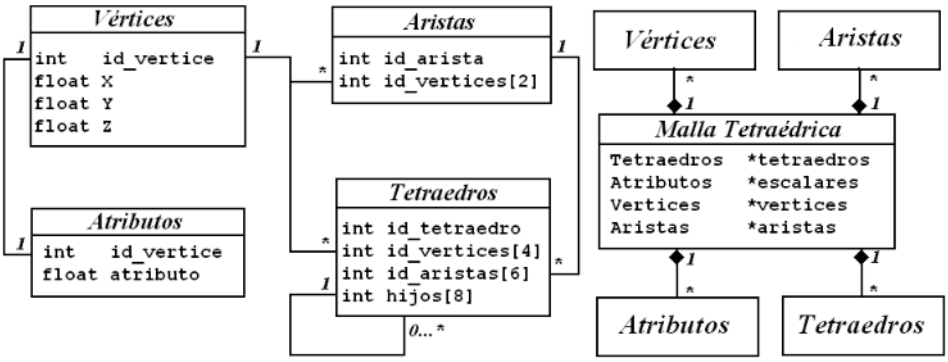

Figura 38: Diagrama de clases de una malla de tetraedros semi-regular.

Al aplicar la transformada wavelet a la red tetraédrica obtenemos la Red Base y un conjunto de Coeficientes de Detalle. El conjunto de los Coeficientes de Detalle, al cual llamaremos simplemente *detalles* de ahora en adelante, está configurado internamente de forma jerárquica. Cada tetraedro de la malla base posee su propia jerarquía de detalles. Esta jerarquía esta configurada, de hecho, como un árbol octal. En otras palabras, cada tetraedro de la malla base está asociado a un árbol octal de detalles y la malla base, por ende, está asociada a una *foresta de detalles*.

En los árboles de detalles, cada nodo contiene seis Coeficientes de Detalle necesarios para refinar un tetraedro en ocho subtetraedros. Además, posee enlaces a sus ocho nodos hijo. Nosotros proponemos una representación eficiente en memoria de la jerarquía de detalles a través de un *Heap*. En esa representación, los nodos del árbol se guardan ordenadamente en un arreglo unidimensional. La posición en el arreglo de un nodo *i* cuyo nodo padre está en la posición *t* está dada por la función *id_wavelet(t, i)*, definida a continuación:

$$id\_wavelet(t,i) = n + 8t + i \tag{42}$$

donde n es el número de tetraedros de la malla base. El número *i* debe estar en el rango de 0 a 7; este número denota el orden de creación de los hijos del nodo *t*. Las raíces de los árboles octales ocupan las primeras posiciones del arreglo, es decir, las posiciones de 0 a *n-1*. Se deduce que las raíces corresponden a los detalles de la malla base. La Figura 39 muestra la relación que existe entre los tetraedros y los detalles en el diagrama de clases. Por supuesto, es necesario agregar el campo id_wavelet a cada tetraedro para establecer su relación con los detalles.

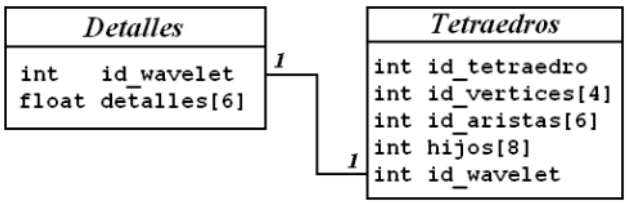

Figura 39: Inclusión de los detalles en el diagrama de clases de la malla.

A continuación describiremos en detalle el proceso de análisis multirresolución en base de wavelets. En el análisis multirresolución se aplica un filtro de wavelet a los atributos de una malla *semi-regular* de tetraedros para obtener una representación en baja frecuencia de los mismos y poder reducir la resolución de la malla. Al aplicar un filtro de wavelet a los atributos de un tetraedro padre T se obtienen seis coeficientes de detalle y la malla pierde resolución al eliminarse los hijos de T. Si bien el proceso está descrito en trabajos anteriores, no se ha publicado un algoritmo concreto para llevarlo a cabo. Nosotros proponemos un algoritmo completo, a saber el primero.

En la Figura 40 mostramos el algoritmo de refinamiento regular propuesto por J. Bey ([Bey95]), que es la base de nuestro algoritmo de refinamiento regular y debe ser visto en detalle para poder comprender el algoritmo de análisis en base de wavelets. Este algoritmo subdivide el tetraedro T, cuyos vértices son (x0, x1, x2, x3), en ocho subtetraedros. Con este fin, es necesario introducir seis nuevos vértices en los puntos medios de las aristas de T. Por ejemplo, el vértice x01 es el nuevo vértice introducido en la arista (x0, x1). Notamos que los índices de vértices del tetraedro padre (x0, x1, x2, x3) están presentes en los cuatro primeros tetraedros hijo. Notamos también que los índices de los seis vértices intermedios (x01, x02, x03, x12, x13, x23) también están presentes en los primeros cuatro tetraedros de forma simétrica. En base a estas premisas definiremos más adelante el algoritmo de análisis.

```
Proc    RegularRefinement( T )

        /* divide T = [x0, x1, x2, x3] into the subtetrahedra Ti, 1 ≤ i ≤ 8 */

        T1 = [ x0, x01, x02, x03]
        T2 = [x01,  x1, x12, x13]
        T3 = [x02, x12,  x2, x23]
        T4 = [x03, x13, x23,  x3]
        T5 = [x01, x02, x03, x13]
        T6 = [x01, x02, x12, x13]
        T7 = [x02, x03, x13, x23]
        T8 = [x02, x12, x13, x23]
Fin Proc
```

Figura 40: Algoritmo de refinamiento regular de J. Bey.

En la Figura 41 mostramos el algoritmo global del análisis en base de wavelets de una malla con la propiedad de conectividad de subdivisión. El algoritmo recorre los niveles de detalle comenzando a partir del penúltimo nivel de detalle más fino (el primero que puede ser *coarsed*) y termina con la malla base que corresponde al nivel cero. Para cada nivel se calculan los detalles de cada tetraedro mediante la aplicación de cierto filtro wavelet y se insertan ordenadamente en tabla de detalles usando como clave el id_wavelet del tetraedro. La Figura 42 muestra el algoritmo que aplica la transformada de wavelet a un tetraedro previamente refinado con el algoritmo de J. Bey ([Bey95]).

```
Proc    AnalisisWavelet( Malla, MaxNivel )
        n ← MaxNivel - 1;
        Mientras n ≥ 0
                Para cada tetraedro t de nivel n en Malla
                        detalles ← Filtro_Wavelet( t, Malla )
                        tablaDetalles ← t.id_wavelet
                        tablaDetalles ← detalles[0]
                        tablaDetalles ← detalles[1]
                        tablaDetalles ← detalles[2]
                        tablaDetalles ← detalles[3]
                        tablaDetalles ← detalles[4]
                        tablaDetalles ← detalles[5]
                Fin Para
                n ← n - 1
        Fin Mientras
Fin Proc
```

Figura 41: Algoritmo global de análisis en base de wavelets.

Para aplicar un filtro de wavelet y encontrar los Coeficientes de Detalle, es necesario contar con los diez atributos que posee el tetraedro. Cuatro de esos atributos se encuentran en los vértices del tetraedro, mientras que los otros seis, se encuentran en los vértices de sus tetraedros hijos. En el algoritmo de la Figura 42 se aprovecha el hecho de que cada tetraedro comparte los mismos índices de vértices con sus primeros cuatro hijos. El algoritmo puede identificar mediante este lazo cuáles son esos hijos y mediante ellos identificar los índices de los otros seis vértices que conforman el tetraedro refinado. Podemos entonces extraer los atributos asociados a esos vértices y calcular los coeficientes de detalle utilizando las wavelets adecuadas para los atributos definidos en los vértices.

```
Funcion    Filtro_Wavelet( tetra, Malla ) : flotante[]
   hijos ← Malla.descendenciaTetra( tetra )
   coeficientes[0] ← Malla.escalares[ tetra.ivertices[0] ]
   coeficientes[1] ← Malla.escalares[ tetra.ivertices[1] ]
   coeficientes[2] ← Malla.escalares[ tetra.ivertices[2] ]
   coeficientes[3] ← Malla.escalares[ tetra.ivertices[3] ]
   Para cada hijo h en hijos
      Si h.ivertices[0] = tetra.ivertices[0]
         func[0] ← Malla.escalares[ h.ivertices[1] ] /* escalar entre 0 y 1 */
         func[1] ← Malla.escalares[ h.ivertices[2] ] /* escalar entre 0 y 2 */
         func[2] ← Malla.escalares[ h.ivertices[3] ] /* escalar entre 0 y 3 */
      Fin Si
      Si h.ivertices[1] = tetra.ivertices[1]
         func[3] ← Malla.escalares[ h.ivertices[2] ] /* escalar entre 1 y 2 */
         func[4] ← Malla.escalares[ h.ivertices[3] ] /* escalar entre 1 y 3 */
      Fin Si
      Si h.ivertices[2] = tetra.ivertices[2]
         func[5] ← Malla.escalares[ h.ivertices[3] ] /* escalar entre 2 y 3 */
      Fin Si
   Fin Para
   detalles[0] ← func[0] - (coeficientes[0] + coeficientes[1]) * 0.5
   detalles[1] ← func[1] - (coeficientes[0] + coeficientes[2]) * 0.5
   detalles[2] ← func[2] - (coeficientes[0] + coeficientes[3]) * 0.5
   detalles[3] ← func[3] - (coeficientes[1] + coeficientes[2]) * 0.5
   detalles[4] ← func[4] - (coeficientes[1] + coeficientes[3]) * 0.5
   detalles[5] ← func[5] - (coeficientes[2] + coeficientes[3]) * 0.5
   retorna detalles
Fin Funcion
```

Figura 42: Algoritmo que aplica la transformada de wavelet a un tetraedro.

6.2.2 Estructuras de Datos

Para realizar la visualización progresiva de una malla de tetraedros en el dominio de wavelets, es necesaria la adaptación de la resolución de la malla, mediante Refinamiento Selectivo, hacia una resolución *ad hoc*. En principio, una resolución *ad hoc* es aquélla que se adapta para mostrar todos los detalles disponibles en el Buffer de Detalles; sin embargo, es posible utilizar heurísticas que ayuden a seleccionar los tetraedros a refinar de acuerdo a criterios dependientes del punto de vista con el objetivo de reducir el número de tetraedros y en consecuencia el tiempo de rendering.

Debido a esto, las estructuras de datos que se utilizarán tienen como objetivo fundamental facilitar el Refinamiento Selectivo eficiente de la malla. Además, el refinamiento selectivo debe producir una malla adecuada para el Rendering de Volúmenes. Muchos algoritmos de Rendering de Volúmenes, tales como el de Proyección de Tetraedros ([Shirley90], [Stein94], [Wylie02], [Weiler02], [Weiler04]), utilizan estructuras indexadas porque esto permite resolver eficientemente las referencias a vértices y atributos al tiempo que minimizan el uso de memoria. Es por ello que la malla generada por el Refinamiento Selectivo debe ser indexada y debe considerar también el cálculo y almacenamiento de las normales de las caras de los tetraedros.

Se debe tener en cuenta el espacio que ocupará la estructura de datos en la memoria del sistema. Es importante minimizar el almacenamiento de datos repetidos, ya que se deben mantener varias resoluciones de los datos en memoria, es decir que cada resolución debe compartir la mayor cantidad de datos posible con las demás. Es necesario usar un sistema de indexado que habilite esta unicidad de datos y agilice la extracción de información. Sin embargo, un abuso en el indexado puede resultar contraproducente y agravar el problema de almacenamiento. Es evidente que se necesita encontrar un balance que permita un almacenamiento eficiente y a la vez un acceso a los datos de forma rápida.

Nosotros consideramos el uso de una estructura de datos similar a la que está planteada en el diagrama de clases de la Figura 38. Aún cuando esta estructura es sencilla y compacta, es necesario modificarla y extenderla para alcanzar nuestros objetivos. Consideremos la tarea de Refinamiento Selectivo en la cual es necesario agregar y quitar tetraedros de la malla, sin agregar vértices, aristas ni normales repetidas a la misma; en esta tarea también es necesario verificar que la eliminación de aristas se realizará sólo cuando éstas no son referenciadas por ningún tetraedro. Se hace necesario realizar búsquedas de vértices, de aristas y de normales antes de insertar un nuevo tetraedro a la malla, de modo que se utilicen los vértices, aristas y normales existentes. Para ello es imperativo definir índices que nos permitan conocer rápidamente si un vértice ya existe en el arreglo de vértices; lo mismo sucede con las normales y las aristas.

En nuestra propuesta, la malla tetraédrica está conformada por un conjunto de seis arreglos dinámicos y un conjunto de tres índices dinámicos. Ambos conjuntos forman nuestra base de datos. El conjunto de arreglos dinámicos lo conforman los arreglos de vértices, escalares, tetraedros, normales, aristas y

detalles. El conjunto de índices dinámicos lo conforman el índice de vértices, de aristas y de normales. Todos los arreglos guardan datos únicos para todo el volumen. El arreglo de detalles almacena los detalles que llegan progresivamente desde el servidor (Figura 44). Dado que se tienen los atributos correspondientes a los vértices, se utilizará el esquema de wavelets basado en vértices explicado en el capítulo 5 sección 5.3.4.

Cada tetraedro consta de cuatro referencias al arreglo de vértices, referencias que además se utilizan para acceder al arreglo de escalares; asimismo posee cuatro referencias al arreglo de normales y seis referencias al arreglo de aristas (Figura 43). Si bien cada tetraedro posee 14 referencias en total, esta representación es económica, ya que todos los tetraedros comparten vértices, escalares, normales y aristas, lo cual aumenta la escalabilidad del sistema cuando crece el número de tetraedros. En el momento de la creación de cada tetraedro, se le asigna un id_wavelet, el cual se calcula de forma idéntica que el id_wavelet de los detalles, es decir, basado en el orden de creación del tetraedro con respecto a sus hermanos y al id_wavelet de su padre. Esta referencia será utilizada para encontrar los coeficientes de detalle que le corresponden en caso de refinamiento.

```
class   Tetraedro
{
   unsigned int      ivertices[4];
   unsigned int      inormales[4];
   unsigned int      iaristas[6];
   unsigned int      id_wavelet;
   unsigned int      padre;
   unsigned int      hermano;
   unsigned int      hijo;
   unsigned short    flags;
   float             diametro;
};
```

```
class   Malla
{
  array<vector3<float>> vertices;
  array<float>          escalares;
  array<Tetraedro>      tetraedros;
  array<vector3<float>> normales;
  mapaAristas           aristas;
  detalles              tablaDetalles;
  octree                indiceVerts;
  gaussMap              indiceNormales;
  garbageCollector      indiceTetras;
  unsigned int          numTetrasBase;
};
```

Figura 43: Estructuras de datos para un tetraedro y para la malla indexada de tetraedros.

Durante el Refinamiento Selectivo es necesario que los arreglos de la malla crezcan dinámicamente cuando se rebasa su límite superior. Consideramos que deben sobre-dimensionarse cierta cantidad de elementos para favorecer la performance del sistema. Un crecimiento moderado favorecerá también el ahorro de memoria. En la práctica, un crecimiento de 64 ó 128 elementos resulta adecuado.

```
class    mapaAristas
{
    struct    arista
    {
        int   a, b;
    };
    arrayDeBits              marcas;
    array<char>              links;
    array< arista >          aristas;
    hash_multimap<int,int>   indice;
    queue<int>               eliminados;
};
```

```
class   detalles
{
  struct       detalle
  {
    unsigned int       id_wavelet;
    float              detalles[6];
  };
  array< detalle >     detalles;
};
```

Figura 44: Estructuras de datos para las aristas y detalles de la malla de tetraedros

El arreglo de vértices está indexado por un Octree que encierra espacialmente a la malla base y se subdivide uniformemente hasta que cada nodo terminal tenga un

radio proporcional al 3% del radio del nodo raíz, esto es aproximadamente 5 niveles de subdivisión. El arreglo de normales está indexado por un mapa gaussiano, similar al presentado por Kumar *et al* ([Kumar96]), y por Zhang y Hoff ([Zhang97]) para hacer *backface culling* jerárquico. Sin embargo, en lugar de usar una esfera de radio uno, nosotros utilizamos un cubo que encierra a dicha esfera. Cada cara del cubo es subdividida por un *kd-tree*. Luego cada pequeño nodo del *kd-tree* es proyectado en la superficie de la esfera. De este modo las normales que son insertadas en el mapa gaussiano quedan agrupadas en los nodos de los *kd-trees*. Esto nos permitirá buscarlas rápidamente dentro de los *kd-trees*.

La Figura 45 muestra el mapa gaussiano inscrito en un cubo alineado a los ejes cartesianos. El *kd-tree* de la cara frontal está subdividido de manera similar a un árbol cuaternario y se puede apreciar la proyección de un nodo terminal en la superficie de la esfera. Las normales comprendidas dentro del ángulo sólido formado por esta proyección quedan agrupadas en el nodo del kd-tree.

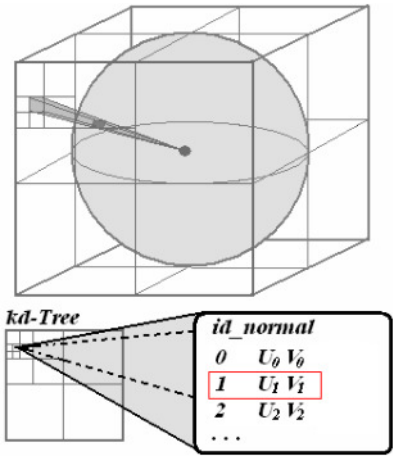

Figura 45: Mapa gaussiano usado para agrupar las normales en los nodos de varios kd-trees.

Para insertar una normal en un kd-tree primero tenemos que identificar a cuál cara del cubo interseca y luego proyectarla sobre el plano que esa cara contiene, obteniendo de este modo una coordenada (u, v). Después insertamos la coordenada (u, v) en el kd-tree asociado a esa cara. Debido a que el cubo está alineado con los ejes cartesianos, resulta sencillo identificar la cara que intersecará la normal, sólo tenemos que examinar el signo de la componente de la normal cuyo valor absoluto es el mayor. La proyección de la normal también es muy eficiente ya que los planos del cubo son paralelos a los planos cartesianos, las ecuaciones se reducen a una división por componente. En la práctica, sólo tres caras del cubo son utilizadas, ya que se agrupan las normales iguales pero de sentidos opuestos; esto reduce enormemente el espacio de almacenamiento de las mismas; sin embargo, dado que los tetraedros poseen referencias directas al arreglo de normales, siguen siendo accedidas rápidamente por el algoritmo de rendering.

El arreglo de aristas está indexado por una tabla Hash. La función de hashing (Figura 46) combina los dos índices de vértices que forman la arista (a, b) en un

número de 32 bits. Es necesario ordenar los índices a y b antes de aplicar la función de hashing, de tal modo que el índice “a” sea menor que el índice “b”.

```
unsigned int   hash_func( unsigned int a, unsigned int b )
{
        return ( (a & 1023) << 10 ) + (b & 1023);
}
```

Figura 46: función de hashing que indexa la arista (a, b).

Todos los índices previamente descritos son necesarios para asegurar que los vértices, aristas y normales sean únicos, ya que deben ser compartidos por los tetraedros. Esto quiere decir que durante la creación de un nuevo tetraedro, éste deberá buscar primero sus vértices en el arreglo de vértices e insertarlos sólo en caso de no encontrarlos. El mismo proceso debe seguirse con sus aristas y normales.

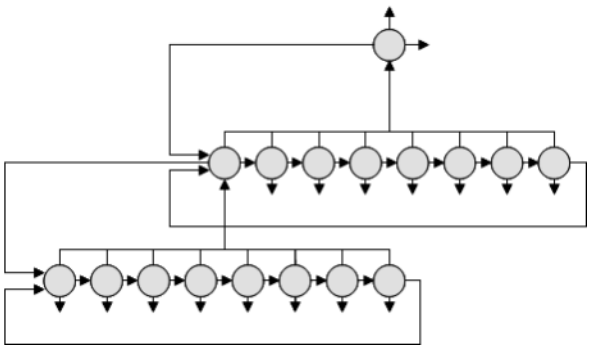

Figura 47: Organización interna de la jerarquía de tetraedros. Cada tetraedro, simbolizado por un círculo, posee tres referencias a otros tetraedros: padre, hermano e hijo.

Finalmente, el arreglo de tetraedros está organizado como se muestra en la Figura 47. Cada tetraedro guarda tres referencias a otros tetraedros: padre, hijo y hermano. La estructura jerárquica de la multirresolución es una foresta de árboles donde cada tetraedro de la malla base es la raíz de un árbol. Los tetraedros hijos de un mismo padre, están enlazados unos con otros de manera circular manteniendo una referencia a su hermano. Un tetraedro padre sólo necesita guardar el índice de un hijo y a través de él puede tener acceso a los demás. Por último, cada tetraedro hijo guarda el índice de su padre.

Los primeros elementos del arreglo de tetraedros corresponden a la malla base. Estos tetraedros son inamovibles. El resto de los tetraedros se crearán conforme se refina sucesivamente la malla y serán insertados en el arreglo de tetraedros, que aumentará su tamaño. Sin embargo, cuando un tetraedro es eliminado, el arreglo queda intacto, y su índice está a disposición para ser reutilizado posteriormente durante la creación de otro tetraedro. Esto sucede frecuentemente durante el Refinamiento Selectivo. La clase denominada *garbageCollector* lleva la cuenta de las posiciones vacías en arreglo de tetraedros y se encarga de su reutilización.

La eliminación de aristas es un poco más compleja. Cada arista mantiene un contador de referencias. Cada vez que es creado un tetraedro, éste encuentra sus

aristas en el arreglo de aristas (o en su defecto inserta nuevas aristas). Cada arista referenciada aumenta su contador; más tarde, cuando se eliminan tetraedros, los contadores de referencias de sus aristas disminuyen. Si algún contador llega a cero, la arista es marcada como disponible para ser reutilizada y se borra del índice de aristas (tabla hash).

El arreglo dinámico de detalles contiene los Detalles que cada tetraedro necesita para refinarse regularmente. Las inserciones en éste arreglo se hacen en orden para poder hacer búsquedas rápidas posteriormente. Los detalles se buscan por id_wavelet mediante búsqueda binaria.

6.2.3 Algoritmo de Refinamiento

Realizamos el refinamiento de la malla utilizando la técnica rojo/verde. Nuestro algoritmo es similar al desarrollado por González-Yuste *et al* ([González-Yuste03]) que es una versión eficiente orientada a objetos del algoritmo de Lohner *et al* ([Lohner92]). La diferencia con el algoritmo de González-Yuste *et al* reside en que éste requiere mantener la adyacencia de tetraedros por arista y por cara, es decir, las aristas y caras requieren mantener actualizada una lista de referencias a los tetraedros que las comparten. Además, el algoritmo de González-Yuste *et al* es recursivo y está orientado a hallar soluciones numéricas. Por otro lado, nosotros proponemos un algoritmo que no requiere mantener la adyacencia de tetraedros explícitamente y es iterativo. Nuestro algoritmo está enfocado en refinar conjuntos grandes de tetraedros. Con este fin, optamos por visitar todos los tetraedros terminales de la malla. Debido a ello, no es necesario guardar explícitamente ningún tipo de adyacencia de tetraedros. Nuestro algoritmo realiza el refinamiento en tres etapas: Marcado, Clasificación y Refinamiento.

En la *etapa de* ***Marcado***, se marcan las aristas de todos los tetraedros que se refinarán regularmente, esto implica marcar las seis aristas de cada tetraedro. Dado que los tetraedros comparten aristas, los tetraedros adyacentes se ven modificados por esta acción; el marcado de aristas es una operación sencilla y rápida. Mantenemos un arreglo de bits de la misma dimensión que el arreglo de aristas; un bit apagado simboliza que la arista no está marcada; de aquí que el marcado se reduce a prender un bit del arreglo. La Figura 48 muestra el pseudo código del algoritmo para la etapa de marcado.

```
Proc  MarcarTetraedrosRojos( listaTetras )
        Para cada tetraedro t en listaTetras
                MarcarTetraedro ( t )
        Fin Para
Fin Proc

Proc    MarcarTetraedro( Tetraedro )
        Para cada arista a en Tetraedro
                MarcarArista( a )
        Fin Para
Fin Proc
```

Figura 48: Marcado de tetraedros regulares para refinamiento.

En la *etapa de* ***Clasificación***, analizamos cada tetraedro y lo clasificamos según el número de marcas que tienen sus aristas. A continuación se describe el tipo de refinamiento que aplicaremos; éste depende únicamente del número de marcas que logra acumular cada tetraedro.

Tipo1 (Refinamiento Regular): todas las aristas del tetraedro están marcadas; el tetraedro se subdivide en ocho subtetraedros de igual volumen.

Tipo2: el tetraedro tiene marcadas tres aristas de una misma cara; el tetraedro se subdivide en cuatro subtetraedros.

Tipo3a: el tetraedro tiene marcadas dos aristas que no pertenecen a la misma cara; el tetraedro se subdivide en cuatro subtetraedros.

Tipo3b: el tetraedro tiene marcadas dos aristas de una misma cara; el tetraedro se subdivide en tres subtetraedros.

Tipo4: el tetraedro tiene marcada una arista; el tetraedro se subdivide en dos subtetraedros.

```
Proc    ClasificaMalla( Malla )
        Repetir
                EsConforme ← VERDADERO.
                ListaIrregulares ← Vacia.

                /*Clasifica los tetraedros según sus aristas marcadas*/
                Para cada tetraedro terminal t en Malla
                        Si t es irregular y tiene marcas
                           ListaIrregulares ← t
                           EsConforme ← FALSO.
                        Si No
                           Si t tiene 6 marcas: t ← Tipo1
                           Si t tiene 3 marcas en la misma cara: t ← Tipo2
                           Si t tiene 2 marcas no en la misma cara: t ← Tipo3a
                           Si t tiene 2 marcas en la misma cara: t ← Tipo3b
                           Si t tiene 1 marca: t ← Tipo4
                           En otros casos:
                                MarcarTetraedro( t )
                                EsConforme ← FALSO
                        Fin Si
                Fin Para

                /*refina regularmente a los padres de los tetraedros
                irregulares que tienen aristas marcadas*/
                Si ListaIrregulares no esta vacia
                        /*Crea una lista de padres*/
                        listaPadres ← Vacia
                        Para cada tetraedro t en ListaIrregulares
                           Si t.padre no esta en listaPadres
                                listaPadres ← t.padre
                           Fin Si
                        Fin Para
                        /*Refina los padres*/
                        Para cada tetraedro tp en listaPadres
                           EliminarSubTetraedros( tp )
                           MarcarTetraedro( tp )
                           tp ← Tipo1
                           RefinarTetraedro( tp, Malla )
                        Fin Para
                Fin Si
        Repetir Mientras Not EsConforme
Fin Proc
```

Figura 49: Clasificación de los tetraedros de la malla

Si un tetraedro posee cuatro o cinco aristas marcadas, no puede pertenecer a ningún tipo antes mencionado, por lo tanto, la malla no quedaría conforme. Esto sucede también con los tetraedros que poseen tres aristas marcadas que no pertenecen a la misma cara. En estos casos se deberá marcar las aristas que restan para completar seis marcas y el tetraedro quedaría clasificado como *Tipo1*. Adicionalmente, tenemos el caso en que un tetraedro irregular posee una o varias aristas marcadas; como no es válido refinar tetraedros irregulares, es necesario primero, eliminarlos. Con esta acción, el tetraedro padre de los tetraedros irregulares ocupa el espacio vacío que dejaron sus hijos. Luego, es necesario marcar las seis aristas del padre y refinarlo regularmente. Con la introducción de nuevas marcas nos vemos obligados a realizar una reclasificación de la malla. Esto puede causar un efecto dominó, ya que al reclasificar algunos tetraedros, potencialmente podemos introducir nuevamente otras marcas.

Claramente esta etapa es la más costosa del proceso. Al no guardar explícitamente la adyacencia de tetraedros por cada arista, que aumentaría innecesariamente el gasto de almacenamiento de memoria, optamos por un algoritmo iterativo que recorre sucesivamente los tetraedros terminales de la jerarquía multirresolución (Figura 49). Se considera que la aproximación recursiva planteada por González-Yuste *et al* ([González-Yuste03]), no es adecuada para nuestro propósito ya que podría agotar rápidamente la pila. El algoritmo de la Figura 49 optimiza el uso de memoria y es robusto al eliminar el problema del desbordamiento de pila.

Terminada la etapa de clasificación ya tenemos una malla conforme lista para ser refinada. La Figura 50 ilustra el refinamiento de cada tetraedro según su tipo:

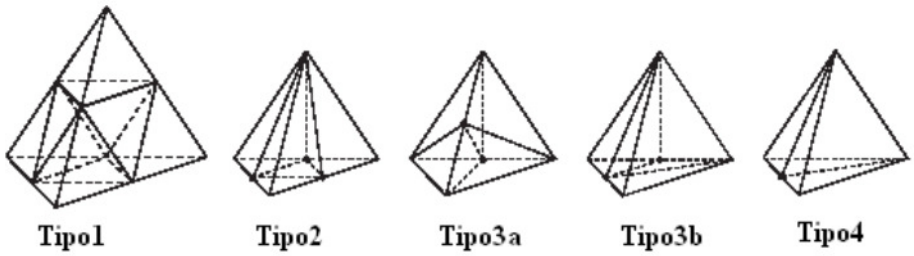

Figura 50: Cinco tipos de refinamiento. El Tipo1 corresponde a refinamiento regular, los demás a refinamiento irregular.

Después del refinamiento es necesario reconstruir el valor escalar definido en los nuevos vértices introducidos en las aristas del tetraedro padre. Con este fin, es necesario tomar los cuatro coeficientes de los vértices originales del tetraedro junto con los seis coeficientes de detalle y reconstruir los valores originales siguiendo el proceso de síntesis de wavelet. De este modo, obtenemos los seis escalares necesarios para aumentar el nivel de detalle de la malla. Los coeficientes de detalle deben ser buscados en el Buffer de Detalles usando el índice id_wavelet del tetraedro padre.

Debido a que los vértices de los tetraedros introducidos por refinamiento irregular son un subconjunto de los vértices introducidos por refinamiento regular, los

escalares reconstruidos en el proceso de síntesis pueden ser utilizados también para aumentar del detalle de los tetraedros irregulares.

Si se da el caso en que los detalles no están disponibles en el Buffer de Detalles, el refinamiento no se puede cancelar ya que la malla tiene que ser conforme. Entonces, interpolamos linealmente los cuatro coeficientes asociados a los vértices del tetraedro padre para obtener los seis escalares que hacen falta para el refinamiento. Esto introduce cierto error en el dominio, pero éste se mantiene bajo debido a que este caso solamente sucede cuando es necesario mantener la conformidad de la malla En la Figura 51 se muestra el algoritmo de refinamiento de un tetraedro, el cual toma en cuenta lo dicho anteriormente.

```
Proc    RefinarTetraedro( Tetra, Malla )
        coeficientes[ 0 ] ← Malla.escalares[ Tetra.ivertices[0] ]
        coeficientes[ 1 ] ← Malla.escalares[ Tetra.ivertices[1] ]
        coeficientes[ 2 ] ← Malla.escalares[ Tetra.ivertices[2] ]
        coeficientes[ 3 ] ← Malla.escalares[ Tetra.ivertices[3] ]

        detalles = Malla.tablaDetalles.buscarDetalles( Tetra.id_wavelet )

        Si existen los detalles
                func_original ← SintesisWavelet( coeficientes, detalles )
        Si No
                func_original[0] ← coeficientes[0]
                func_original[1] ← coeficientes[1]
                func_original[2] ← coeficientes[2]
                func_original[3] ← coeficientes[3]
                func_original[4] ← ( coeficientes[0] + coeficientes[1] ) * 0.5
                func_original[5] ← ( coeficientes[0] + coeficientes[2] ) * 0.5
                func_original[6] ← ( coeficientes[0] + coeficientes[3] ) * 0.5
                func_original[7] ← ( coeficientes[1] + coeficientes[2] ) * 0.5
                func_original[8] ← ( coeficientes[1] + coeficientes[3] ) * 0.5
                func_original[9] ← ( coeficientes[2] + coeficientes[3] ) * 0.5
        Fin Si

        Si Tetra es de Tipo1:  RefinaTetraTipo1 ( Tetra, func_original )
        Si Tetra es de Tipo2:  RefinaTetraTipo2 ( Tetra, func_original )
        Si Tetra es de Tipo3a: RefinaTetraTipo3a( Tetra, func_original )
        Si Tetra es de Tipo3b: RefinaTetraTipo3b( Tetra, func_original )
        Si Tetra es de Tipo4:  RefinaTetraTipo4 ( Tetra, func_original )
Fin Proc
```

Figura 51: Refinamiento de un tetraedro según su tipo

En nuestra implementación, hacemos el análisis multirresolución sobre funciones definidas en los vértices de los tetraedros, por lo tanto usamos el esquema de wavelets basado en vértices descripto en el capítulo anterior. En la Figura 52 ilustramos el cálculo de los escalares de la función original utilizando los coeficientes del tetraedro padre y los coeficientes de detalle.

```
Funcion  SintesisWavelet( coeficientes[4], detalles[6] ) : flotante[]
   func_original[0] ← coeficientes[0]
   func_original[1] ← coeficientes[1]
   func_original[2] ← coeficientes[2]
   func_original[3] ← coeficientes[3]
   func_original[4] ← detalles[0] + (coeficientes[0] + coeficientes[1]) * 0.5
   func_original[5] ← detalles[1] + (coeficientes[0] + coeficientes[2]) * 0.5
   func_original[6] ← detalles[2] + (coeficientes[0] + coeficientes[3]) * 0.5
   func_original[7] ← detalles[3] + (coeficientes[1] + coeficientes[2]) * 0.5
   func_original[8] ← detalles[4] + (coeficientes[1] + coeficientes[3]) * 0.5
   func_original[9] ← detalles[5] + (coeficientes[2] + coeficientes[3]) * 0.5
   retorna func_original
Fin Funcion
```

Figura 52: Síntesis de wavelet para una función definida sobre los vértices del tetraedro

El algoritmo de Refinamiento Selectivo, el cual planifica los refinamientos, a menudo decide forzar el refinamiento de tetraedros irregulares. Esto se debe a que se actualiza el Buffer de Detalles y se dispone de los coeficientes de detalles. En esta situación, **antes de la etapa de clasificación**, es necesario eliminar los tetraedros irregulares y refinar regularmente al padre. El algoritmo de refinamiento completo se muestra en la Figura 53. Asimismo, en la Figura 54 se muestra el algoritmo que fuerza el refinamiento de tetraedros irregulares.

```
Proc    refinarRegulares( listaTetras, Malla )
        MarcarTetraedrosRojos( listaTetras )

        ClasificaMalla( Malla )

        Para cada tetraedro terminal t en Malla
                Si t es de Tipo1, Tipo2, Tipo3a, Tipo3b o Tipo4
                        RefinarTetraedro( t, Malla )
                Fin Si
        Fin Para
Fin Proc
```

Figura 53: Algoritmo de refinamiento

```
Proc    refinarIrregulares( listaTetras, Malla )
        /*Crea una lista de padres*/
        listaPadres ← Vacia
        Para cada tetraedro t en listaTetras
                Si t.padre no esta en listaPadres
                        listaPadres ← t.padre
                Fin Si
        Fin Para

        /*Refina los padres*/
        Para cada tetraedro tp en listaPadres
                EliminarSubTetraedros( tp )
                MarcarTetraedro( tp )
                tp ← Tipo1
                RefinarTetraedro( tp, Malla )
        Fin Para

        ClasificaMalla( Malla )

        Para cada tetraedro terminal t en Malla
                Si t es de Tipo1, Tipo2, Tipo3a, Tipo3b o Tipo4
                        RefinarTetraedro( t, Malla )
                Fin Si
        Fin Para
Fin Proc
```

Figura 54: Algoritmo que fuerza el refinamiento de tetraedros irregulares

En las Figuras 55 y 56 se muestra el Rendering de Volúmenes de datos en el dominio de wavelets. Se reconstruye el volumen en el dominio espacial usando el algoritmo de refinamiento descrito y se añaden los detalles al dominio mediante la síntesis de wavelet. Se puede notar como, en cada nivel de refinamiento, el detalle aumenta considerablemente. Ambos datasets han sido remuestreados para poder aplicar el análisis multirresolución de wavelets; esto repercute en la calidad del Rendering de Volúmenes.

Figura 55: Rendering de Volúmenes del dataset HARMONIC. De izquierda a derecha se muestran tres LODs extraídos mediante el refinamiento de la malla.

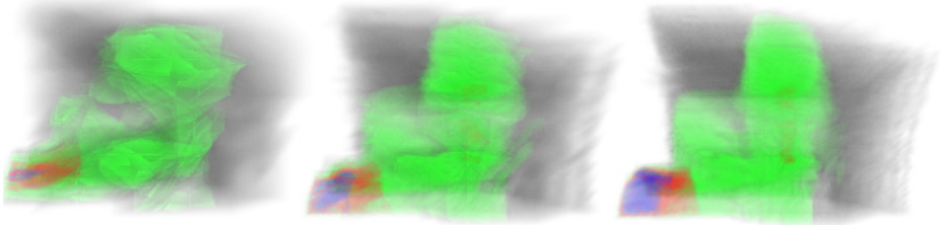

Figura 56: Rendering de Volúmenes del dataset SPX. De izquierda a derecha se muestran tres LODs extraídos mediante el refinamiento de la malla.

6.2.4 Algoritmo de Coarsening

Sólo es posible realizar el *coarsening* de los tetraedros refinados regularmente; dado que los tetraedros irregulares existen para hacer conforme la malla, el *coarsening* de los mismos carece de sentido. Sin embargo, junto con los tetraedros regulares, también aplicamos el *coarsening* sobre los tetraedros irregulares que son adyacentes a éstos, lo que provoca que el *coarsening* de los regulares sea verdaderamente efectivo. Esto puede verse más claramente en la Figura 57.

El algoritmo de *coarsening* que proponemos es original y permite volver a la malla base desde cualquier resolución debido a que podemos aplicarlo a conjuntos grandes de tetraedros. De forma similar que el algoritmo de refinamiento, dividimos al *coarsening* en tres etapas: Borrado, Clasificación y Refinamiento.

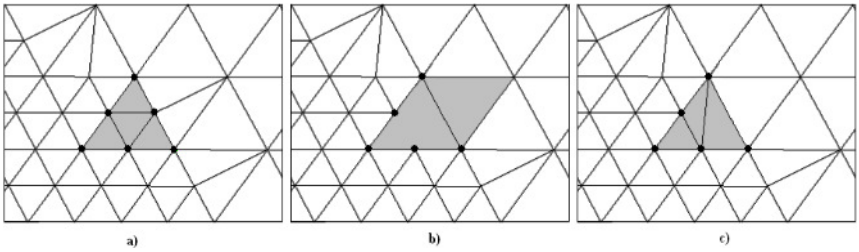

Figura 57: a) Tetraedro refinado regularmente al que se aplicará *coarsening*. b) Después de eliminar los subtetraedros se deberá desmarcar las aristas adyacentes a tetraedros irregulares, en consecuencia también estos se serán eliminados. c) Finalmente, se reclasifica la malla según el número de marcas que tiene para dejarla conforme.

En la *etapa de* ***Borrado***, se eliminan todos los subtetraedros regulares cuyos padres se requiere que pasen a una menor resolución. Adicionalmente, necesitamos desmarcar las aristas del padre que son adyacentes sólo a tetraedros irregulares; así, éstos podrán eliminarse posteriormente (Figura 57b). Desafortunadamente, al no guardar explícitamente la adyacencia de tetraedros por arista, no podemos saber directamente qué tetraedros irregulares son adyacentes. Nuestra solución es bastante simple. En primera instancia, desmarcamos todas las aristas de los padres y, debido a que los tetraedros comparten aristas, los adyacentes se verán modificados por esta acción. Posteriormente, sólo los adyacentes regulares restaurarán sus marcas y permanecerán intactos, mientras que los vecinos irregulares serán detectados y eliminados. El algoritmo que lleva a cabo este proceso se detalla en la Figura 58.

```
Proc   Borrado( listaPadresRegulares, Malla )
       tablaHash ← Vacia
       Para cada tetraedro t en listaPadresRegulares
              EliminarSubTetraedros( t )
              Para cada arista a en t
                     DesmarcarArista( a )
                     tablaHash ← a
              Fin Para
       Fin Para
       Para cada tetraedro terminal t en Malla
              Si t.padre no ha sido procesado antes
                     Si alguna arista de t.padre existe en tablaHash
                            Si t es regular
                                   MarcarArista( aristas de t.padre )
                            Si No
                                   EliminarSubTetraedros( t.padre )
                            Fin si
                     Fin Si
              Fin si
       Fin Para
Fin Proc
```

Figura 58: Pseudo código para la fase borrado de tetraedros.

El algoritmo de Coarsening utiliza los mismos algoritmos de clasificación y refinamiento que se definieron anteriormente para el refinamiento. El algoritmo completo se muestra en la Figura 59.

```
Proc   coarseRegulares( listaPadresRegulares, Malla )
       Borrado( listaPadresRegulares, Malla )

       ClasificaMalla( Malla )

       Para cada tetraedro terminal t en Malla
              Si t es de Tipo1, Tipo2, Tipo3a, Tipo3b o Tipo4
                     RefinarTetraedro( t, Malla )
              Fin Si
       Fin Para
Fin Proc
```

Figura 59: Algoritmo de coarsening.

6.2.5 Refinamiento Selectivo

El refinamiento selectivo, corazón del sistema de rendering progresivo, consiste en adaptar la resolución de la malla para mostrar los detalles que llegan progresivamente al cliente y son insertados en el Buffer de Detalles.

```
Proc    refinamientoSelectivoSimple( Malla )
        ListaRegulares ← Vacía
        ListaIrregulares ← Vacía
        Para cada tetraedro terminal t en Malla
                Si t es regular y están disponibles los detalles de t
                        ListaRegulares ← t
                Fin Si
                Si t es irregular y están disponibles los detalles de t.padre
                        ListaIrregulares ← t
                Fin Si
        Fin Para

        Si ListaIrregulares No esta Vacía
                refinarIrregulares( ListaIrregulares, Malla )
        Fin Si
        Si ListaRegulares No esta Vacía
                refinarRegulares( ListaRegulares, Malla )
        Fin Si
Fin Proc
```

Figura 60: Refinamiento Selectivo simple de la malla

Una primera aproximación consiste en adaptar la resolución de la malla en cada *time step* para mostrar la máxima cantidad de detalles. En este caso, la cantidad de detalles mostrados sólo está limitada por la cantidad de coeficientes de detalles que están presentes en el Buffer de Detalles. El algoritmo de Refinamiento Selectivo más simple posible que lleva a cabo esta tarea se muestra en la Figura 60. Este algoritmo refina sólo los tetraedros que encuentran sus coeficientes de detalle en el Buffer de Detalles. Conforme el Buffer de Detalles crece, la malla se adapta aumentando su resolución para mostrar los detalles. En la Figura 61 se aprecia el resultado de utilizar la técnica de la Figura 60. Sin embargo, con el aumento de tetraedros en la malla, el tiempo que toma realizar el rendering es cada vez mayor, reduciendo la interactividad de la visualización.

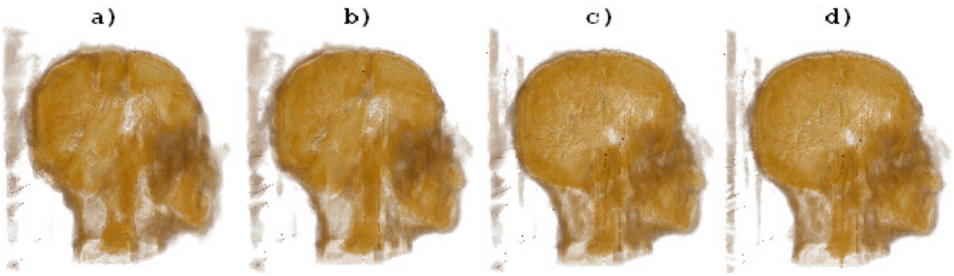

Figura 61: Rendering de Volúmenes del dataset Head256 a diferentes resoluciones según la cantidad de coeficientes de detalle presentes en el Buffer de Detalles. a) Buffer con 8000 coeficientes de detalle. b) Con 13000 coeficientes de detalle. c) Con 18000 coeficientes de detalle. c) Con 23000 coeficientes de detalle.

Para aliviar este problema, es posible refinar la malla con respecto al Punto de Visión (PV) para mostrar más detalle en las partes cercanas al PV y mostrar menos detalle en las partes alejadas del mismo. Este proceso, efectivamente, acotaría el número de tetraedros que se mostrarán en cada *time step,* aumentando la interactividad. Con este propósito, se incluyen dos heurísticas en el algoritmo de Refinamiento Selectivo, las cuales son ([Hoppe97]): *Frustum de Visión* y *Error Geométrico en el Espacio de la Pantalla*. Estas heurísticas determinarán si un tetraedro debe ser refinado, *coarsed* o ignorado. El efecto de usar estos criterios se puede observar en la Figura 62.

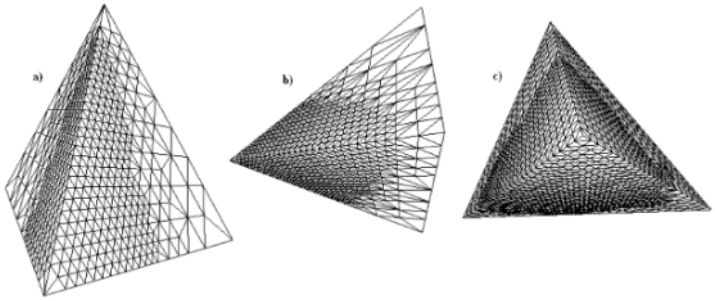

Figura 62: a) y b) Todos los tetraedros fuera de la pirámide de visión son *coarsed*. c) Refinamiento basado en error el geométrico en el espacio de la pantalla.

En el caso del *Frustum de Visión*, el objetivo es realizar el *coarsening* de todos los tetraedros que están fuera de la pirámide de visión. Al igual que Hoppe ([Hoppe97]), usamos esferas como volumen de encierro para aproximar un tetraedro. Si la esfera está completamente fuera de la pirámide de visión, el tetraedro es *coarsed*. En caso contrario es ignorado.

La heurística que toma en cuenta el *Error Geométrico en el Espacio de la Pantalla*, propone calcular el error tomando como base la proyección en perspectiva de la esfera que encierra al tetraedro. Según esta heurística, cuando la proyección en perspectiva ocupa más píxeles verticales que cierta tolerancia, el tetraedro debe ser refinado. Si por el contrario, la proyección ocupa un número de píxeles muy por debajo de la tolerancia, el tetraedro debe ser *coarsed*; en otro caso debe ser ignorado.

En el momento de la creación de cada tetraedro, calculamos el diámetro de la esfera que lo encierra. Posteriormente, usamos ese diámetro precalculado para encontrar la proyección de la esfera, así como para determinar si está dentro de la pirámide de visión. La Ecuación (43) calcula el número de píxeles verticales que proyecta la esfera sobre la pantalla, donde *diam* es el diámetro de la esfera, *yres* es la resolución vertical de la pantalla en píxeles, *dist* es la distancia euclidiana del centro de la esfera al punto de visión y *fov* es el campo de visión en radianes. En suma, esta expresión asegura que la esfera que contiene cada tetraedro tendrá una proyección vertical equivalente a *err* píxeles.

$$err(t) = \frac{yres}{2 \times \tan\left(\frac{fov}{2}\right)} \left(\frac{diam_t}{dist_t}\right) \tag{43}$$

Nuestro algoritmo funciona incrementalmente, es decir, dado un nuevo punto de visión, no es necesario refinar nuevamente toda la malla, sino que la podemos modificar gradualmente. Si asumimos que el punto de visión no experimenta cambios bruscos, es posible la adaptación de la malla en tiempos interactivos. Nuestro algoritmo procede en dos etapas: *Coarsening* y Refinamiento. El algoritmo que realiza el Refinamiento Selectivo, tomando en cuenta las heurísticas mencionadas anteriormente, se muestra en la Figura 63.

```
Proc    RefinamientoSelectivo( Malla )
        ListaCoarsening ← Vacia
        ListaRegulares ← Vacía
        ListaIrregulares ← Vacía
        Para cada tetraedro terminal t en Malla
                Si t.padre esta en listaCoarsening
                        Continuar Bucle Para
                Fin Si
                Si t es regular
                        Si t.padre esta fuera del Frustum
                                Si todos los hijos de t.padre son terminales
                                        ListaCoarsening ← t.padre
                                Fin Si
                                Continuar Bucle Para
                        Fin Si
                        Si t.padre.errorGeometrico < Tolerancia
                                Si todos los hijos de t.padre son terminales
                                        ListaCoarsening ← t.padre
                                Fin Si
                                Continuar Bucle Para
                        Fin Si
                Fin Si
                Si t esta fuera del Frustum
                        Continuar Bucle Para
                Fin Si
                Si t.errorGeometrico > Tolerancia
                        Si t es regular
                                Si están disponibles los detalles de t
                                        ListaRegulares ← t
                                Fin Si
                        Si No
                                Si están disponibles los detalles de t.padre
                                        ListaIrregulares ← t
                                Fin Si
                        Fin Si
                Fin Si
        Fin Para
        /*Etapa de Coarsening*/
        Si ListaCoarsening No esta Vacia
                coarseRegulares( ListaCoarsening, Malla )
        Fin Si
        /*Etapa de Refinamiento*/
        Si ListaIrregulares No esta Vacía
                refinarIrregulares( ListaIrregulares, Malla )
        Fin Si
        Si ListaRegulares No esta Vacía
                refinarRegulares( ListaRegulares, Malla )
        Fin Si
Fin Proc
```

Figura 63: Pseudo código para el refinamiento selectivo.

En la etapa de *Coarsening* nos encargamos de eliminar el sobre refinamiento que puede existir en la malla. Dado que una nueva posición del punto de visión introduce un nuevo error geométrico y posiblemente una nueva posición con respecto al *Frustum*, utilizamos ambos criterios para realizar el *coarsening* de la malla. Si un tetraedro está fuera del *Frustum* o su proyección está muy por debajo de la tolerancia, será pasado a una resolución menor.

En la etapa de *Refinamiento* utilizamos las dos heurísticas planteadas para decidir si un tetraedro debe ser refinado. Un tetraedro será refinado si y sólo si sus coeficientes de detalle se encuentran presentes en el Buffer de Detalles, está dentro del *Frustum* y su error geométrico es mayor que cierta tolerancia dada por el usuario. La tolerancia debe estar expresada en cantidad de píxeles.

Resulta particularmente difícil encontrar una medida exacta que indique cuándo debemos reducir el nivel de detalle, es decir, determinar cuándo existe sobre refinamiento. Una mala estimación puede resultar en que se realice frecuentemente el *coarsening* innecesario de tetraedros, haciendo más lenta la visualización. Además, el *coarsening* masivo de tetraedros puede ser notorio para el usuario durante una simulación interactiva. Nosotros adoptamos la siguiente heurística:

Si el error geométrico en el espacio de pantalla del ***padre*** *del tetraedro t es menor que la tolerancia, entonces el tetraedro t está sobre refinado y debe ser reemplazado por su padre.*

Esto se puede expresar formalmente en el siguiente predicado:

Sea Γ_i una malla de tetraedros de resolución i y sea t un tetraedro:

$$\forall t \in \Gamma_i \;\; Si\ err(t.padre) < tolerancia \Rightarrow coarse(t)$$

En la práctica, usando este simple predicado para estimar el sobre refinamiento, el *coarsening* aumenta (ocurre con más frecuencia) proporcionalmente con el aumento de la distancia con respecto al observador. Esto favorece la performance del Rendering de Volúmenes cuando la malla está alejada del observador. Cuando la malla está cerca, el *coarsening* prácticamente no tiene efecto alguno, por lo que la performance del Rendering de Volúmenes sólo depende del número de tetraedros que existen en la malla.

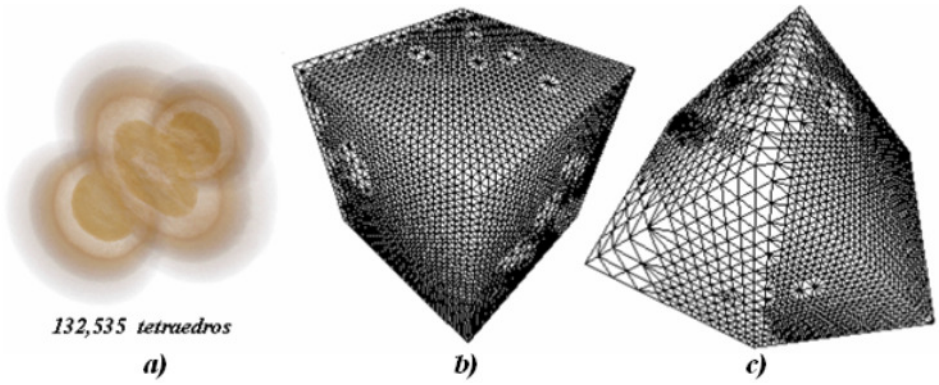

Figura 64: a) Rendering de Volúmenes del dataset HARMONIC aplicando el Error Geométrico en Espacio de la Pantalla como criterio para refinamiento selectivo de la malla. b) Rendering de la frontera de la malla de tetraedros de la imagen de la izquierda desde el punto de vista del observador. c) Rendering de la misma frontera vista de costado por una tercera persona.

En las Figuras 64 y 65, se muestra el resultado de aplicar el algoritmo de refinamiento selectivo usando sólo el Error Geométrico en el Espacio de la Pantalla como heurística de refinamiento. En este caso, se notan los defectos en el Rendering de Volúmenes a causa del refinamiento irregular que introduce la variación de resoluciones. Sin embargo, los defectos no son muy notorios ya que se producen en su mayoría en las zonas alejadas de la posición del observador.

Otra posible estimación del sobre refinamiento, similar a la anterior, es utilizar al abuelo de un tetraedro, en lugar de al padre, como referencia para calcular el Error Geometrico en el Espacio de la Pantalla. Esta estimación agrega mas tolerancia al sobre refinamiento, haciendo que el *coarsening* sea menos notorio visualmente y ayudando a que el refinamiento sea mas uniforme en las zonas cercanas al punto de visión.

Por otro lado, la heurística del Frustum de Visión tiene efecto cuando la malla está muy cerca del observador, de modo tal que sólo parte del volumen es visible. En este caso, los tetraedros que están fuera del Frustum son *coarsed*, reduciendo el número de tetraedros procesados en el Rendering. En la práctica, esta heurística también puede impactar negativamente la eficiencia del rendering si es que las porciones no visibles (que fueron *coarsed*) se hacen nuevamente visibles una y otra vez. Esto es a causa de que se deben refinar nuevamente porciones que recientemente fueron *coarsed* y que ahora son visibles; además, se suma el costo de un nuevo *coarsening* a otra parte de la malla, que ahora es no visible. En cambio, si la exploración se hace dentro de una zona definida, el Rendering de Volúmenes resulta favorecido gracias al menor número de tetraedros.

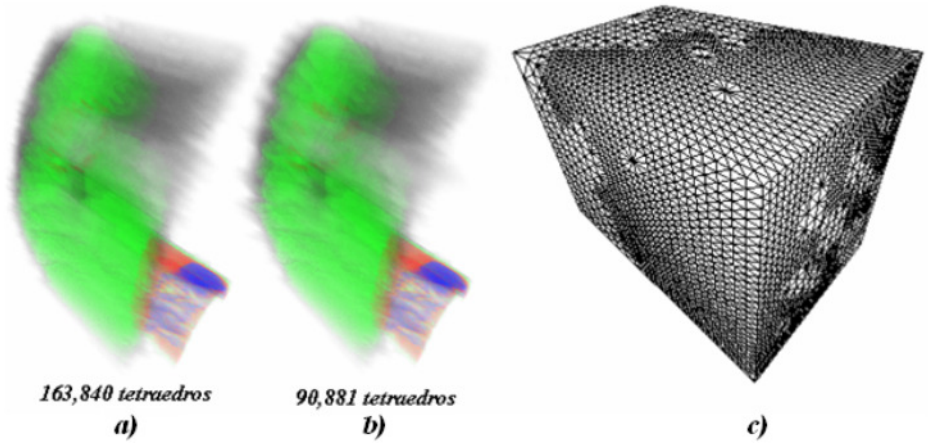

Figura 65: a) Rendering de Volúmenes del dataset SPX a máxima resolución. b) Rendering de Volúmenes del dataset SPX aplicando el Error Geométrico en Espacio de la Pantalla como criterio para refinamiento selectivo de la malla. c) Rendering de la frontera de la malla de tetraedros de la imagen b) desde el punto de vista del observador.

La Figura 66 muestra el resultado de aplicar el algoritmo de refinamiento selectivo usando tan sólo la heurística del Frustum de Visión para determinar el *coarsening* de la malla. Notamos que la calidad del Rendering de Volúmenes no se ve afectada en absoluto por esta heurística. Esto se debe a que la transición entre resoluciones es suave y esto se debe en parte a que usamos esferas como representación de tetraedros, y esta aproximación introduce un error en los

cálculos de visibilidad que nos sirve de tolerancia contribuyendo así a la suavidad de las transiciones.

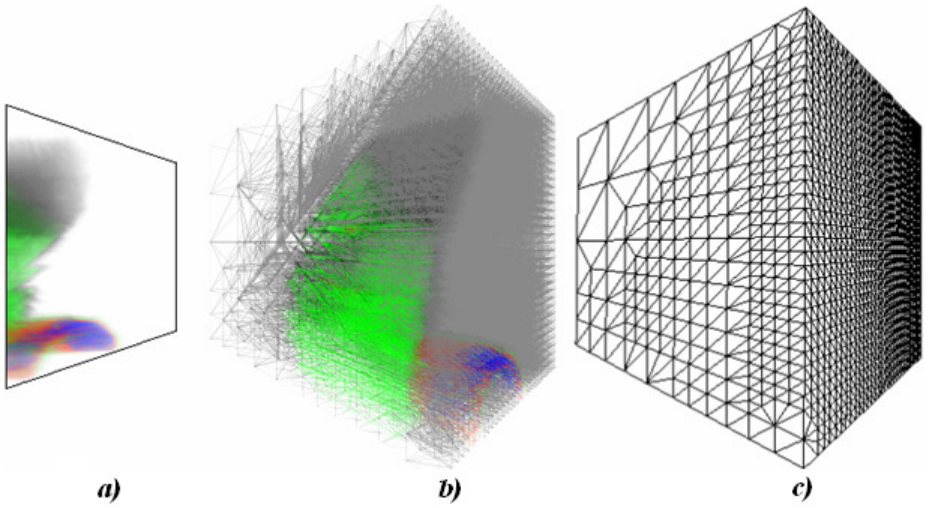

Figura 66: a) Rendering de Volúmenes del dataset SPX aplicando la heurística del *Frustum de Visión* como criterio para el *coarsening* de la malla. b) Rendering de malla de alambre del volumen completo visto por una tercera persona. c) Rendering de la frontera de la malla de tetraedros de la imagen b).

6.3 Obtención de Mallas con Conectividad de Subdivisión

Los Datasets usados en nuestras pruebas son remuestreos de los Datasets originales. Dado que los Datasets usualmente no poseen la propiedad de conectividad de subdivisión, la cual es necesaria para calcular la transformada wavelet sobre redes tetraédricas, éstos debieron ser generados. Así es que los Datasets usados en nuestras pruebas fueron obtenidos mediante el remuestreo de los Datasets originales.

Mediante el muestreo mapeamos la red original a una red que sí posee dicha propiedad. Debemos tener en cuenta que el muestreo no siempre capta todos los detalles del dataset original y que las mallas muestreadas poseen más tetraedros que las mallas originales. Si la resolución de la malla destino (malla con la propiedad de conectividad de subdivisión) es menor que la de la malla de origen (malla original), el muestreo implica una pérdida de detalle. En la práctica, es probable que las partes poco detalladas del volumen sean muestreadas sin pérdida de detalle, mientras que las partes muy detalladas sean muestreadas con pérdida de detalle. Por esta razón, no podemos tomar a los datasets originales como referencia en términos de calidad ni de performance de rendering.

A continuación describimos el proceso de remuestreado:

Primero calculamos el AABB (Axis Aligned Bounding Box) de la malla de origen y lo subdividimos en cinco tetraedros como se muestra en la Figura 67. En el algoritmo de la Figura 68 se detalla el proceso de subdivisión.

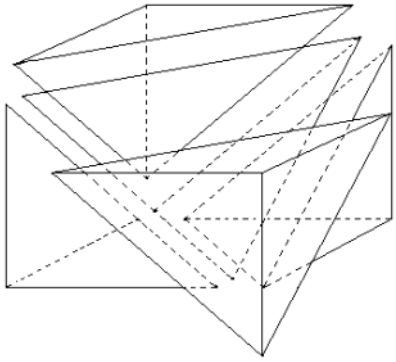

Figura 67: Subdivisión de un AABB en cinco tetraedros.

```
Funcion  Divide_AABB_En_Tetraedros( min, max ) : Malla
        Malla ← Vacia
        Malla.Vertices[0] ← vector( min.x, max.y, min.z )
        Malla.Vertices[1] ← vector( max.x, max.y, min.z )
        Malla.Vertices[2] ← vector( max.x, min.y, min.z )
        Malla.Vertices[3] ← vector( min.x, min.y, min.z )
        Malla.Vertices[4] ← vector( min.x, max.y, max.z )
        Malla.Vertices[5] ← vector( max.x, max.y, max.z )
        Malla.Vertices[6] ← vector( max.x, min.y, max.z )
        Malla.Vertices[7] ← vector( min.x, min.y, max.z )
        Malla.Tetraedros[0] ← Tetra( 0, 1, 5, 2 )
        Malla.Tetraedros[1] ← Tetra( 0, 5, 4, 7 )
        Malla.Tetraedros[2] ← Tetra( 0, 5, 7, 2 )
        Malla.Tetraedros[3] ← Tetra( 0, 7, 3, 2 )
        Malla.Tetraedros[4] ← Tetra( 5, 6, 7, 2 )
        retorna Malla
Fin Funcion
```

Figura 68: Algoritmo de subdivisión del AABB en cinco tetraedros

Los cinco tetraedros forman la malla base de una multirresolución, la cual es, de hecho, la malla de destino. Refinamos uniformemente la malla base usando la subdivisión regular de Bey ([Bey95]) hasta alcanzar una resolución similar a la de la malla de origen. El algoritmo de refinamiento uniforme se puede apreciar en la Figura 69. El diámetro de la esfera que encierra al tetraedro más pequeño de la malla de origen es tomado como referencia para calcular cuántos niveles de subdivisión son necesarios para la malla de destino. Aproximamos el valor de cada escalar asociado a los vértices de la malla destino interpolando los escalares de los tetraedros de la malla de origen que los contienen. Utilizamos un Octree para encontrar los tetraedros próximos a un vértice y luego determinamos cuál contiene al vértice. Un tetraedro contiene un vértice, si y sólo si, las coordenadas baricéntricas del vértice en el tetraedro son todas positivas. El algoritmo de remuestreo se muestra en la Figura 70.

```
Proc    RefinamientoUniforme( Malla, nivel )
        n ← 1
        Mientras n ≤ nivel
                Para cada tetraedro terminal t en Malla
                        MarcarTetraedro( t )
                        t ← Tipo1
                Fin Para
                Para cada tetraedro terminal t en Malla
                        RefinarTetraedro( t, Malla )
                Fin Para
                n ← n + 1
        Fin Mientras
Fin Proc
```

Figura 69: Algoritmo de Refinamiento Uniforme

```
Funcion Remuestreo( MallaOrigen, nivel ) : Malla Semi-Regular
    aabb ← Calcula_AABB( MallaOrigen )

    MallaDestino ← Divide_AABB_En_Tetraedros( aabb.min, aabb.max )

    RefinamientoUniforme( MallaDestino, nivel )

    octree ← CrearOctree( MallaOrigen )

    Para cada tetraedro terminal tDestino en MallaDestino
        Para cada vertice v de tDestino
            Si el escalar asociado a v aún no ha sido muestreado

                nodo_Octree ← octree.Contiene_a_vertice( v )

                Para cada tetraedro tOrigen en nodo_Octree
                    baricentricas ← tOrigen.CoordenadasBaricentricas( v )
                    Si tOrigen contiene a v
                        Muestra ← baricentricas * Escalares de tOrigen
                        Escalar asociado a v ← Muestra
                        Salir del bucle Para
                    Fin si
                Fin Para

            Fin Si
        Fin Para
    Fin Para
    retorna MallaDestino
Fin Funcion
```

Figura 70: Pseudo código del algoritmo de remuestreo

En el algoritmo de remuestreo, cada nodo terminal del Octree encierra espacialmente a un conjunto de tetraedros de la malla de origen. Al mismo tiempo, podemos asociar a cada vértice de la malla destino con un nodo terminal del Octree. De este modo, a cada vértice de la malla destino le corresponde un cluster de tetraedros, de los cuales sólo uno contiene al vértice. Es posible acotar el espacio de búsqueda dentro del cluster si consideramos como candidatos sólo a los tetraedros cuyo AABB contiene al vértice. En consecuencia, el número de tetraedros sobre los cuales se realiza el cálculo de coordenadas baricéntricas es conservativo.

Después del remuestreo tenemos una red tetraédrica *semi-regular* que cumple con la propiedad de conectividad de subdivisión. Aplicamos el análisis multirresolución basado en wavelets descripto en el capítulo 5 sección 5.3.4 ([Castro97], [Boscardin01], [Castro05], [Castro06]) y obtenemos como resultado la malla base y un archivo con los coeficientes de detalle indexados por su identificador único id_wavelet.

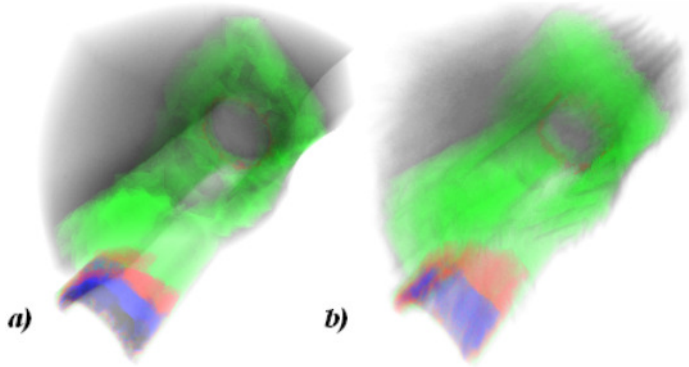

Figura 71: Rendering de Volúmenes del dataset SPX. a) Malla original. b) Malla muestreada.

En la Figura 71 comparamos el Rendering de Volúmenes del dataset SPX original (izquierda) versus la versión muestreada (derecha). La malla original posee 12,936 tetraedros, mientras que la malla muestreada tiene 163,840 tetraedros. A pesar de la enorme diferencia en el número de tetraedros que posee la malla muestreada, no logra captar todos los detalles de la malla original. Esto se debe a que el dataset SPX es de topología no estructurada y posee zonas curvilíneas cóncavas y convexas con tetraedros de diferentes tamaños y formas. Esto lo hace particularmente difícil de muestrear regularmente.

6.4 Resultados

Mostramos el desempeño de los algoritmos de Refinamiento Selectivo a través de la visualización de distintos conjuntos de datos. En el transcurso de cada visualización, el buffer de detalles se llena progresivamente a razón de 1000 coeficientes añadidos en cada *time step*, hasta que se completan todos los detalles o se alcanza un número significativo.

Adicionalmente, agregamos rotación a los volúmenes para obtener distintas vistas de los mismos mientras transcurre el refinamiento. Estos rotan, alrededor del eje vertical, un ángulo de 0.05 radianes en cada *time step*. Es posible realizar interacciones directamente sobre el volumen en tiempos interactivos siempre y cuando se gradúen adecuadamente algunos parámetros como la tolerancia de refinamiento y la distancia de visualización. Si el refinamiento toma un tiempo demasiado largo, las interacciones también pueden realizarse sobre un LOD uniforme mientras que el refinamiento continúa realizándose en segundo plano.

El Cuadro 1 muestra las características de los conjuntos de datos utilizados en nuestras pruebas. Todos son remuestreos de los datasets originales, es por esto que difieren en sus dimensiones. Nosotros analizamos los resultados obtenidos en la visualización de los datasets Harmonic y Head256. El dataset Harmonic, de tamaño medio, posee cerca de 200 mil tetraedros, mientras que el dataset Head256 posee cerca de 1.5 millones de tetraedros.

Nombre	Tetraedros	Tetraedros Malla Base	Coeficientes de Detalle	Dimensiones
SPX	187,245	5	23,405	6 x 7 x 9.3
HARMONIC	187,245	5	23,405	30 x 30 x 30
HEAD256	1,497,965	5	187,245	222 x 222 x 210
ENGINE	1,497,965	5	187,245	255 x 255 x 109

Cuadro 1: Características de los datasets utilizados en nuestras pruebas

En el Cuadro 2 se aprecia el resumen de datos extraídos después de realizar una simulación del rendering progresivo del dataset Harmonic durante 312 *time steps*, lo cual equivale a aplicar una rotación total de 5π radianes al volumen. Este número de *time steps* es más que suficiente para completar el llenado progresivo del Buffer de Detalles.

Definimos distintos umbrales de tolerancia, expresados como el porcentaje de píxeles con respecto de la resolución vertical de la pantalla, para probar el algoritmo de Refinamiento Selectivo que usa heurísticas. La distancia de visualización en este caso es de 30 unidades y se deriva de las dimensiones del volumen. Se puede notar que la cantidad promedio de símplices generados aumenta conforme la tolerancia indicada disminuye. Lo mismo sucede con la cantidad de memoria promedio. El tiempo promedio de refinamiento es suficientemente bajo para realizar la visualización interactiva a 8.5 FPS (Cuadros Por Segundo) utilizando una tolerancia del 20%. En este caso, el número de tetraedros promedio es sólo el 10% del número de tetraedros totales. Conforme aumenta la precisión de la visualización el tiempo de refinamiento aumenta y la visualización se realiza en última instancia a 1.2 FPS en promedio.

Refinamiento Selectivo							
HARMONIC	**Cantidad promedio de símplices generados**			**Memoria**	**Tiempo Promedio (seg.)**		
Tolerancia	**#Tetraedros**	**#Vértices**	**#Aristas**	**MB**	**Refinam.**	**Rendering**	**FPS**
20%	19383	3757	24753	6.39	0.026	0.079	8.502
15%	30913	6474	38399	7.42	0.037	0.112	6.108
12.5%	105940	24130	128792	14.15	0.235	0.288	1.836
10%	161683	28213	197454	18.99	0.399	0.411	1.193
6.25%	177744	29083	218458	20.15	0.296	0.510	1.205
Detalles	**Refinamiento Selectivo Simple**						
23,405	182819	29759	225449	20.56	0.183	0.529	1.365

Cuadro 2: Resumen de los datos extraídos de las visualizaciones del dataset Harmonic

El Cuadro 3 muestra el resumen de datos de la misma simulación anterior pero aplicada al dataset Head256. La distancia de visualización en este caso es de 269 unidades. Los resultados son consistentes con los obtenidos para el dataset Harmonic a excepción del caso en que la tolerancia es de 6.25%. En este caso el número de tetraedros promedio supera los 700 mil por lo que no es posible visualizarlo en tiempos interactivos.

Refinamiento Selectivo							
HEAD256	**Cantidad promedio de símplices generados**			**Memoria**	**Tiempo Promedio (seg.)**		
Tolerancia	**#Tetraedros**	**#Vértices**	**#Aristas**	**MB**	**Refinam.**	**Rendering**	**FPS**
20%	17286	3649	21938	10.57	0.021	0.069	9.732
15%	26392	5284	33072	11.38	0.034	0.103	6.481
12.5%	106112	24594	128888	18.55	0.260	0.287	1.718
10%	153945	28252	187605	22.69	0.421	0.396	1.158
6.25%	784857	154413	932437	78.62	8.297	1.903	0.097
Detalles	**Refinamiento Selectivo Simple**						
23,405	182819	29759	225449	24.93	0.183	0.530	1.364

Cuadro 3: Resumen de los datos extraídos de las visualizaciones del dataset Head256

Probamos el desempeño del algoritmo de Refinamiento Selectivo Simple utilizando los datasets antes mencionados pero acotando el número máximo de detalles mostrados a 23,405. Los resúmenes de datos se muestran en la parte de abajo de los Cuadros 3 y 4 para los datasets Harmonic y Head256 respectivamente. La visualización, en tiempos interactivos, de todos los datasets es posible acotando adecuadamente el número de detalles a mostrar.

En las Figuras 72, 73, 74 y 75 se pueden apreciar las graficas del tiempo de refinamiento y de rendering de los algoritmos de Refinamiento Selectivo mientras se llena progresivamente el Buffer de Detalles. Se muestra como referencia, en gris, la cantidad de tetraedros terminales generados por el refinamiento selectivo y, en gris punteado, la cantidad de detalles que contiene el Buffer de Detalles. Ambas están escaladas a los intervalos de tiempo correspondientes para poder ser comparados fácilmente.

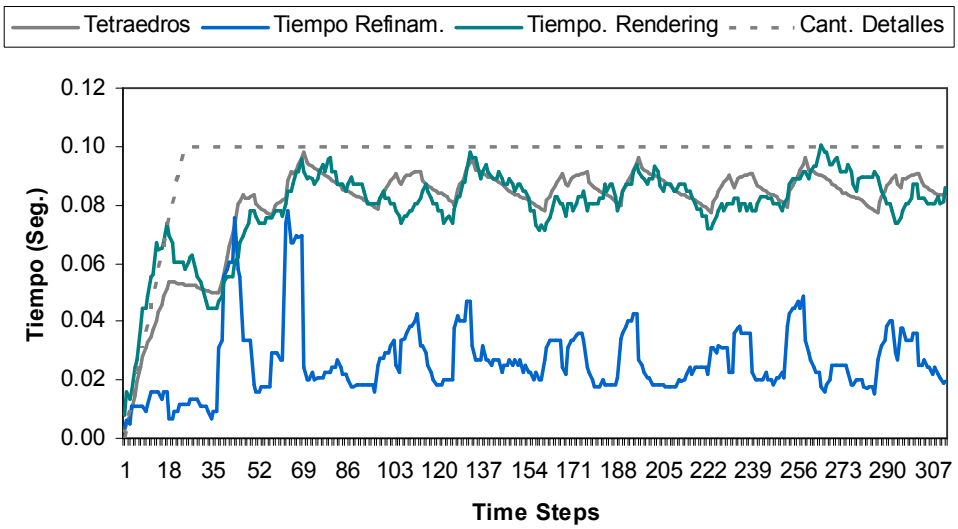

Figura 72: Gráfico del tiempo que toma el Rendering Progresivo del dataset Harmonic con 20% de tolerancia

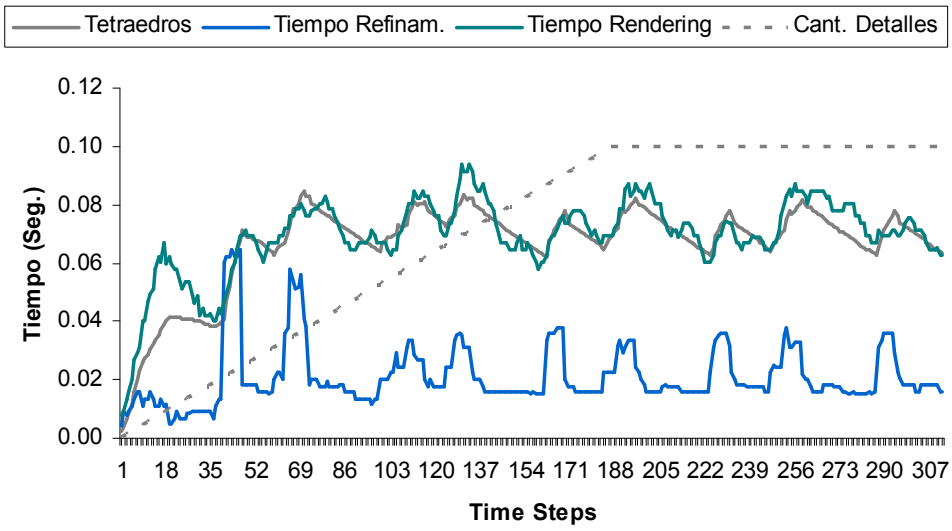

Figura 73: Grafico del tiempo que toma el Rendering Progresivo del dataset Head256 con 20% de tolerancia

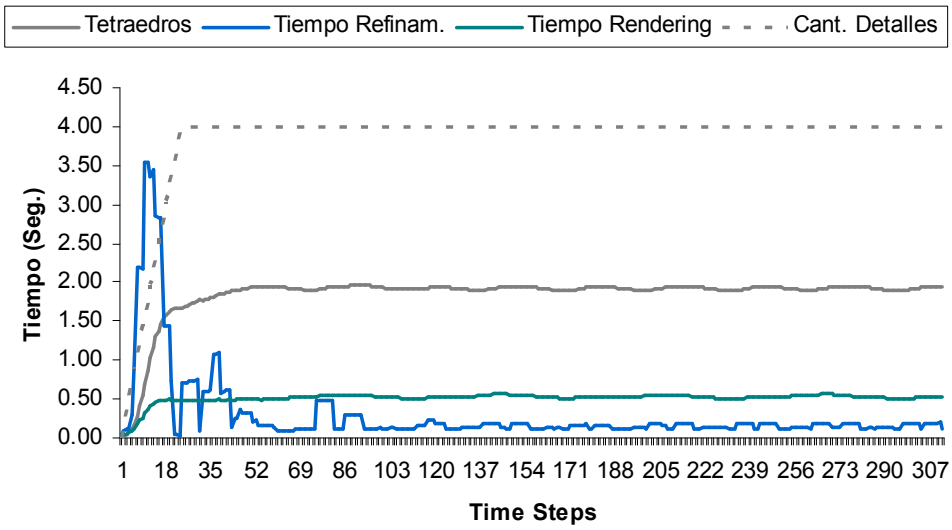

Figura 74: Grafico del tiempo que toma el Rendering Progresivo del dataset Harmonic con 6.25% de tolerancia

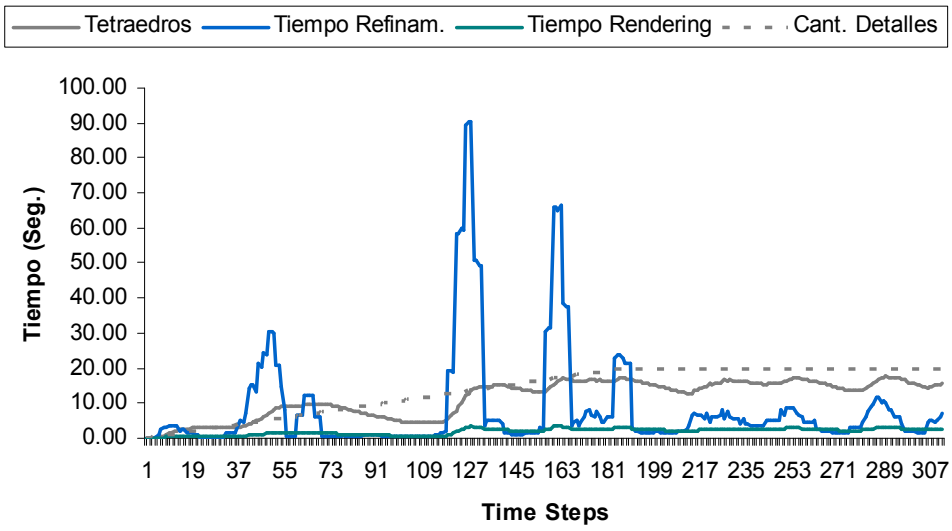

Figura 75: Grafico del tiempo que toma el Rendering Progresivo del dataset Head256 con 6.25% de tolerancia

Notamos que en todas las gráficas el tiempo de refinamiento es mayor durante el llenado progresivo del Buffer de Detalles. Luego decrece y mantiene un patrón constante. El caso más dramático se encuentra en la Figura 75, en la cual el tiempo de refinamiento alcanza un pico de 100 segundos. Esto hace que la visualización sea no interactiva durante esos picos.

Las Figuras 76 y 77 muestran la grafica del tiempo que toma el refinamiento selectivo usando el algoritmo de Refinamiento Selectivo Simple. En estas gráficas, muy similares debido a la cantidad de detalles mostrados, se ve claramente que el tiempo que toma el refinamiento durante el llenado del Buffer de Detalles crece constantemente hasta un pico de 5 segundos. Esto hace que la visualización sea no interactiva durante el llenado del Buffer de Detalles.

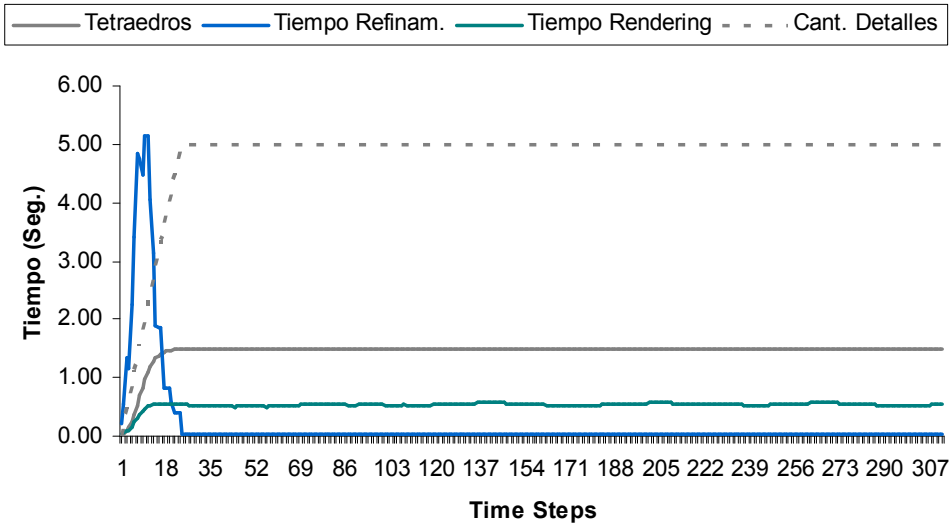

Figura 76: Grafico del tiempo que toma el Rendering Progresivo del dataset Harmonic con el algoritmo de refinamiento simple y un tope de 23,405 detalles

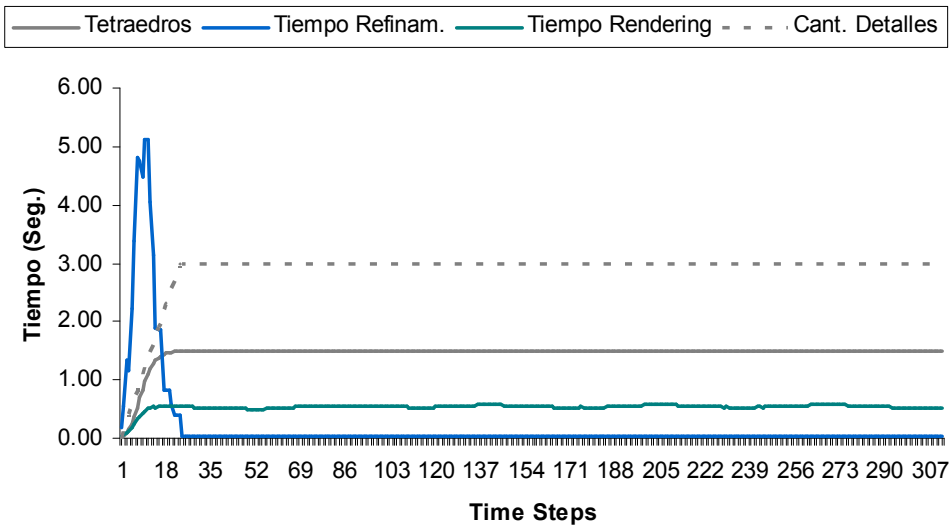

Figura 77: Grafico del tiempo que toma el Rendering Progresivo del dataset Head256 con el algoritmo de refinamiento simple y un tope de 23,405 detalles

Debido a esta pérdida de interactividad durante intervalos de varios segundos durante el llenado progresivo del Buffer de Detalles, es recomendable que las técnicas de refinamiento se ejecuten como Threads independientes en segundo plano. De este modo, las interacciones se pueden realizar sobre los LODs extraídos en cada *time step*, es decir, extraídos en el momento en que se sincroniza el Thread de refinamiento con el Thread principal. La interactividad con los LODs sólo depende del número de tetraedros que éstos poseen.

En las Figuras 78 y 79 se muestra el Rendering de Volúmenes de los datasets Harmonic y Head256 respectivamente, utilizando los algoritmos de Refinamiento Selectivo presentados.

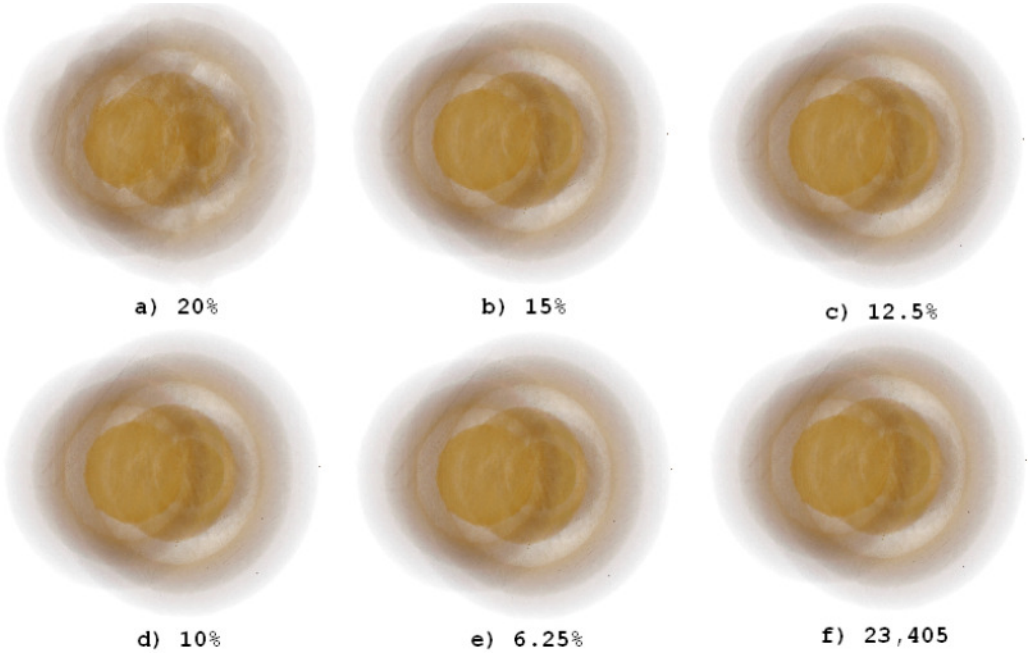

Figura 78: Rendering de Volúmenes del dataset Harmonic utilizando el Refinamiento Selectivo basado en heurísticas con distintos valores para la tolerancia. En f) se muestra el Rendering de Volúmenes utilizando el Refinamiento Selectivo Simple sólo con 23,405 detalles.

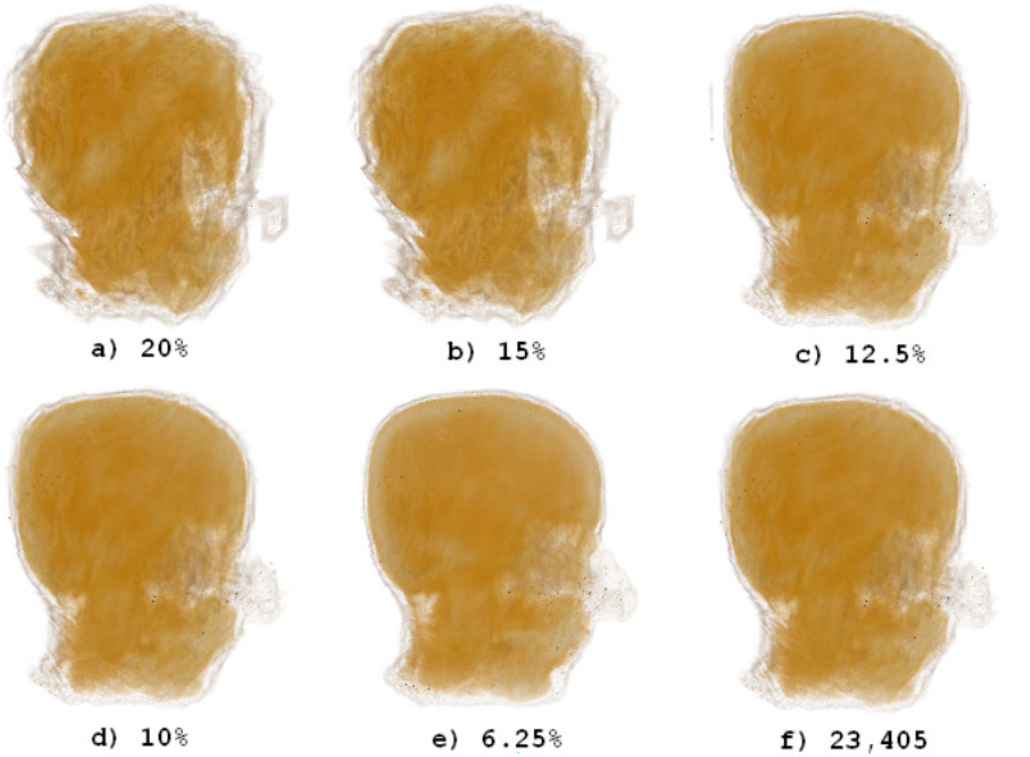

Figura 79: Rendering de Volúmenes del dataset Head256 utilizando el Refinamiento Selectivo basado en heurísticas con distintos valores para la tolerancia. En f) se muestra el Rendering de Volúmenes utilizando el Refinamiento Selectivo Simple sólo con 23,405 detalles.

CAPÍTULO VII

CONCLUSIONES Y TRABAJO FUTURO

En esta tesis se presentó una arquitectura de visualización progresiva Cliente – Servidor capaz de realizar el rendering progresivo de redes tetraédricas en el dominio wavelet cuya transmisión por la red se realiza de manera progresiva, por paquetes y en orden arbitrario. Se implementó completamente el pipeline de Rendering Progresivo y al hacerlo se generaron varias contribuciones.

Se presentó un conjunto de algoritmos que habilitan el Rendering Progresivo de volúmenes mediante el Refinamiento Selectivo y la aplicación de la Transformada Wavelet. Se presentaron dos algoritmos de Refinamiento Selectivo, uno simple y otro basado en heurísticas dependientes del punto de visión y por lo tanto variantes en el tiempo. Estos algoritmos requieren la definición de dos algoritmos básicos: un algoritmo de Refinamiento y uno de Coarsening. Ambos pueden ser aplicados también en la generación de mallas de Elemento Finito.

Asimismo, se presentó el diseño de las estructuras de datos necesarias para realizar las tareas de Refinamiento y Coarsening y se describió el funcionamiento de los índices propuestos. Estas estructuras de datos son compactas y no requieren almacenar explícitamente ningún tipo de adyacencia de tetraedros.

Se presentó un algoritmo para realizar el análisis multirresolución basado en wavelets así como un algoritmo para realizar el remuestreo de volúmenes irregulares hacia volúmenes que poseen la propiedad de conectividad de subdivisión. Estos procesos producen una malla base y un conjunto de detalles que pueden transmitirse progresivamente por la red.

Realizamos experimentos para probar el desempeño de nuestros algoritmos. Utilizamos varios datasets generados con nuestros algoritmos de remuestreo y análisis multirresolución. En las pruebas realizamos el Rendering Progresivo de Volúmenes utilizando tanto el algoritmo de Refinamiento Selectivo Simple como el algoritmo de Refinamiento Selectivo basado en heurísticas.

Nuestros algoritmos de Refinamiento Selectivo son simples, robustos y no requieren mantener explícitamente la adyacencia de tetraedros. El algoritmo basado en heurísticas habilita el Rendering Progresivo de volúmenes en tiempos interactivos graduando adecuadamente la tolerancia de error y la distancia de visualización. No obstante, las visualizaciones obtenidas no siempre son de alta calidad ya que la aplicación de heurísticas implica la pérdida de detalle en aras de mantener acotado el número de tetraedros que se renderizarán.

La pérdida de la interactividad puede producirse durante decenas de segundos al momento de refinar masivamente el volumen. Esto puede suceder en cualquier instante, inclusive durante el llenado progresivo del Buffer de Detalles. Sin embargo es posible extraer LODs seleccionando directamente de los tetraedros terminales de la jerarquía, lo cual posibilita la realización de interacciones sobre ellos siempre y cuando el algoritmo de refinamiento se ejecute en un Thread en segundo plano.

Nuestro trabajo futuro incluye el desarrollo de algoritmos de Remeshing para volúmenes no estructurados con el objetivo de obtener redes tetraédricas de mejor calidad y que además tengan la propiedad de conectividad de subdivisión. También nos centraremos sobre el diseño de algoritmos que trabajen en memoria secundaria para realizar el refinamiento y coarsening de redes tetraédricas que exceden los recursos de memoria locales. Asimismo consideraremos la extensión de nuestros algoritmos para trabajar con datos comprimidos.

BIBLIOGRAFÍA

[Bajaj97] C.Bajaj, V.Pascucci, and D.Schikore, *"The Contour Spectrum"*, in Visualization `97, pp. 167{173, IEEE, 1997.

[Bank83] R.E.Bank, A.H. Sherman, and A. Weiser, *"Refinement algorithms and data structures for regular local mesh refinement"*, in Scientific Computing, R. Stepleman, ed., Amsterdam IMACS/North Holland, 1983, pp 3-17.

[Bernardon04] Bernardon Fabio, Pagot Christian, Comba Joao, Silva Claudio, *"GPU-based Tiled Ray Casting using Depth Peeling"*, Technical Report of Scientific Computing and Imaging Institute University of Utah, July 2004.

[Bey95] Bey, Jürgen. *"Tetrahedral Grid Refinement"*, Computing, 55(4):355—378 (1995).

[Borgo04] Borgo Rita, Cignoni Paolo, Scopigno Roberto, *"Simplicial-based multiresolution volume datasets management: an overview"*, in Geometric Modeling for Scientific Visualization, Springer-Verlag, Heidelberg, 2004.

[Boscardin01] Boscardín, Liliana. *"Wavelets Definidas Sobre Volúmenes"*. MS thesis, Universidad Nacional del Sur, 2001.

[Cabral94] Cabral B, Cam N, Foran J, *"Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware"*, In Proc. of IEEE Symposium on Volume Visualization, pages 91–98, 1994.

[Callahan05] Callahan Steven, Ikits Milan, Comba Joao, Silva Claudio,*"Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering"*, *In* IEEE Transactions on Visualization and Computer Graphics, Vol. 11, Nº. 3, 2005.

[Callahan06] Steven P. Callahan, Louis Bavoil, Valerio Pascucci, Claudio T. Silva, *"Progressive Volume Rendering of Large Unstructured Grids"*, IEEE Transactions On Visualization And Computer Graphics, Vol. 12, Nº 5, October 2006.

[Castro05] Castro Silvia M., *"Compresión de Volúmenes"*, Disertación de Tesis de Doctorado, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Argentina, 2005.

[Castro06] Castro S., Castro L., Boscardín L., De Giusti A., *"Multiresolution Wavelet Based Model for Large Irregular*

Volume Data Sets", in Proceedings of WSCG'2006, pp 101–108, 8, 2006.

[Castro97] Castro, Silvia. *"Modelamiento Y Rendering de Volúmenes Mediante la Transformada Wavelet"*. MS thesis, Universidad Nacional del Sur, 1997.

[Cignoni00] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, *"Simplification of tetrahedral volume with accurate error evaluation"*, Proceedings IEEE Visualization'00, IEEE Press, 2000, pp. 85–92.

[Cignoni03] P. Cignoni, Paola Magillo, Leila De Floriani, Enrico Puppo, and R.Scopigno, *"Memory-efficient selective refinement on unstructured tetrahedral meshes for volume visualization"*, IEEE TVCG 9 (2003)

[Cignoni94] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno, *"Multiresolution modeling and rendering of volume data based on simplicial complexes"*, Proc.of 1994 Symp. on Volume Visualization, October 17-18 1994, pp. 19–26.

[Crawfis93] Crawfis, R., Max, N., *"Texture Splats for 3D Scalar and Vector Field Visualization"*, IEEE Visualization'93 Proceedings, October, 1993, 261-267.

[Crawfis95] Crawfis Roger, *"New Techniques for the Scientific Visualization of Three-Dimensional Multi-variate and Vector Fields"*, PhD thesis, University of California Davis, 1995.

[Debrin88] Debrin Robert, Carpenter Loren, Hanrahan Pat, *"Volume Rendering"*, ACM Computer Graphics, Volume 22, Number 4, August 1988.

[DeFloriani97] L. De Floriani, E. Puppo, and P. Magillo, *"A formal approach to multiresolution modeling"*, Theory and Practice of Geometric Modeling, Springer-Velrag, 1997.

[Engel01] Engel K, Kraus M, Ertl T, *"High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading"*. In Proc. Graphics Hardware, 2001.

[Engel02] Engel K, Ertl T, *"Interactive high-quality volume rendering with flexible consumer graphics hardware"*, EUROGRAPHICS 2002 (2002), 3.

[Farias01] Farias Ricardo, *"Efficient Rendering Of Volumetric Irregular Grids Data"*, PhD. Thesis Dissertation, State University Of New York at Stony Brook, 2001.

[Fisher03] R. Fisher, S. Perkins, A. Walker, E. Wolfart, *"Digital Filters"*, http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm, 2003

[Garland97] M. Garland, P.S. Heckbert, *"Surface Simplification Using Quadric Error Metrics"*, Proc. SIGGRAPH 1997, pp. 209-216, Aug. 1997.

[Garrity90] Garrity Michael, *"Raytracing Irregular Volume Data"*. In *Computer Graphics Proceedings of the San Diego Workshop on Volume Visualization*, volume 24, pages 35-40, November 1990.

[González-Yuste03] J.M. González-Yuste, R. Montenegro, J.M. Escobar, G. Montero and E. Rodríguez, *"Local Refinement Triangulations Using Object-Oriented Methods"*, Advances in Engineering Software, 2003, in press.

[Grosso97] R. Grosso, C. Lurig, and T. Ertl, *"The multilevel finite element method for adaptive mesh optimization and visualization of volume data"*, IEEE Visualization ´97, pp. 387–394.

[Haber90] Haber, R.B. and McNabb, D.A. 1990. *"Visualization Idioms: A Conceptual Model for Scientific Visualization Systems"*. In: Visualization In Scientific Computing, Shriver, B., Neilson, G.M., and Rosenblum, L.J., Eds., IEEE Computer Society Press, 74-93.

[Hadwiger01] Hadwiger Markus, Theußl Thomas, Hauser Helwig, Gröller Eduard, *"Hardware-Accelerated High-Quality Reconstruction of Volumetric Data on PC Graphics Hardware"*, Proceedings of Vision, Modeling, and Visualization, 2001.

[He96] T. He, L. Hong, A. Kaufman, and H. P_ster, *"Generation of transfer functions with stochastic search techniques"*, in Visualization `96, pp. 227{234, 489, IEEE, 1996.

[Hoppe96] H. Hoppe, *"Progressive meshes"*, Proceedings of SIGGRAPH '96 (1996), 99–108.

[Hoppe97] H. Hoppe, *"View-dependent refinement of progressive meshes"*, Computer Graphics (SIGGRAPH'93 Proceedings), 1997.

[IRIS] IRIS Explorer: http://www.nag.co.uk/welcome_iec.asp

[Kainz06] Kainz F, Bogart R, *"Technical Introduction to OpenEXR"*, Industrial Light & Magic technical product report, 2006, www.openexr.com

[Kindlmann98] G. Kindlmann and J. W. Durkin, *"Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering"*, in IEEE Symposium on Volume Visualization, pp. 79{86, 1998.

[Kniss02] Kniss Joe, Engel Klaus, Hadwiger Markus, Rezk-Salama Christof, *"High-Quality Volume Graphics on Consumer PC Hardware"*, SIGGRAPH 2002 San Antonio Texas, Course Notes 42, 2002.

[Kolb] Andreas Kolb. *"Visualization over the Internet"*. Institute of Computer Graphics, Vienna University of Technology, Austria.

[Krueger03] Krueger J., Westermann R., *"Acceleration Techniques for GPU-based Volume Rendering"*. In Proceedings IEEE Visualization 2003, 2003.

[Kumar96] S. Kumar, D. Manocha, W. Garrett, and M. Lin. *"Hierarchical back-face computation"*. In Proceedings of Eurographics Workshop on Rendering techniques, pages 235–ff., 1996.

[Kwok92] Kwok Bernard, *"Analysis of Radiosity Techniques in Computer Graphics"*, Masters Thesis, Department of Computer Science - York University, May 1992.

[Lacroute95] Lacroute Philippe, *"Fast Volume Rendering Using a Shear-Warp Factorization Of the Viewing Transformation"*, PhD Thesis, Stanford University, September 1995

[LaMar99] LaMar, Eric C., Bernd Hamann, and Kenneth I. Joy. *"Multiresolution Techniques for Interactive Texture-based Volume Visualization",* Proceedings of IEEE Visualization '99. pp. 355-362.

[Leven02] Leven Joshua, Corso Jason, Cohen Jonathan, Kumar Subodh*, "Interactive Visualization of Unstructured Grids Using Hierarchical 3D Textures"*, In Proceedings of IEEE Symposium on Volume Visualization and Graphics, pages 37–44, 2002.

[Levoy88] Levoy, M., *"Display of Surfaces from Volume Data"*, IEEE Computer Graphics and Applications, Vol. 8, No. 3, May, 1988, pp. 29-37.

[Levoy90] Levoy Marc, *"Efficient Ray Tracing of Volume Data"*, ACM Transactions on Graphics, 9(3):245-261, July 1990.

[Li03] Li Wei, Mueller Klaus, Kaufman Arie, *"Empty Space Skipping and Occlusion Clipping for Texture-based Volume Rendering"*, Technical Report, Center for Visual Computing (CVC) and Department of Computer Science Stony Brook University, Stony Brook, NY 11794-4400, 2003.

[Lohner92] Lohner R, Baum JD. *"Adaptive h-refinement on 3D unstructured grids for transient problems"*. Int J Numer Meth Fluids 1992;14:1407-19.

[Marks97] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. P_ster, W. Ruml, K. Ryall, J. Seims, and S. Shieber, *"Design galleries: a general approach to setting parameters for computer graphics and animation"*, in ACM SIGGRAPH Conference on Computer Graphics, pp. 389{400, 1997.

[Martig03] Martig S, Castro S, Fillottrani P, Estévez E.: *"Un Modelo Unificado de Visualización"*, Proceedings 8vo.Congreso Argentino de Ciencias de la Computación. La Plata, Argentina. Octubre, 2003.

[Max00] Max Nelson, Williams Peter, Silva Claudio, *"Approximate Volume Rendering for Curvilinear and Unstructured Grids by Hardware-assisted Polyhedron Projection"*, Technical Report of Lawrence Livermore National Laboratory and AT&T Laboratory-Research, 2000.

[Max03] Max Nelson, Williams Peter, Silva Claudio, Cook Richard, *"Volume Rendering for Curvilinear and Unstructured Grids"*, Proceedings of ACM SIGGRAPH 2000*,* Vol. 11, 53–61

[Max95] Max Nelson, *"Optical Methods for Direct Volumen Rendering"*, IEEE Transactions on Visualization and Computer Graphics, Vol 1, Nº 2, June 1995.

[McCormick87] McCormick B., Defanti T, Brown M.: *"Visualization in Scientific Computing"*. Computer Graphics 21, 6 (1987).

[Moreland04] Moreland Kenneth**, "***Fast High Accuracy Volume Rendering"*, PhD Thesis, University of New Mexico and Sandia National Laboratories, July 2004.

[Neophytou05] Neophytou Neophytos, Mueller Klaus, *"GPU Accelerated Image Aligned Splatting"*, Volume Graphics 2005 I. Fujishiro, E. Gröller (Editors), 2005

[OVDE] Open Visualization Data Explorer: http://www.research.ibm.com/dx/

[Pascucci02] V. Pascucci, *"Slow growing subdivision (sgs) in any dimension: towards removing the curse of dimensionality"*, Computer Graphics Forum (Proc. EUROGRAPHICS'02), no. 3, 451 – 460.

[Popovic97] J. Popovic and H. Hoppe, *"Progressive simplicial complexes"*, ACM Computer Graphics Proc., Annual Conference Series, (SIGGRAPH 97), pp. 217–224.

[Porter84] T. Porter, T. Duff, *"Compositing digital images"*, Computer Graphics 18(3) (1984), 253–259.

[Rezk-Salama00] Rezk-Salama C, Engel K, Bauer M, Greiner G, Ertl T, *"Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization"*, In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 109-118, August 2000.

[Rivara84] Rivara María-Cecilia, *"Mesh refinement processes based on the generalized bisection of simplices"*, SIAM Journal on Numerical Analysis (1984), no. 3, 604–613.

[Roettger03] Roettger Stefan, Ertl Thomas, *"Cell Projection of Convex Polyhedra"*, In Volume Graphics 2003 Fujishiro, Mueller, Kaufman (Editors), 2003.

[Ronfard96] R. Ronfard, J. Rossignac. *"Full-range approximation of triangulated polyhedra"*. Computer Graphics Forum, 15(3), August 1996. Proc. Eurographics '96.

[Rorad06] Rorad Nicolas, Jones Mark W, *"Agents Based Visualization and Strategies"*, in Proceedings of WSCG'2006, pp 63–70, 8, 2006.

[Sabella88] Sabella Pablo, *"A Rendering Algorithm for Visualizing 3D Scalar Fields"* ACM Computer Graphics, Volume 22, Number 4, August 1988.

[Schroeder02] Schroeder W, Martin K, Lorensen B, "*The Visualization Toolkit",* Pearson Education, Inc, third edition, 2002. ISBN 1-930934-07-6.

[Shewchuk97] Shewchuk Jonathan Richard, *"Delaunay Refinement Mesh Generation"*, PhD. Thesis Dissertation, Carnegie Mellon University, 1997.

[Shirley90] Shirley P., Tuchman A., *"A Polyhedral Approximation to Direct Scalar Volume Rendering"*, Computer Graphics 24 (November, 1990), 63–70.

[Stegmaier05] Stegmaier Simon, Strengert Magnus, Klein Thomas, Ertl Thomas, *"A Simple and Flexible Volume Rendering Framework for Graphics-Hardware.based Raycasting"*, Volume Graphics (2005) To Appear, 2005.

[Stein94] Stein C, Becker B, Max N, *"Sorting and hardware assisted rendering for volume visualization"*, In *Proceedings of the 1994 Symposium on Volume Visualization*, pages 83-89, October 1994.

[Watt01] Watt Alan, Policarpo Fabio, *"3D Games Real Time Rendering and Software Technology"*, Addison-Wesley, First Edition, Volume one, ISBN 0-201-61921-0, 2001.

[Watt03] Watt Alan, Policarpo Fabio, *"3D Games Animation and Advanced Real-time Rendering"*, Addison-Wesley, First Edition, Volume two, ISBN 0-201-78706-7, 2003.

[Weiler00] Weiler, M, R Westermann, C Hansen, K Zimmerman, and T Ertl. *"Level-Of-Detail Volume Rendering via 3D Textures",* Proceedings of Volume Visualization and Graphics Symposium 2000. pp. 7-13.

[Weiler02] Weiler Manfred, Kraus Martin, Ertl Thomas, *"Hardware-based View-independent Cell Projection"*, In *Proceedings of IEEE Volume Visualization and Graphics Symposium 2002*, pages 13-22, October 2002.

[Weiler03] Weiler M, Kraus M, Merz M, Ertl T, *"Hardware-Based Ray Casting for Tetrahedral Meshes"*. In Proceedings of IEEE Visualization 2003, pages 333–340, 2003.

[Weiler04] Weiler M, P. Mallon N, Kraus M, Ertl T, *"Texture–Encoded Tetrahedral Strips"*. In *Proceedings of IEEE Volume Visualization'04*, 2004.

[Westover89] Westover, L.A., *"Interactive Volume Rendering"*, Proceedings of Volume Visualization Workshop (Chapel Hill, N.C., May 18-19), Department of Computer Science, University of North Carolina, Chapel Hill, N.C., 1989, pp. 9-16.

[Wilhelms91] Wilhelms Jane, van Gelder Allen, *"A Coherent Projection Approach for Direct Volume Rendering"*. In *Computer Graphics (ACM SIGGRAPH 91)*, volume 25, pages 275-284, July 1991.

[Wylie02] Wylie Brian, Moreland Kenneth, Fisk Lee Ann, Crossno Patricia, *"Tetrahedral Projection Using Vertex Shaders"*, In *Proceedings of the 2002 IEEE Symposium on Volume Visualization and Graphics*, pages 7-12, October 2002.

[Yagel92] Yagel, R, Kaufman, A., *''Template-Based Volume Viewing''*, Computer Graphics Forum, 11, 3 (September 1992), 153-167.

[Zhang97] H. Zhang and I. Kenneth E. Hoff. *"Fast backface culling using normal masks"*. In Proceedings of Symposium on Interactive 3D Graphics, pages 103–106, 1997.

[Zhou97] Y. Zhou, B. Chen, and A. Kaufman, *"Multiresolution tetrahedral framework for visualizing regular volume data"*, Proc. IEEE Visualization '97, pp. 135–142.