Universidad de las Ciencias Informáticas
Centro Vertex, Entornos Interactivos 3D, Facultad 4

Visualización de grandes volúmenes de datos empleando Octree

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Antonio David Castillo Oliva

Tutores: MSc. Luis Guillermo Silva Rojas

MSc. Rubén Alcolea Núñez

La Habana, 2018

Science is what we understand well enough to explain to a computer; art is everything else.

Donald Knuth

Dedicatoria

A mis padres, por su apoyo incondicional durante toda mi vida y ser mi ejemplo a seguir.
A mis abuelos, por ser la voz de la experiencia que me aconseja en todo momento.
A Amalia, por estar a mi lado en los momentos críticos, obligarme a esforzarme en todo y ser una persona indispensable en mi vida.
A mi prima Lixi, por siempre estar a mi lado brindandome apoyo y consejo.
A mi hermana, por ser un pilar de inspiración.

Agradecimientos

A mis tutores Guille y Rubén, por ser como otros padres conmigo, por sus revisiones y observaciones durante la investigación y el apoyo incondicional que me brindaron cada vez que se presentó un problema.
A Amalia, por ser una persona indispensable para mi, sin la cual, no hubiese logrado terminar esta carrera.
A la profe Zaida, por ser mi segunda madre y preocuparse por mi como si fuera su hijo.
A todos los muchachos del aula y el apartamento, por ser como mi otra familia y ayudarme a relajarme cuando lo necesitaba.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Antonio David Castillo Oliva
Autor

MSc. Luis Guillermo Silva Rojas
Tutor

MSc. Rubén Alcolea Núñez
Tutor

Resumen

El continuo avance de las tecnologías, ha traído consigo mejoras en todas las ramas de la ciencia. En la medicina, por ejemplo, se han combinado dispositivos de obtención de imágenes tridimensionales, tales como Tomografías Computarizadas e Imágenes por Resonancia Magnética, con programas informáticos que permiten visualizar en 3D el interior de los pacientes. El crecimiento en el tamaño de los datos volumétricos ha provocado una brecha con respecto a la memoria disponible en las tarjetas gráficas, lo que impide la visualización de volúmenes de alta resolución, cuando superan la memoria de la GPU. En el Centro de Entornos Interactivos 3D (Vertex), de la Universidad de las Ciencias Informáticas, se desarrolla el *software* Vismedic - Illustration, el cual permite generar ilustraciones interactivas a partir de datos volumétricos. En tal sentido, el presente trabajo tiene como objetivo desarrollar un algoritmo que permita visualizar imágenes que superen la capacidad de memoria de las tarjetas gráficas. Se realizó un estudio de las técnicas de optimización de la visualización, seleccionando la estructura de datos octree, por ser la que más se adapta a las necesidades. La propuesta de solución consiste en la implementación de un algoritmo para visualizar volúmenes de datos que combina el algoritmo *Raycasting* con la estructura de datos octree. Como resultado, con la solución propuesta, es posible visualizar volúmenes de datos que superan la capacidad de la tarjeta gráfica y que antes no era posible visualizar.

Palabras clave: volumen, octree, *Raycasting*, visualización.

Índice general

Índice de figuras

Índice de tablas

Introducción

Desde sus orígenes, el hombre utilizó las imágenes como herramientas de comunicación indispensables, en ellas, el hombre primitivo, expresaba sus intereses de caza en bocetos realizados en las cuevas. El sentido más utilizado por el hombre es la vista; se estima que el cerebro dedica el 50 % de las neuronas a dicho sentido [1]. Por otro lado, la densidad de información en las imágenes es mayor que la de un texto, si los datos que se quieren expresar son complejos o hay grandes cantidades de ellos, es lógico que se piense en el uso de imágenes para su representación.

La representación en forma de imágenes se utiliza en todas las ramas de la ciencia. La visualización de datos científicos registrados por distintos dispositivos de captación recibe el nombre de Visualización Científica; cuando estos son muestras representativas de una región de volumen se utiliza el término Visualización de Volúmenes [2]. Dentro de la Visualización Científica existe un campo especial, Visualización Médica; establecido como una área de investigación en la década de los 80. Existen tres eventos fundamentales que formaron las raíces de la visualización científica; el primero es la representación de sus trabajos por los científicos, un ejemplo son los dibujos de Leonardo Da Vinci (ver imagen 1), la segunda el procesamiento de imágenes, empleado fundamentalmente al analizar imágenes médicas y la tercera son los gráficos por computadora, que proveen una serie de algoritmos indispensables para la representación [3]. Las visualizaciones médicas tienen su inicio en las técnicas tradicionales de diagnóstico con rayos X y ha tenido un desarrollo considerable en las últimas décadas con la aparición de nuevos dispositivos de captación como las Tomografías Computarizadas (CT, por sus siglas en inglés), las Imágenes por Resonancia Magnética (MRI, por sus siglas en inglés) y las Tomografías por Emisión de Positrones (PET, por sus siglas en inglés), que proporcionan información realmente tridimensional de las estructuras [2].

En la actualidad, los avances tecnológicos, han permitido que los equipos médicos posean sistemas informáticos que permiten manipular las imágenes médicas y visualizarlas de forma tridimensional. Año tras año son más los especialistas que sustituyen sus equipos tradicionales por programas informáticos que le permitan realizar acciones como ajustar el contraste o mejorar el brillo, aunque el proceso de obtención de la imagen no haya sido el deseado, para mejorar de esa forma la calidad del diagnóstico emitido.

A la hora de seleccionar dicho software, existe una gran variedad, desde muy modestos con características limitadas, hasta otros con gran cantidad de funcionalidades pero extremadamente costosos. Internacionalmente en este ámbito destacan empresas como PHILIPS y SIEMENS, también existen herramientas dedicadas al desarrollo de aplicaciones de visualización de volúmenes. Un ejemplo es el programa OsiriX [4] para MAC OS X®, que se presenta como una herramienta útil en el manejo de imágenes tomográficas

Figura 1. Ejemplo de ilustración anatómica realizada por Da Vinci.

para el diagnóstico y planificación preoperatoria en pacientes con fracturas del esqueleto facial. También existe el caso del programa de código abierto Voreen [5] e InViWo [6], ambas herramientas punteras en la visualización de volúmenes; además vale destacar el trabajo realizado en este campo por el grupo VRVis [7] de la Universidad Tecnológica de Viena.

En Cuba, uno de los pilares en esta rama, lo constituye el proyecto IMAGIS [8], desarrollado en el Centro de Biofísica Médica de la Universidad de Oriente y desplegado en la mayoría del Sistema Nacional de Salud Cubano. En la Universidad de las Ciencias Informáticas (UCI) existe el *software* alas PACS, desarrollado en el Centro de Informática Médica (CESIN), diseñado para visualizar imágenes médicas y es el estipulado por el Ministerio de Salud Pública para su uso en instalaciones hospitalarias [9]. En el Centro de Entornos Interactivos 3D (Vertex) de la Facultad 4 se desarrolla el *software* Vismedic - Illustration, el cual permite a través de imágenes médicas volumétricas en formato DICOM visualizar estructuras anatómicas en forma tridimensional empleando la Unidad de Procesamiento Gráfico (GPU, por sus siglas en inglés).

Para lograr la visualización Vismedic emplea el algoritmo *Raycasting* basado en GPU (ver epígrafe 1.5), para lo que se necesita enviar el volumen completo a la GPU. El algoritmo *Raycasting* implementado en Vismedic permite la visualización en tiempo real de volúmenes de mediano tamaño (menos de 512^3). Sin embargo, los volúmenes de alta resolución, obtenidos por dispositivos modernos de adquisición de datos volumétricos, usualmente exceden la capacidad de almacenamiento de la memoria de video de las tarjetas gráficas. Lo que causa que los volúmenes de mejor resolución no puedan visualizarse en la aplicación. Teniendo en cuenta la situación problémica se plantea como **problema de la investigación:** ¿Cómo visualizar de forma interactiva volúmenes de datos que superan la capacidad de memoria de la GPU? A partir del problema planteado, se define como **objeto de estudio** la visualización de volúmenes y como **campo de acción** la visualización de grandes volúmenes de datos que superen la capacidad de memoria de las tarjetas gráficas. Además se define como **objetivo general** implementar un algoritmo de visualización de volúmenes que permita visualizar, de forma interactiva, los volúmenes de datos que exceden la capacidad de la memoria de la GPU. Este trabajo defiende la siguiente idea: el empleo de técnicas de optimización de visualización directa de volumen permitirá la representación de grandes volúmenes de datos en el proyecto Vismedic.

Para dar cumplimiento al objetivo planteado es necesario realizar las siguientes **tareas de la investigación:**

1. Elaboración del marco teórico a partir del estado del arte actual referente al tema.
2. Selección de la metodología, herramientas de software y lenguajes de programación para el desarrollo del algoritmo.
3. Implementación del algoritmo seleccionado.
4. Integración del algoritmo seleccionado con la aplicación principal del proyecto Vismedic.
5. Validación de los resultados a través de pruebas de rendimiento.

Para realizar la investigación y elaborar el siguiente trabajo se emplearon varios **métodos científicos de investigación**, entre los cuales se pueden mencionar los siguientes:

Métodos teóricos:

- **Histórico – Lógico:** Mediante este método se analizará la evolución y desarrollo del objeto de investigación y sus elementos más importantes; las variantes que presenta y sus formas de optimización, con el fin de seleccionar la más adecuada para la solución.
- **Analítico – Sintético:** Se usará para analizar la información de las técnicas existentes para visualización de volúmenes con el objetivo de seleccionar un conjunto de ellas que contribuyan a la presente investigación.

Métodos empíricos:

- **Pruebas:** Se realizarán diferentes pruebas al algoritmo implementado para determinar si se comporta según los resultados esperados.
- **Observación:** Se empleará para constatar los resultados tanto visuales como espaciales alcanzados y determinar la influencia de estos sobre el rendimiento de la GPU.

A continuación se muestra la estructura del presente trabajo donde se incluye la síntesis de los capítulos y secciones fundamentales:

Capítulo 1. Fundamentación teórica.
En este capítulo se definen los principales conceptos que serán empleados durante todo el trabajo, además se presentan las bases teóricas fundamentales relacionadas con la optimización de la visualización de volúmenes. Se abordan conceptos de adquisición de datos volumétricos y técnicas de optimización para su visualización, mostrando las ventajas que pueden traer estos últimos para un mejor aprovechamiento de la GPU.

Capítulo 2. Propuesta de solución.
Durante este capítulo se describe la solución propuesta, así como su integración con la arquitectura de Vismedic − Illistration. También se presentan los algoritmos implementados, al igual que una descripción de las clases fundamentales. Además, se detallan las herramientas y metodologías de desarrollo empleadas en conjunto con los resultados ingenieriles de la metodología seleccionada.

Capítulo 3. Evaluación de los resultados.
En este capítulo se toman casos de prueba para validar los resultados alcanzados, fundamentalmente relacionados con el rendimiento de la GPU. Además se realizan pruebas de aceptación, con el objetivo de verificar si la solución cumple con los requisitos establecidos. De igual forma, se muestran evidencias visuales de los resultados alcanzados con volúmenes de datos incapaces de visualizar sin el empleo de la solución.

Glosario de términos.
Se elaboró un Glosario de Términos con el objetivo de facilitar la comprensión del lenguaje utilizado relacionado con el objeto de estudio.

CAPÍTULO 1

Fundamentación teórica

El presente capítulo aborda los principales conceptos y bases teóricas fundamentales de la obtención y visualización de datos volumétricos. Además se profundizan conceptos y algoritmos de visualización; al igual que los principales métodos de optimización. También se muestran descripciones de otros trabajos cuyo objetivo es similar al de la investigación.

1.1. Datos volumétricos

Según el diccionario de la Real Academia de la Lengua Española (RAE), un dato se define en la Informática como "información dispuesta de manera adecuada para su tratamiento por una computadora", y volumen como la "magnitud física que expresa la extensión de un cuerpo en tres dimensiones"; por tanto, se puede definir como dato volumétrico a la información de un cuerpo u objeto para su representación en tres dimensiones por la computadora.

Los datos adquiridos para la representación de volúmenes se representan generalmente como un conjunto de imágenes individuales. Cada imagen representa un pequeño corte del objeto escaneado y está compuesta por píxeles (elementos de imagen). Estos píxeles se ordenan en una red bidimensional donde la distancia entre dos píxeles (*pixel distance*) es constante en cada dirección. Para la mayoría de la imágenes médicas, las direcciones horizontal (x) y vertical (y) tienen distancias idénticas; esto permite el cálculo de la posición actual multiplicando el valor de la distancia por el índice del píxel. Si se asume que i indexa en la posición horizontal **x** y j en la posición vertical **y**, la posición del píxel $P_{i,j}$ puede ser calculada [10]. La imagen 1.1 Izquierda muestra la red 2D de la imagen.

Los datos volumétricos combinan imágenes en una red 3D (ver imagen 1.1 Centro). Los elementos ahora pasan a llamarse vóxeles (elementos de volumen) y están localizados en los puntos de la red. En adición a las dimensiones horizontal **x** y vertical **y** también se tiene una dimensión representando la profundidad (**z**). La distancia entre dos imágenes continuas se denomina **distancia entre cortes** (*slice distance*) y la distancia entre las tres dimensiones es conocida como *voxel spacing*. La posición del vóxel $V_{i,j,k}$ dentro del volumen se determina por los valores de distancia y el índice.

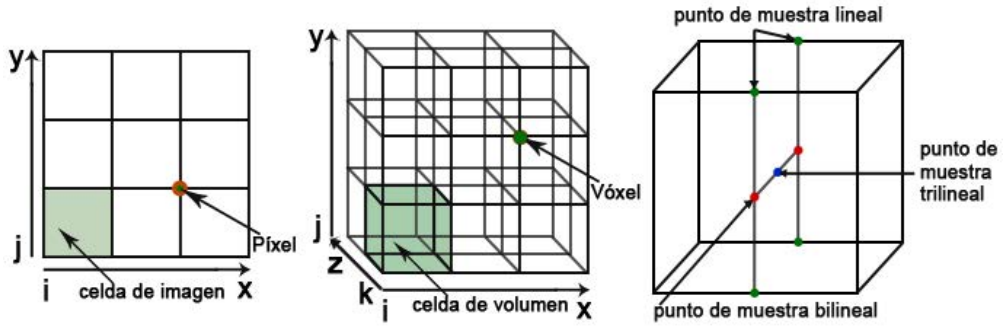

Figura 1.1. Izquierda: Red de imagen 2D. Centro: Volumen de datos. Derecha: Interpolación trilineal [11].

Un volumen de datos está definido por posiciones discretas de la red 3D, aunque frecuentemente es necesario calcular puntos dentro de una celda de volumen; el método más empleado para calcular dichos puntos es la interpolación trilineal [10], compuesta por siete interpolaciones lineales (ver imagen 1.1 Derecha). Como primer paso, cuatro interpolaciones lineales calculan los puntos entre dos vóxeles vecinos en un borde de la celda del volumen en una dirección. En el segundo paso se combinan dos interpolaciones lineales entre vóxeles localizados en la misma cara de la celda del volumen para computar el resultado de la interpolación bilineal. Finalmente, el resultado obtenido de las dos interpolaciones bilineales en las caras opuestas de la celda del volumen, se combinan por interpolación lineal para obtener el punto de muestra trilineal [11].

1.2. Técnicas de adquisición de datos volumétricos

Los datos volumétricos se utilizan en diferentes ramas de la ciencia tales como la Geología, Arqueología y Meteorología, por mencionar algunas; la Medicina ha jugado un papel fundamental en el desarrollo y uso de las modalidades de adquisición fundamentales. Los datos adquiridos se utilizan con diversos propósitos como diagnóstico, planeamiento de terapia, monitoreo post-operación e investigación médica. A continuación se resumen los principios de funcionamiento de los principales métodos.

1.2.1. Rayos X

Los rayos X fueros descubiertos por Wilhelm Conrad Röntgen en 1895, marcando el nacimiento de las imágenes médicas, por primera vez las partes interiores del cuerpo humano podían ser visualizadas sin la necesidad de cortar la piel, esta técnica se basa en el atenuación de la cantidad de rayos X que viajan a través del objeto escaneado [10] (ver imagen 1.2).

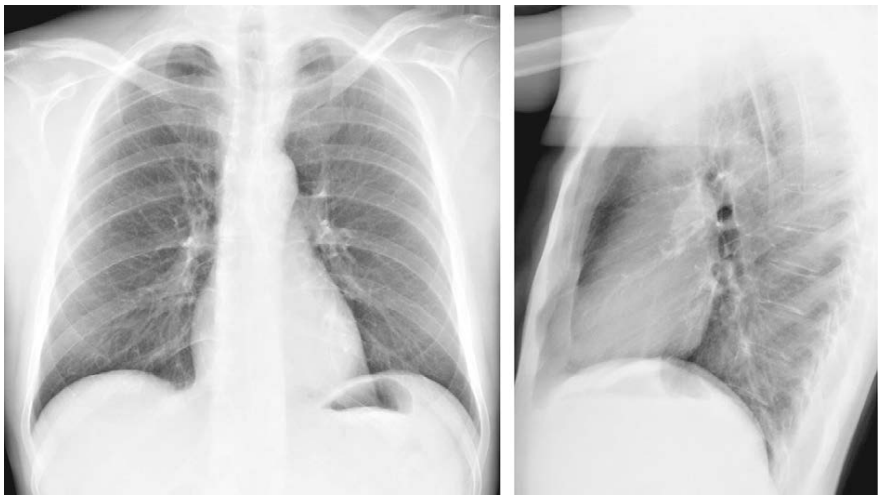

Figura 1.2. Vistas frontal y lateral del tórax obtenidas con rayos X.

1.2.2. Tomografía Computarizada

Las CT fueron introducidas en 1968 por Godfrey Hounsfield y Allan McLeod Cormack, propiciando la primera representación volumétrica de objetos; en esencia esta técnica representa una serie de imágenes individuales de rayos X que componen el volumen, dichas imágenes se adquieren mediante un sistema de emisor/receptor que se encuentra rotando alrededor del objeto escaneado [10].
El inconveniente de dicha técnica son las grandes cantidades de radiación que recibe el paciente. Una vez adquiridos, los valores se normalizan en unidades Hounsfield, donde el agua se representa con el 0 y el aire con -1000. Para mejorar la visualización de ciertas arterias o tejidos es necesaria la adición de agentes contrastantes dentro del objeto escaneado para resaltar dichas áreas [12] (ver imagen 1.3).

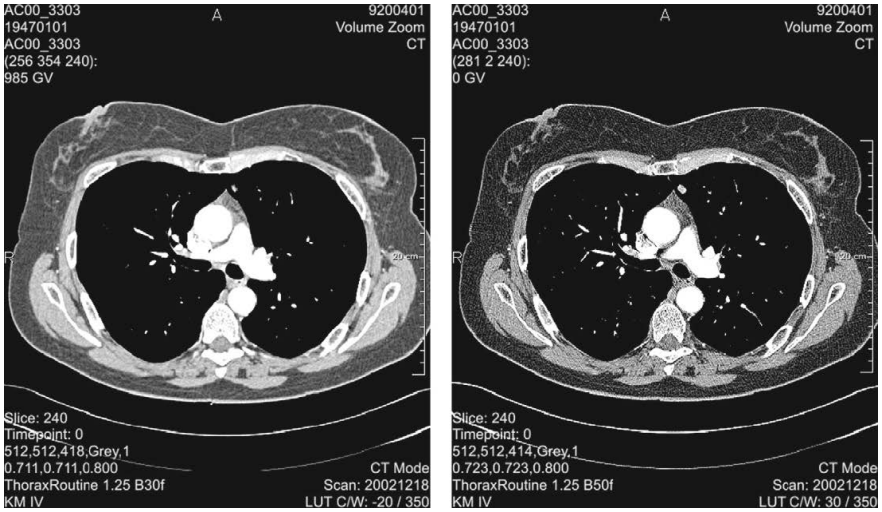

Figura 1.3. Imagen parcial del tórax obtenida mediante tomografías computarizadas.

1.2.3. Imágenes por Resonancia Magnética

Las MRI se basan en las diferentes propiedades del tejido humano en un campo magnético, en particular la ocurrencia de núcleos de hidrógeno, que pueden ser considerados como imanes dipolos pequeños que se alinean paralelos o anti-paralelos a lo largo del campo magnético [10].
Esta técnica se utiliza con frecuencia para representar tejidos blandos, como el cerebro, sin embargo no es recomendado utilizarla para tejidos con poca agua como los huesos. Una ventaja que tiene con respecto a las CT es que no emite radiación que pueda ser perjudicial para el paciente [12] (ver imagen 1.4).

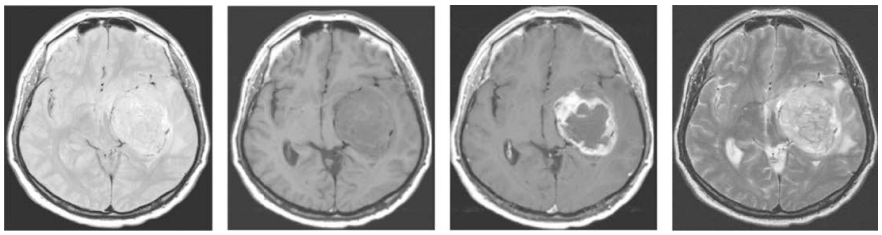

Figura 1.4. Set de cuatro resonancias magnéticas para proveer información complementaria sobre el diagnóstico de un tumor cerebral.

1.2.4. Imagen por Resonancia Magnética Funcional

Las Imágenes por Resonancia Magnética Funcional (fMRI, por sus siglas en inglés) son utilizadas para representar la actividad neuronal de ciertas partes del cerebro mientras los pacientes realizan actividades cognitivas o de comportamiento. Dicha técnica detecta cambios en el flujo sanguíneo y el metabolismo del oxígeno en el cerebro. Las áreas activadas encontradas mediante fMRI son usualmente superpuestas a una MRI para tener noción de la orientación espacial y localización de las áreas del cerebro [12] (ver imagen 1.5).

1.2.5. Ultrasonido

El ultrasonido se basa en ondas de sonido que son emitidas a frecuencias muy altas desde la sonda hacia la respectiva parte del cuerpo, estas son parcialmente reflejadas si golpean una interfaz entre dos tipos de tejidos; las ondas reflejadas son grabadas por sensores localizados cerca de la fuente emisora y, al igual que un sonar, se reciben por un transductor y se usan para generar las imágenes [10] (ver figura 1.6).

1.2.6. Tomografía por Emisión de Positrones

La PET es una técnica de la medicina nuclear que representa la actividad metabólica de los tejidos. Funciona inyectando una sustancia radioactiva de corta vida en el paciente, que cuando comienza a decaer emite positrones y estos al interactuar con un electrón (del cuerpo humano) genera dos fotones *gamma* que viajan en dirección opuesta. Estos fotones pueden ser medidos y utilizados para calcular su posición inicial.

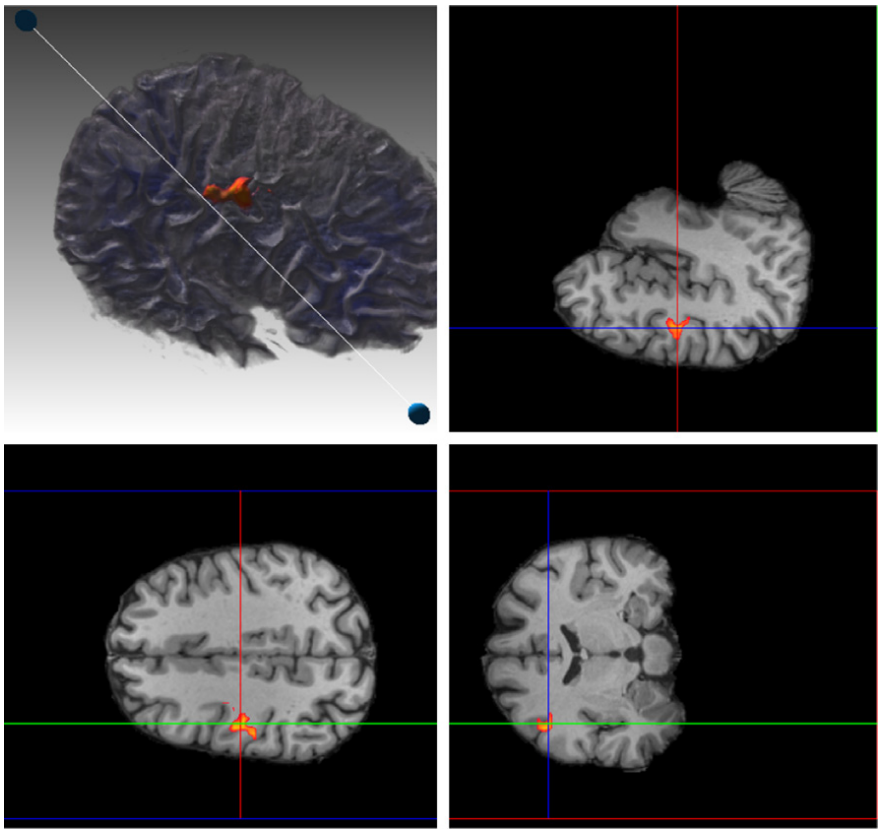

Figura 1.5. Imágenes por resonancia magnética funcional obtenidas después de una simulación de audición.

Esta técnica tiene una resolución espacial muy baja pero es muy útil en la representación de áreas de gran actividad metabólica como los tumores [12] (ver imagen 1.7).

1.3. Visualización de volúmenes

La disciplina Visualización se encarga de la representación de información significativa a partir de varios tipos de datos no gráficos. La Visualización de Volúmenes en particular consiste en métodos de extracción, clasificación y representación de información contenida en conjuntos de datos volumétricos tales como volúmenes, *datasets* o modelos. Estos son colecciones de valores escalares que definen una propiedad física medible en una región del espacio como la densidad [13]. Al emplear una combinación de técnicas de muestreo, filtrado, clasificación y visualización, se genera una representación gráfica de los objetos contenidos en dicha región del espacio basada en los valores escalares asociados.

Las técnicas de Visualización de Volúmenes normalmente se dividen en dos grupos: Visualización Indirecta de Volumen (IVR, por sus siglas en inglés) y Visualización Directa de Volumen (DVR, por sus siglas en inglés). El primer grupo calcula inicialmente una representación intermedia del volumen que luego se mostrará en la pantalla. Los algoritmos de IVR se basan con frecuencia en el uso de planos como el *Planar*

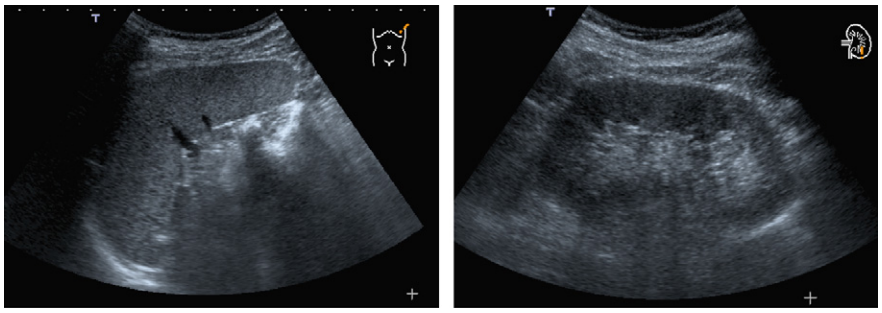

Figura 1.6. Imágenes de ultrasonidos que representan el bazo (izquierda) y el riñón (derecha).

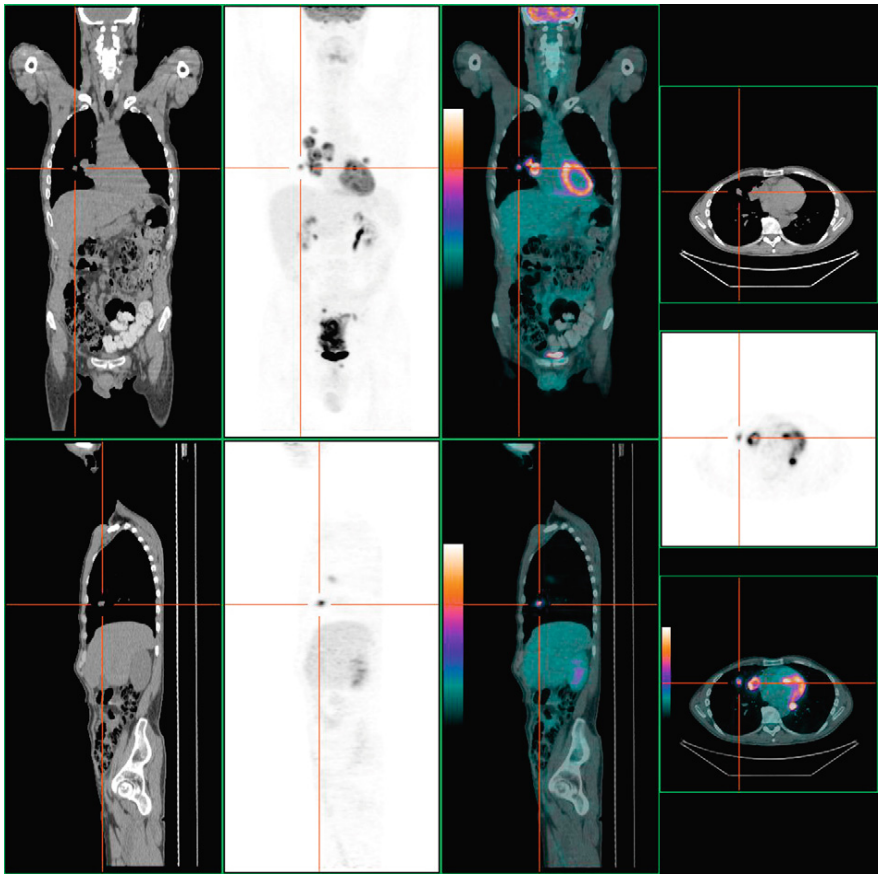

Figura 1.7. Datos obtenidos mediante CT y PET mostrados de forma aislada (izquierda) y superpuestas (derecha) [10].

Rendering o superficies geométricas como el *Surface Rendering* para la realización de la representación intermedia [11]. En la extracción de iso-superficies un algoritmo clásico es *Marching Cubes* [14], encargado de construir una superficie triangular a partir del análisis de cada vóxel. Dicho algoritmo, como todos los métodos de extracción de superficies, está basado en una decisión binaria a partir de la pertenencia o no de las porciones de los vóxeles a la superficie. Aunque es apropiado para distinguir entre diferentes superficies escaneadas por CT, presenta limitaciones sensibles al ruido y a volúmenes con superficies poco definidas.

La DVR, por otro lado, genera imágenes de conjuntos de datos volumétricos sin la necesidad de extraer explícitamente superficies geométricas [12]. Esta se basa en un modelo óptico que define como el volumen emite, refleja, dispersa y ocluye la luz [15].

1.4. Algoritmo Raycasting

Raycasting es considerado como el algoritmo por defecto para DVR debido a su flexibilidad y capacidad de asimilar mejoras de rendimiento y calidad de imagen. Fue propuesto inicialmente por Mark Levoy en 1988. De forma general, una vez obtenidos los datos volumétricos, el algoritmo se puede resumir conceptualmente en los siguientes pasos:

1. Por cada píxel de la imagen a representar, se traza uno o más rayos que atraviesan el volumen.
2. A lo largo de cada rayo se reconstruyen las muestras, frecuentemente tomadas a intervalos regulares (Δs). Generalmente se emplea Interpolación Trilineal como filtro de reconstrucción.
3. La muestra interpolada y diferentes valores asociados, tales como las derivadas de primer y segundo orden, se transforman en color y opacidad mediante la función de transferencia.
4. Los colores y opacidades obtenidos se iluminan, frecuentemente empleando el modelo de iluminación de Phong. En la mayoría de los casos se necesita el gradiente local del volumen, como una aproximación de la normal de la superficie local, con el objetivo de simular la interacción con fuentes de luz direccionales. El método de cálculo de gradiente más usado es Diferencias Centrales.
5. El color y opacidad obtenido durante la iluminación se integra con las muestras anteriores a lo largo del rayo.
6. Cuando el rayo abandona el volumen, se le asigna el color y la opacidad acumulados al píxel del cual se originó el rayo.
7. Cuando se obtiene el color y la opacidad de todos los píxeles, se proyecta en pantalla la imagen resultante.

Aunque existen numerosas implementaciones del algoritmo, generalmente solo se modifican algunos de los pasos descritos anteriormente con el objetivo de optimizar el rendimiento, la calidad de la visualización o para adaptarlo a las características específicas de la aplicación [11].

1.5. Raycasting basado en GPU

La idea general para portar el algoritmo *Raycasting* a la GPU, es almacenar el volumen en una textura 3D. El *hardware* gráfico tiene implementado de forma eficiente la manipulación y acceso a las texturas 3D. Seguidamente se ejecuta un *fragment shader* para simular el trazado de los rayos hacia el volumen. El *fragment shader* se ejecuta una vez por cada fragmento, ajustándose naturalmente al trazado de rayos del *Raycasting*. Es necesario realizar un conjunto de pasos para configurar el *fragment shader*. Para determinar las posiciones de entrada y salida de los rayos dentro del volumen, es necesario representar una geometría

intermedia que envuelva el volumen completamente. La geometría envolvente más sencilla es un cubo, donde los colores de los vértices coinciden con sus posiciones espaciales. De esta forma la representación de las caras delanteras codifica, en el *buffer* de color, las posiciones 3D de entrada de los rayos en el volumen (ver imagen 1.8 a). De igual manera sucede con la representación de las caras traseras y posiciones de salida (ver imagen 1.8 b). Al conocer los puntos de entrada y salida del volumen, se pueden calcular los vectores de dirección para cada rayo (ver imagen 1.8 c), lo que permite avanzar pequeños intervalos en la dirección del rayo para tomar las muestras. En cada paso las muestras se interpolan, se evalúan en las funciones de transferencia, se iluminan y se mezclan con las anteriores. Al concluir con la evaluación de todos los rayos, la imagen final se proyecta en pantalla [16].

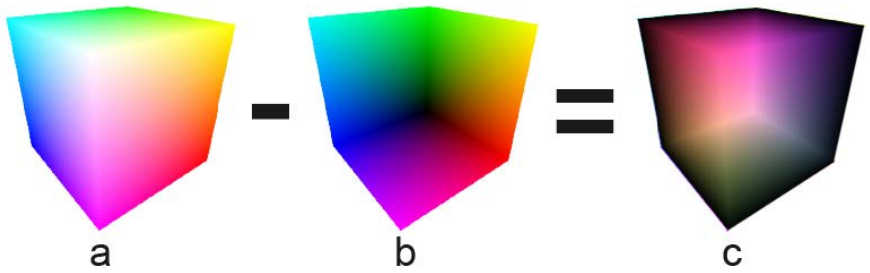

Figura 1.8. Representación de la geometría envolvente para el algoritmo Raycasting. a) Caras delanteras. b) Caras traseras. c) Textura direccional.

La ejecución del *fragment shader* del *Raycasting* comienza con la representación de las caras delanteras con el *shader* activado. De esta forma se llama automáticamente al *fragment shader* por cada píxel perteneciente a las caras frontales. El algoritmo *Raycasting* basado en GPU se resume en un proceso de cuatro pasos [17]:

1. **Generación de las caras delanteras:** se representan las caras delanteras de la caja envolvente del volumen y se almacenan en una textura.
2. **Generación de la textura direccional:** se representan las caras traseras de la caja envolvente del volumen, se guarda en otra textura y se le sustrae a la textura generada en el paso 1. Los vectores resultantes se almacenan en otra textura (imagen 1.8 c).
3. ***Raycasting:*** se toma la posición inicial de la textura generada en el paso 1 y a lo largo del vector direccional se toman muestras mientras el rayo no se salga del volumen o mientras que la opacidad acumulada sea menor que 1.0 (terminación temprana del rayo).
4. **Presentación:** el resultado del *Raycasting* se presenta en la pantalla.

En la mayoría de las implementaciones basadas en GPU solo es necesario modificar el paso 3, donde se ejecuta el *fragment shader*. En este paso se debe tener en cuenta para cada muestra la función de transferencia y la iluminación.

1.5.1. Vertex Shaders

Los *vertex shaders* son programas que se ejecutan en el *Vertex Processor* de la GPU, una vez por cada vértice de la geometría que se quiere representar. El *vertex shader* no puede crear ni eliminar vértices, solo modificarlos. Tradicionalmente esta unidad ejecuta una serie de operaciones fijas en función de las capacidades de la tarjeta gráfica:

- Transformaciones de vértices.
- Transformación y normalización de vectores.
- Generación de coordenadas de textura.
- Mapeo de coordenadas de textura.
- Iluminación.

De todas las funciones, la más importante para el *Raycasting* es la transformación de vértices, la cual se realiza mediante la multiplicación por la matriz de modelado (*Model-View Matrix*) y por la matriz de proyección (*Projection Matrix*) [3].

1.5.2. Fragment Shaders

Los *fragment shaders* son programas que se ejecutan en el *Fragment Processor* de la GPU, una vez por cada fragmento. Los fragmentos corresponden a un píxel para el *shader* actual, esto significa que para un píxel de la imagen final se pueden evaluar varios fragmentos pertenecientes a diferentes *shaders*. El *Fragment Processor*, en función de las capacidades de la GPU, puede ejecutar diferentes operaciones; algunas de sus principales funcionalidades son las siguientes:

- Operaciones vectoriales y matriciales de punto flotante con valores interpolados.
- Acceso, interpolación y mapeo de texturas.
- Modulación y mezcla de colores.
- Iluminación por fragmentos.

La característica fundamental de los *fragment shaders*, desde el punto de vista del algoritmo *Raycasting*, es su modo de ejecución: **una vez por cada fragmento de la geometría**. Su comportamiento es muy semejante al principio de funcionamiento del algoritmo *Raycasting*: **un rayo por cada píxel de la imagen a representar**. La evaluación de los *fragment shaders* se realiza automáticamente de forma paralela, contribuyendo directamente al rendimiento del algoritmo [11].

1.5.3. GLSL

OpenGL Shading Language o GLSL como es llamado comúnmente, es un lenguaje de alto nivel con una sintaxis basada en el lenguaje de programación C. GLSL ofrece un rico set de instrucciones que unifican el procesado de vértices y fragmentos. Además ofrece más control a los desarrolladores sobre el *rendering pipeline* sin tener que usar lenguajes específicos para cada *hardware* [18].

1.6. Técnicas de optimización para DVR

A la hora de representar los datos volumétricos aparecen diferentes contratiempos, entre los más comunes se encuentra la complejidad de los datos, adicionalmente, con técnicas de adquisición modernas, el espacio de almacenamiento supera las tarjetas gráficas disponibles, dificultando su representación en tiempo real. Para solucionar dichas dificultades se crearon diferentes métodos de optimización de visualización de volúmenes.

1.6.1. Estructuras de datos

Una estructura de datos es una forma particular de organizar datos en una computadora para que puedan usarse eficientemente. Existen diferentes tipos de estructuras de datos y cada una es adecuada para una aplicación específica. Una estructura eficiente es clave para el diseño de algoritmos eficientes. Esta sección muestra diferentes tipos especializados en optimizar la DVR.

Octree

A la hora de describir un objeto mediante todos sus vóxeles se generan grandes listas de datos. La enumeración de modelos hechos por descomposición tiene muchas ventajas como simplicidad, generalidad y permiten un gran número de algoritmos, sin embargo el consumo de memoria es alto; para resolver dicho inconveniente se puede cambiar la división regular del espacio por una más eficiente, una división adaptativa. La técnica de octree es el principal ejemplo de esta nueva división.

Partiendo de un modelo inicial (ver imagen 1.9 nodo raíz), los octrees utilizan la división recursiva del espacio en ocho octantes que se ubican en una estructura lógica de árbol (ver imagen 1.9 nivel 1). La idea fundamental es un algoritmo divide y vencerás. Si el octante contiene elementos del volumen se describe como *full*, en caso contrario como *empty*. El espacio es dividido recursivamente hasta alcanzar el nivel de profundidad deseado (ver imagen 1.9 nivel 2).

El número de nodos es proporcional a la superficie del objeto [19]. La división del espacio en un número constante de nodos, permite un elevado nivel de detalle sin la necesidad de realizar demasiadas iteraciones. Además, la aplicación de la estructura presenta ventajas a la hora de aplicar nuevos métodos de optimización, debido a que la estructura de octree genera una división jerárquica y regular del espacio.

Kd-tree

Un kd-tree es una estructura de datos ligeramente diferente a octree. En lugar de dividir el volumen en las tres dimensiones al mismo tiempo, solo se escoge una, generando una partición binaria cada ves que se subdivide. La orientación de la subdivisión puede cambiar en cualquier paso [10]. Dado que hay muchas opciones para escoger los planos por donde se divide la estructura, hay diferentes formas de generar kd-trees. El sistema habitual para su construcción es:

- Conforme se desciende en el árbol, se emplean ciclos a través de los ejes para la selección de los planos; esto quiere decir que, si en el nodo raíz se seleccionó el eje **x**, en sus descendientes se escogería el eje **y** y en sus descendientes el eje **z**.

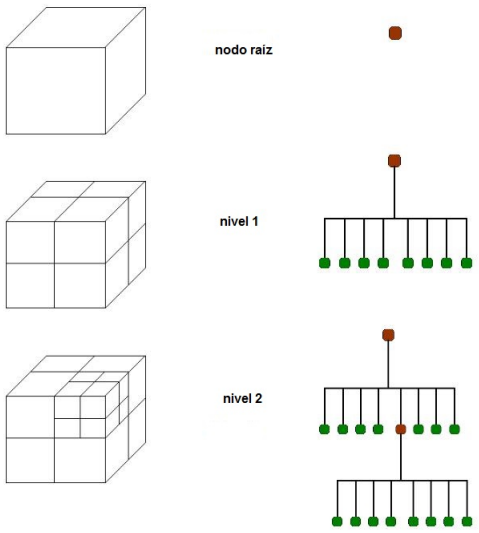

Figura 1.9. División del volumen aplicando la estructura de datos octree.

- En cada paso, el punto seleccionado para crear el corte sería la mediana de los puntos puestos en el árbol, lo que respeta sus coordenadas en el eje que está siendo usado.

De esta forma se obtiene un kd-tree balanceado, donde cada nodo hoja está a la misma distancia de la raíz. La forma de árbol binario de este algoritmo, provoca que sean necesarias un gran número de iteraciones si se quiere obtener mayor nivel de detalle en los nodos. Por otro lado, la posibilidad de variar, tanto el eje de corte como la distancia de corte, provoca que no necesariamente la división final del espacio sea regular, lo que dificulta un poco el trabajo a la hora de aplicar otros métodos de optimización.

1.6.2. Bricking

Como se mencionó anteriormente uno de los problemas más comunes en la Visualización Médica es el tamaño significativo de los datos volumétricos, una solución intuitiva para dicho inconveniente es el ***bricking***, o sea, dividir el volumen en otros más pequeños que no excedan la capacidad de la GPU. *Bricking* se basa en la filosofía "divide y vencerás", se encarga de subdividir el mundo de vóxeles en *bricks* (bloques) de menores dimensiones, de manera tal que, a la hora de representar el volumen, en lugar de tener un único elemento que sobrepasa el tamaño de la tarjeta gráfica, se tienen varios que individualmente caben en la memoria interna de la GPU. Los *bricks* se almacenan en la memoria principal, y se representan en orden *back to front* (de atrás hacia delante), obteniendo así el volumen global. El *bricking* no reduce la cantidad de memoria necesaria para representar los datos del volumen original. Cada *brick* se debe enviar a la memoria local de la GPU para su representación. El rendimiento de dicha técnica está sujeto a la velocidad de transferencia entre la memoria principal y la memoria local de la GPU (ver imagen 1.10). Este proceso puede ser visto como una jerarquía de cachés. Primero se tiene el modelo en el disco, se envía a la memoria principal si no excede su tamaño, en caso contrario habría que aplicar métodos *Out of core* (ver epígrafe

1.6.3). Una vez en la memoria principal se envía a la memoria gráfica, si no sobrepasa su capacidad se visualiza, en caso contrario se aplica *bricking* [20].

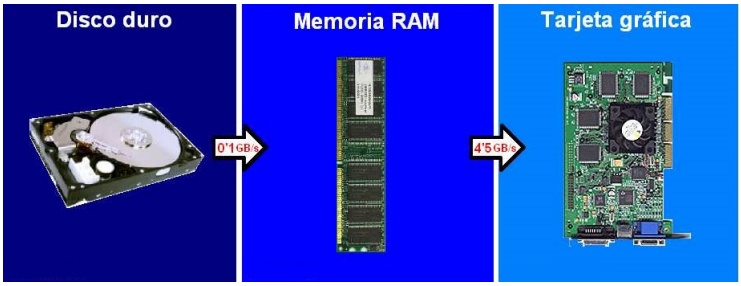

Figura 1.10. Esquema del flujo de datos entre los distintos dispositivos de almacenamiento.

El brickin no implica pérdida de precisión o detalle, por lo que es ideal para el trabajo con imágenes médicas, que necesitan la máxima precisión pues, el mínimo detalle puede ser importante para el profesional a la hora de sacar conclusiones sobre lo que está viendo. Otro factor determinante a favor del método es la sencillez de la idea y la menor complejidad a la hora de la implementación. Por otro lado, la dependencia de la velocidad de transferencia y la no disminución del tamaño del *dataset*, provoca que solo pueda ser implementada en Computadora Personal (PC, por sus siglas en inglés) con características elevadas.

1.6.3. Out of core

La visualización de grandes volúmenes de datos en una PC básica es una tarea difícil. El principal desafío es la brecha existente entre el tamaño de la memoria principal y el tamaño de los conjuntos de datos. Dicha brecha no solo existe, sino que se está ensanchando. Aunque los tamaños de memoria están creciendo exponencialmente, los conjuntos de datos están creciendo más rápido [21]. Para cerrar esta brecha se implementan algoritmos *out of core*, también conocidos como algoritmos externos o algoritmos de memoria secundaria. Estos mantienen la mayor parte de los datos en el disco, y en la memoria principal solo la parte de los datos que se está procesando; provocando que el rendimiento dependa de la velocidad de transferencia entre el disco duro, la Memoria de Acceso Aleatorio (RAM, por sus siglas en inglés) y la GPU.

1.7. Trabajos relacionados

CellFast

Esta solución utiliza el acercamiento de la proyección de tetraedros de Shirley y Tuchman [22], que es un rápido algoritmo para la visualización de volumen no estructurado, para ello genera triángulos que pueden ser representados por aceleración de *hardware*. También propone algunas variantes de optimización con el objetivo de lograr el mayor rendimiento posible para la visualización interactiva de datos no estructurados, entre las cuales se encuentran: la utilización de *OpenGL*, una variante personalizada del algoritmo *quicksort*, organización de memoria para mejorar la eficiencia de la caché, listas de visualización y eliminación

tetraédrica.

Templeted-Based Octree Projection

Presenta un nuevo algoritmo de renderizado que utiliza *Raycasting* y mapeo de texturas. El volumen en cuestión es almacenado y representado en una estructura de datos octree. El tamaño de los bloques es escogido en cada iteración para minimizar el procesamiento innecesario de vóxeles vacíos [23]. El algoritmo explota la forma uniforme, la orientación y el tamaño de los bloques del octree para la construcción de plantillas para la interacción de rayos con bloques en el caso de *Raycasting*, e intersecciones del plano Z con los bloques en el caso del mapeo de texturas. Luego las plantillas se pegan en todos los bloques de la estructura de datos octree, evitando así el costoso cálculo de intersección.

1.8. Consideraciones parciales

Hasta este punto se puede concluir que el estudio de los principales conceptos relacionados con el tema formaron las bases científicas de la investigación. El análisis de los principales métodos de obtención de datos volumétricos permitió un mayor entendimiento del funcionamiento de dichas técnicas de obtención. Por otro lado, se presentaron varias técnicas de optimización de DVR entre las cuales se encontraban las estructuras de datos octree y kd-tree. La estructura seleccionada para aplicar a la solución fue octree debido a que se implementa de forma sencilla y rápida. Además, como resultado se obtiene una división regular del espacio, cosa que no sucede en kd-tree, lo que posibilita la aplicación de otros métodos de optimización. Por otro lado, para generar árboles con suficiente nivel de detalle en cuanto a cantidad de nodos, se necesitan menos cantidad de iteraciones con octree que con kd-tree. En el caso de *bricking*, si bien la subdivisión del volumen en fragmentos iguales con tamaños menores a la capacidad de la GPU permite visualizar los grandes volúmenes, al no tener una estructura jerárquica definida, dificulta la aplicación de nuevos algoritmos de optimización.

CAPÍTULO 2

Propuesta de solución

En el presente capítulo se presenta la solución propuesta para la visualización de datos volumétricos empleando octree. De igual forma, se describe la arquitectura que presenta el *software* Vismedic - Illustration. Además se detallan los elementos fundamentales de la solución en cuestión y se enfatiza en la integración de esta con la arquitectura existente en el sistema Vismedic-Illustration. Finalmente, se brinda una descripción de las clases que se adicionaron y del funcionamiento de los principales algoritmos implementados.

2.1. Solución

El objetivo a alcanzar durante el desarrollo es, lograr visualizar volúmenes que excedan la capacidad de la GPU disponible. Para ello se propone una integración entre la estructura de datos octree y el algoritmo *Raycasting*. El funcionamiento se puede separar en tres simples pasos:

1. Construir el octree a partir de los datos volumétricos.
2. Reorganizar los nodos hojas no vacíos para representar primero los más lejanos y luego los más cercanos (*back-to-front order*).
3. Aplicar el algoritmo Raycasting para los nodos hojas no vacíos del octree y mezclar los resultados.

A continuación se describe el funcionamiento de los pasos antes mencionados, el epígrafe 2.1.1 explica los dos primeros pasos, mientras que el epígrafe 2.1.2 detalla el tercero.

2.1.1. Construir Octree

Una vez cargado un *dataset*, se le aplica la estructura de datos octree, para ello se subdivide en 8 nodos de igual tamaño, cada uno con la mitad de ancho, largo y profundidad que el nodo padre (en este caso el volumen original). Luego, cada nodo se clasifica en: **vacío**, si no contiene elementos del volumen y **ocupado**, si contiene elementos del volumen. La clasificación se realiza según el resultado de la comparación entre los valores de densidad de los vóxeles que ocupan el nodo y el umbral (*isovalue*) definido por el usuario.

El proceso se repite recursivamente en los nodos **ocupados** hasta alcanzar el nivel máximo de profundidad introducido por el usuario. Conjuntamente con este proceso se define si los nodos analizados tienen la clasificación de hoja como se muestra en el pseudocódigo del epígrafe 2.5.

Al término del proceso anterior se tiene como resultado un conjunto de nodos hojas, los cuales se adicionan a una lista (exceptuando los clasificados como **vacíos**) para facilitar su visualización (ver imagen 2.1 **a** y **b**). La lista se ordena de mayor a menor, según la distancia con respecto a la cámara empleando el método *sort* de la *Standard Template Library* (STL) de C++, lenguaje empleado para el desarrollo de la solución (ver epígrafe 2.6.7).

2.1.2. Visualización

Una vez concluida la división, clasificación y el ordenamiento, se procede a la visualización de la textura 3D para cada nodo. Para ello se empleó el algoritmo *Raycasting*. De manera tradicional, dicha representación se realiza una sola vez, pero, debido al gran tamaño de los volúmenes que se buscan visualizar, se modificó parte del funcionamiento. Para ello se recorre la lista resultante del paso anterior y mediante *Raycasting* se envía el fragmento de textura 3D correspondiente al nodo que se está analizando en ese momento a la GPU para su representación en pantalla.
Al término de la tarea anterior, la memoria principal se limpia y se repite el proceso con el siguiente nodo en la lista. Con el objetivo de mezclar los resultados se utilizó un *FrameBufferObject* de *OpenGL*. El proceso concluye una vez recorrida la lista; el resultado final es una imagen compuesta por la representación de los nodos de la estructura de datos octree de forma independiente, pero que muestra como un todo el volumen, logrando así visualizar un volumen que excede la capacidad de la GPU disponible (ver imagen 2.1 c). El siguiente pseudocódigo refleja el proceso de ordenamiento y visualización.

```
1 NlogNorderedRender(RaycastingBase raycasting){
2         //se define el criterio de ordenamiento de la lista de hojas.
3         sort(hojas.begin(), hojas.end(), comparar);
4         Para i=0 hasta hojas.size() incremento 1 hacer{
5         //representar fragmento de textura.
6         }
7 }
```

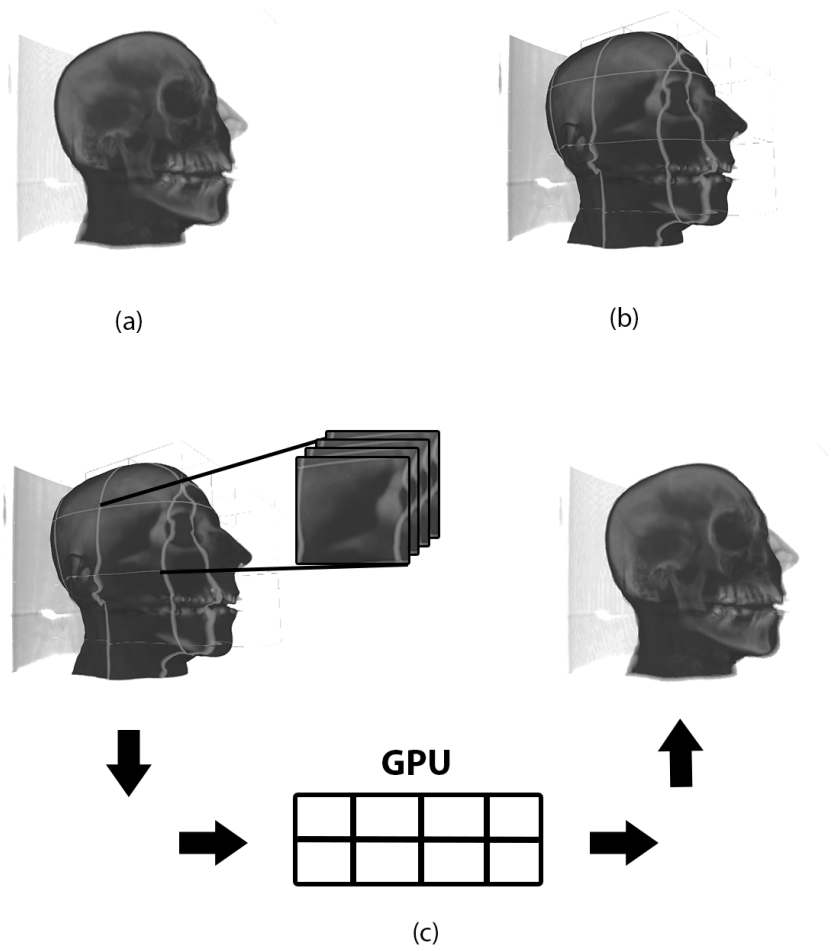

Figura 2.1. Flujo de funcionamiento de la solución. a: Volumen cargado en una PC con GPU superior al tamaño del modelo. b: Modelo luego de aplicarle la estructura de datos octree. c: Carga de la textura 3D de cada nodo no vacío, para su posterior envío a la GPU y lograr representar el modelo en una PC con GPU inferior al tamaño del volumen.

2.2. Integración con la arquitectura de Vismedic - Illustration

El software Vismedic-Illustration presenta una arquitectura modular que le permite soportar de manera natural y ordenada la ejecución de los pasos necesarios para generar una ilustración mediante DVR [11]. Para ello presenta diferentes módulos especializados en realizar el trabajo de cada paso (Ver imagen 2.2). Vismedic-Illustration presenta en algunos módulos interfaces de *plugins* para de esta forma aumentar sus funcionalidades sin modificar el comportamiento general.

La solución desarrollada se adiciona al módulo Visualización. Esta hereda de las clases ***Volume*** y ***Raycasting-Base***. Las clases adicionadas con el fin de cumplir el objetivo propuesto se distribuyeron de la siguiente forma: la clase principal ***Octree*** (ver epígrafe 2.4) en el núcleo del módulo Visualización, mientras

que las clases ***Octree-Plugin*** y ***Octree-Raycasting*** (ver epígrafe 2.4) en la interfaz de *plugins* (ver imagen 2.2).

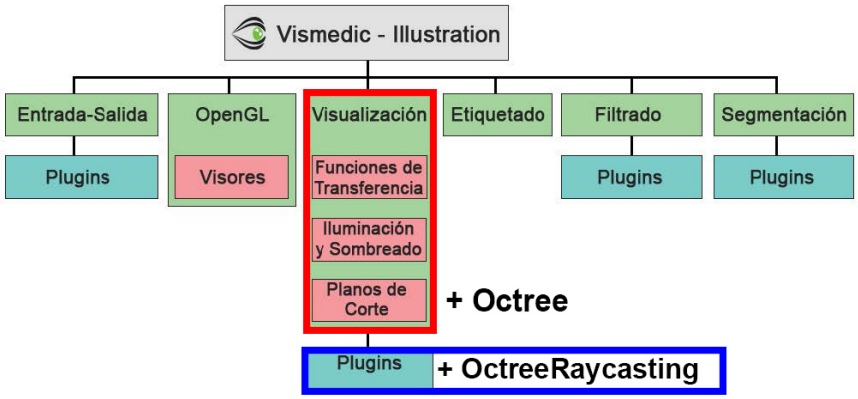

Figura 2.2. Integración de la solución desarrollada con la arquitectura general de Vismedic-Illustration.

Los objetos con carácter global de Vismedic - Illustration se comparten en la clase ***CoreSettings***, diseñada utilizando el patrón Singleton para facilitar su acceso. Además Vismedic-Illustration ofrece un conjunto de funcionalidades para la comparación de algoritmos de visualización, algunas de estas son: calcular los cuadros por segundo, guardar la imagen de la visualización actual e intercambiar los algoritmos de visualización, manteniendo el volumen de datos actual y la proyección; esta última facilita la comparación de dos o más algoritmos de visualización, teniendo en cuenta la calidad de la imagen generada para los mismos datos de entrada. A continuación se describen brevemente las funcionalidades de los módulos presentes en el *software*.

- **Módulo Entrada-Salida**
  Es el punto de inicio cuando se representa cualquier volumen. Las principales clases de este módulo son ***Volume*** y ***VolumeReader***. La primera se encarga de almacenar la descripción del volumen y una referencia a los datos volumétricos en memoria. La clase ***VolumeReader*** es la encargada de leer los archivos desde la unidad de almacenamiento y convertirlos a la representación interna de Vismedic-Illustration [11].
- **Módulo Filtrado**
  Es el encargado de filtrar el volumen para contribuir a mejorar la calidad de la imagen final, mediante la reducción del ruido presente en el volumen, el resaltado de los bordes o el aumento del contraste. El proceso de filtrado incide directamente en la calidad de la segmentación y la calidad final de la visualización.
- **Módulo Segmentación**
  Tiene como función dividir el volumen en regiones de interés, lo que posibilita el tratamiento independiente, aplicarles diferentes funciones de transferencia o algoritmos de visualización. Este módulo contribuye además a mejorar la calidad de la visualización ilustrativa.

- **Módulo OpenGL**
  Tiene la responsabilidad de ofrecer un contexto de visualización que permita aprovechar las potencialidades de las tarjetas gráficas actuales y facilitar la creación de diferentes estructuras de visores. Está escrito completamente usando C++, *OpenGL* y GLSL para el trabajo en la GPU.
- **Módulo Visualización**
  La función principal de este módulo es generar una proyección tridimensional mediante DVR, de los datos contenidos en la clase ***Volume***, tomando como parámetros los valores del visor y la cámara actual. Desde el punto de vista algorítmico, es el módulo más complicado. Dentro de sus funcionalidades se encuentran la generación de texturas 3D para el algoritmo *Raycasting*, la ejecución de la variante de *Raycasting* implementada y la activación y desactivación de los *shaders*.
- **Módulo Etiquetado**
  El módulo en cuestión obtiene sus entradas desde el sistema de visores y desde el módulo de Visualización. Los visores sirven de guía para ubicar etiquetas, mientras que el módulo de Visualización provee la imagen generada y los parámetros de la representación. La clase principal del módulo es ***LabelManager*** y es la encargada de almacenar las propiedades de las etiquetas (coordenadas 3D en el espacio de imagen, posición global, texto, radio y color de fondo) y calcular la disposición final sobre la imagen.

2.3. Requerimientos del sistema

Los requerimientos de un sistema son las descripciones de los servicios que proporciona y sus restricciones operativas. Ellos reflejan las necesidades de los clientes de un sistema que ayuda a resolver algún problema. Los requerimientos del sistema reflejan con detalle las funciones, servicios y restricciones operativas del sistema [24].

2.3.1. Requisitos funcionales

Los requisitos funcionales son las declaraciones de los servicios que debe proporcionar el sistema, de la manera que debe reaccionar a entradas particulares y de cómo debe comportarse en situaciones particulares. En algunos casos pueden declarar también lo que el sistema no debe hacer [24]. A continuación se presentan los requerimientos funcionales identificados:

- **RF1:** Crear octree.
- **RF2:** Eliminar nodos vacíos.
- **RF3:** Ordenar nodos hojas.
- **RF4:** Configurar octree.
- **RF5:** Visualizar volumen.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales son restricciones de los servicios o funciones que ofrece el sistema. En ellos se incluyen las restricciones de tiempo sobre el proceso de desarrollo y estándares. A menudo estos requisitos son aplicables al sistema en su totalidad pero normalmente se aplican a características o servicios individuales del sistema [24]. Seguidamente se reflejan los requisitos no funcionales identificados en la solución.

Hardware

Como requerimiento mínimo se recomienda que la estación de trabajo posea una tarjeta gráfica de al menos 512 MB con soporte para *OpenGL* 3 y RAM de al menos 2 GB.

2.4. Descripción de las clases fundamentales

Una clase es una plantilla para la creación de objetos de datos según un modelo ya definido. Cada una es un modelo encargado de definir variables, su estado y los métodos para operar con ellas. Las clases son un pilar indispensable en la Porgramación Orientada a Objetos. La solución propuesta se encuentra estructurada en diferentes clases con funciones particulares y los algoritmos necesarios para cumplirla. Esta estructura permite una organización del código por funciones, facilitando el mantenimiento en caso de la ocurrencia de errores. Las clases implementadas para la solución se encuentran en el módulo Visualización del *software* Vismedic - Illustration; a continuación se presenta una descripción.

- ***Octree***
  Define la estructura general del árbol donde se almacenarán los elementos del volumen, en conjunto con los métodos necesarios a la hora de trabajar con ellos; hereda de las clases ***Volume*** y ***Raycasting-Base***. Sus principales atributos son:
  - m_parent - Almacena el nodo padre del nodo actual.
  - m_children - Almacena el arreglo de nodos hijos del nodo actual.
  - m_level - Indica el nivel en el cual se encuentra el nodo.
  - m_maxlevel - Indica el máximo nivel que podrá alcanzar el árbol.
  - m_width - Indica el ancho del nodo actual.
  - m_heigth - Indica el alto del nodo actual.
  - m_depth - Indica la profundidad que tiene el nodo actual.
  - m_isleaf - Indica si el nodo es hoja.
  - m_isEmpty - Indica si el nodo está vacío.
  - m_data - Almacena la información del nodo que será representado en pantalla.

  Entre los principales métodos se encuentran:
  - Octree - Constructor de la clase, encargado de aplicar la estructura de datos al volumen cargado.

- createTexture3D - Es el encargado de crear el fragmento de textura 3D de cada nodo no vacío del árbol.
- NlogNorderedRender - Encargado de representar en pantalla la lista de nodos hojas ordenados mediante un algoritmo de ordenamiento con complejidad temporal $O(n \log n)$.

- ***Octree-Plugin***
  Incorpora el nuevo plugin a la arquitectura del proyecto; hereda de la clase ***Render-Algorithm-Plugin-Interface***. Sus principales métodos son:
  - OctreeRaycastingPlugin - Constructor de la clase.
  - instantiateRenderAlgorithm - Encargado de inicializar el algoritmo.

- ***Octree-Raycasting***
  Implementa una variante del algoritmo *Raycasting* que utiliza la estructura de datos octree, implementada en este trabajo; hereda de las clases ***Octree*** y ***Raycasting-Base***. Como atributo de la clase se encuentra m_octree, el cual será el encargado de almacenar el árbol resultante. Entre sus principales métodos se encuentran:
  - initOctree - Encargado de inicializar los valores que se emplearán para la construcción de la estructura de datos.
  - render - Encargado de representar en pantalla el volumen resultante.

2.5. Algoritmos implementados

Los algoritmos son conjuntos de instrucciones o reglas prescritas, las cuales son finitas, bien definidas y ordenadas y permiten llevar a cabo actividades mediante pasos sucesivos [25]. Con motivo de resolver el problema de la investigación propuesto se implementaron diferentes algoritmos, cada uno con una función específica. De igual manera se buscará optimizar el rendimiento del código para no sobrecargar innecesariamente el rendimiento de la estación de trabajo.

Octree

El algoritmo octree es el encargado de aplicar la estructura de datos al volumen que se está procesando en ese momento. Este es un método recursivo que se llama para cada nodo. En la solución propuesta se le adicionó además la función de insertar los nodos hojas a una lista, la cual permite representar los elementos en la pantalla, comenzando siempre por el más alejado de la cámara. A continuación se muestra el pseudocódigo de la variante implementada en este trabajo.

```
1 Octree(Octree padre, Volumen volumen, entero nivel, entero X, entero Y, entero
      Z, entero nivelMaximo){
2         Si (nivel = 0){
3         // inicializar valores de las variables con los valores del volumen.
4         }Sino{
```

```
5           // inicializar valores de las variables con la mitad de los valores
                del volumen.
6           }
7           // se determina si el nodo actual es vacío.
8           // se determina si el nodo actual es hoja.
9           Si (no es hoja){
10                  // inicializar los 8 hijos del nodo actual.
11                  Si (nivel < nivelMaximo){
12                          // aplicar el algoritmo recursivamente a cada uno de
                                los nodos hijos.
13                          }
14          }Sino{
15          // asignarle los datos que representa dicho nodo y agregarlo a la
                lista de hojas.
16          }
17  }
```

2.6. Metodología y herramientas de desarrollo

Las siguientes secciones muestran la metodología de desarrollo de software empleada durante la elaboración del presente trabajo, así como las principales herramientas que asistieron el proceso de diseño y la implementación de la solución propuesta.

2.6.1. Metodología de desarrollo de software

La metodología de desarrollo de software escogida fue Programación Extrema (XP, por sus siglas en inglés). Esta pertenece al grupo de metodologías ágiles en la que se da máxima prioridad a la obtención de resultados y reduce la burocracia utilizada por las metodologías tradicionales. La metodología XP se ajusta a una serie de reglas que están centradas en las necesidades del cliente para así lograr un producto de buena calidad en poco tiempo. Su filosofía es satisfacer por completo las necesidades del cliente, por lo que lo integra como un miembro más del equipo de desarrollo. La metodología está diseñada para el desarrollo de aplicaciones que requieran un grupo pequeño de programadores, donde la comunicación sea más factible [26]. Sus características principales son:

- Permite introducir requisitos nuevos o cambiar los anteriores de manera ágil.
- Es adecuada para proyectos pequeños o medianos.
- Es adecuada para proyectos con alto riesgo.
- Tiene un ciclo de vida iterativo e incremental.
- Cada una de sus iteraciones dura alrededor de 3 semanas.
- No produce demasiada documentación acerca del diseño o la planificación.

- Frecuente integración del equipo de programación con el cliente.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas.

Concretamente dicha metodología hace principal énfasis en la adaptabilidad en lugar de la previsibilidad. Por eso contempla los cambios de requisitos sobre la marcha, pues es algo común dentro del proceso de desarrollo de software [27].

2.6.2. Historias de usuario

Las Historias de Usuario (HU) son las técnicas utilizadas en XP para especificar los requisitos de *software*. En ellas el cliente describe brevemente las características que el sistema debe poseer, ya sean requisitos funcionales o no funcionales. Su tratamiento es dinámico y flexible debido a que en cualquier momento pueden desecharse, reemplazarse por otras, añadirse nuevas o ser modificadas [28]. La actual sección presenta la descripción de las HU elaboradas para el desarrollo de la solución.

Tabla 2.1. Historia de usuario # 1

Historia de usuario	
Número: 1	**Nombre:** Crear octree
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1.5	**Iteración asignada:** 1
Programador responsable: Antonio David Castillo Oliva	
Descripción: El sistema debe ser capaz de generar la estructura de datos octree a partir de un volumen ya cargado por el *software* Vismedic - Illustration.	
Observaciones:	
Interfaz: N/A	

Tabla 2.2. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Eliminar nodos vacíos
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 1	**Iteración asignada:** 1
Programador responsable: Antonio David Castillo Oliva	
Descripción: El sistema debe ser capaz de eliminar los nodos del árbol que no contengan elementos del volumen.	
Observaciones:	

Continúa en la próxima página

Tabla 2.2. Continuación de la página anterior

Interfaz:
N/A

Tabla 2.3. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Ordenar nodos hojas
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 1	**Iteración asignada:** 2
Programador responsable: Antonio David Castillo Oliva	
Descripción: El sistema debe ser capaz de ordenar los nodos hojas del volumen con respecto a la distancia a la cámara.	
Observaciones:	
Interfaz: N/A	

Tabla 2.4. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Configurar octree
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 0.7	**Iteración asignada:** 2
Programador responsable: Antonio David Castillo Oliva	
Descripción: El sistema debe permitir al usuario cambiar los parámetros empleados para la construcción de la estructura de datos (cantidad de niveles e *isovalue*) y recalcular el volumen a partir de estos nuevos valores.	
Observaciones:	
Interfaz: N/A	

Tabla 2.5. Historia de usuario # 5

Historia de usuario

Continúa en la próxima página

Tabla 2.5. Continuación de la página anterior

Número: 5	**Nombre:** Visualizar volumen
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 1.2	**Iteración asignada:** 2
Programador responsable: Antonio David Castillo Oliva	
Descripción: El sistema debe ser capaz de representar el volumen, una vez aplicada la estructura de datos octree, de forma tal que los elementos más cercanos a la cámara sean los últimos en representarse.	
Observaciones:	
Interfaz: N/A	

2.6.3. Estimación de esfuerzo por historias de usuario

Una vez realizadas todas las HU, se realizó la estimación de esfuerzo de cada una. De esta forma se puede observar el esfuerzo total necesario para desarrollar la solución, así como el esfuerzo individual para cada HU. La tabla 2.6 refleja los resultados obtenidos.

Tabla 2.6. Estimación de esfuerzo por historia de usuario

Iteración	Historias de usuario		Puntos estimados (semanas)
1	1	Crear octree	1.5
	2	Eliminar nodos vacíos	1
2	3	Ordenar nodos hojas	1
	4	Configurar octree	0.7
	5	Visualizar volumen	1.2
Total			**5.4**

2.6.4. Plan de iteraciones

Con el objetivo de identificar la duración de cada iteración de la solución propuesta, se elaboró un plan de iteraciones, donde se refleja además la duración total del proyecto. La tabla 2.7 muestra los resultados obtenidos luego de la elaboración del plan.

Tabla 2.7. Plan de duración de las iteraciones

Iteración	Historias de usuario		Duración (semanas)
1	1	Crear octree	2.5
	2	Eliminar nodos vacíos	

Continúa en la próxima página

Tabla 2.7. Continuación de la página anterior

2	3	Ordenar nodos hojas	2.9
	4	Configurar octree	
	5	Visualizar volumen	
Total			**5.4**

2.6.5. Tarjetas CRC

Las tarjetas Clase, Responsabilidad y Colaboradores (CRC) son las encargadas de describir las propiedades de las clases propias del sistema. El objetivo de las tarjetas CRC es desarrollar, discutir y evaluar modelos orientados a objetos. El principal problema de estas es que difuminan la diferencia entre los objetos de las clases [29]. A continuación se reflejan las tarjetas realizadas para cada clase de la solución.

Tabla 2.8. Tarjeta CRC # 1

Tarjeta CRC	
Clase: Octree	
Responsabilidad	**Colaboración**
• Crear octree • Eliminar nodos vacíos • Ordenar nodos hojas	Volume Core Core-Settings Raycasting-Base Gl-Viewport

Tabla 2.9. Tarjeta CRC # 2

Tarjeta CRC	
Clase: Octree-Plugin	
Responsabilidad	**Colaboración**
• Integrar la solución a Vismedic-Illustration • Representar la interfaz gráfica de la solución	Render-Algorithm-Plugin-Interface Octree-Raycasting

Tabla 2.10. Tarjeta CRC # 3

Tarjeta CRC	
Clase: Octree-Raycasting	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla 2.10. Continuación de la página anterior

• Inicializar los valores necesarios para aplicar el algoritmo octree • Visualizar volumen	Raycasting-Base Octree

2.6.6. Herramientas de desarrollo

Para la implementación se utilizó Qt, *framework* multiplataforma orientado a objetos que utiliza el lenguaje C++ de forma nativa. Además funciona en las principales plataformas y tiene una gran comunidad de desarrolladores [30]. Por otro lado, fue la herramienta empleada durante el desarrollo del *software* Vismedic - Illustration, lo que facilita la integración con la solución. Entre sus principales características se encuentran:

- Presenta herramientas para la rápida navegación por el código.
- Permite desplegar el sistema en múltiples plataformas con el mismo código base.
- Permite a los desarrolladores centrarse en la producción de código y no en las particularidades del sistema operativo.
- Brinda acceso total al código fuente para su revisión y edición.

2.6.7. Lenguaje de programación

Como lenguaje de programación se utilizó C++, lenguaje imperativo orientado a objetos derivado de C. Al igual que su predecesor, se encuentra muy ligado al *hardware* subyacente, brindándole una considerable potencia en la programación a bajo nivel. Sin embargo, se le han añadido elementos que le permiten un estilo de programación con alto nivel de abstracción. C++ es el lenguaje por excelencia de las aplicaciones de realidad virtual y hace uso eficiente del paradigma de programación orientada a objetos. Brinda control de memoria y administración de los recursos de la estación de trabajo [31]. Se selecciona además, debido a que es el lenguaje empleado en Vismedic - Illustration. Dentro de sus principales ventajas se encuentran:

- **Difusión:** debido a que es uno de los lenguajes más empleados actualmente, posee un gran número de usuarios y excelente bibliografía.
- **Versatilidad:** se puede emplear para resolver cualquier tipo de problemas ya que es un lenguaje de propósito general.
- **Portabilidad:** el código fuente puede ser compilado para diferentes plataformas dado que se encuentra estandarizado.
- **Eficiencia:** es uno de los lenguajes más rápidos en cuanto a tiempo de ejecución.
- **Herramientas:** existe una gran cantidad de compiladores, depuradores y bibliotecas basados en este lenguaje.

2.7. Consideraciones parciales

Al término del presente capítulo se puede concluir que el empleo del algoritmo de octree en conjunto con el algoritmo *Raycasting* permiten la visualización de conjuntos de datos que no podían ser representados anteriormente; resulta factible aplicar la metodología de desarrollo XP debido a que el equipo de desarrollo es pequeño. Además las HU permitieron un mayor entendimiento de los requisitos del sistema, así como el esfuerzo estimado para cada uno. De igual forma el plan de iteraciones permitió visualizar el tiempo estimado para el desarrollo de la solución y de esta forma hacer una correcta planificación.

CAPÍTULO 3

Evaluación de los resultados

En el presente capítulo se evalúan los principales resultados obtenidos en la investigación. Primeramente se describe el entorno donde se aplicaron las pruebas, haciendo énfasis en las características principales de la estación de trabajo (RAM, Unidad Central de Procesamiento (CPU, por sus siglas en inglés) y GPU). Luego se hace una descripción de los conjuntos de datos empleados en las pruebas, resaltando la cantidad de *bits*, la resolución, cantidad de cortes y la memoria RAM que ocupan. Por último, una vez concluida la ejecución de las pruebas, se muestran evidencias visuales de los resultados alcanzados con volúmenes de datos incapaces de representar sin el empleo de la estructura de datos octree.

3.1. Pruebas de software

Las aplicaciones informáticas no están exentas de errores, es por ello que se les aplica un conjunto de pruebas para garantizar que cuentan con la calidad requerida. Las pruebas del *software* son un elemento crítico para la garantía de la calidad y representan una revisión final de las especificaciones, del diseño y la codificación. Existen varias normas que pueden servir acertadamente como objetivos de las pruebas [32]:

- La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Las pruebas no pueden demostrar que el *software* está libre de defectos o que se comportará en todo momento como se espera. Siempre es posible que una prueba que se haya pasado por alto pueda descubrir problemas adicionales con el sistema [24].

3.2. Entorno de pruebas

Las pruebas se realizaron en una PC con las siguientes propiedades:

- **CPU:** Intel Core i3 2130
- **RAM:** 4 GB DDR3 a 1333 Mhz
- **GPU:** NVidia GeForce 9800 GT de 512 MB

Es válido resaltar que las prestaciones de la PC disponible no son las idóneas para la visualización de grandes volúmenes, debido a la obsolescencia de sus componentes. Además, la tarjeta gráfica tiene una década de explotación, lo que ha disminuido considerablemente su rendimiento.

3.3. Conjuntos de datos para las pruebas

Para la realización de las pruebas se emplearon los *datasets* disponibles en el *Visible Human Project*. Los modelos están diseñados para servir como punto de referencia común para el estudio de la anatomía humana, como conjuntos de datos comunes de dominio público para realizar pruebas a algoritmos de imágenes médicas y como banco de pruebas [33]. El proyecto es dirigido por la Biblioteca Nacional de Medicina de los Estados Unidos y sus modelos se encuentran disponibles públicamente. La tabla 3.1 muestra los detalles de los modelos seleccionados para aplicar las pruebas.

Tabla 3.1. Descripción de los *datasets* empleados para la realización de las pruebas.

Dataset	Cantidad de bits	Resolución	Cantidad de cortes	RAM que ocupa
female.raw	16	512x512	700	350 MB
female-lg.raw	8	512x512	1000	250 MB
male.raw	16	512x512	1024	512 MB
male-lg.raw	8	512x512	1624	406 MB
VisibleMale.raw	16	512x512	750	375 MB

Es necesario destacar que la memoria de la GPU es compartida con el resto de las aplicaciones en uso del sistema operativo, por tanto, si los modelos presentan tamaños cercanos a la capacidad, resulta imposible visualizarlos a pesar de no superar el tamaño de la tarjeta gráfica.

3.4. Ejecución de las pruebas

La tabla 3.2 muestra las pruebas realizadas a la solución desarrollada junto con los conjuntos de datos empleados para cada una y los resultados arrojados. De igual forma, la tabla 3.3 refleja los resultados obtenidos al intentar cargar todos los modelos disponibles.

Tabla 3.2. Descripción de las pruebas realizadas y resultados obtenidos.

Prueba	Modelo	Dimensiones	Frames por segundo	Resultados de la prueba

Representar volumen.	female.raw	512x512x700	2	El software muestra el modelo seleccionado luego de aplicarle la estructura de datos octree.
	female-lg.raw	512x512x1000	6	
Representar volumen mayor a la capacidad de la GPU.	male-lg.raw	512x512x1624	4	Mediante el empleo de la estructura de octree, el software es capaz de representar un volumen que exceda las capacidades de la GPU disponible.
Modificar valores de octree.	female-lg.raw	512x512x1000	6	El software permite modificar la cantidad de niveles empleados en la construcción de octree y el *isovalue* y recalcular la estructura a partir de los nuevos valores.

Tabla 3.3. Resultados obtenidos al intentar cargar los volúmenes disponibles con la estructura de datos implementada.

Volumen	Dimensiones	Visualiza
female	512x512x700	Si
female-lg	512x512x1000	Si
male	512x512x1024	Si
male-lg	512x512x1624	Si
VisibleMale	512x512x750	Si

3.4.1. Cantidad de niveles del Octree

La cantidad de niveles empleados en la construcción de la estructura de datos octree define la cantidad de subdivisiones que tendrá el volumen. Mientras más subdivisiones presente, mayor será el nivel de detalle reflejado. De igual forma, una mayor cantidad de nodos trae consigo mayor nivel de procesamiento, por lo

que sería necesario un procesador potente si se emplea un alto nivel de octree. El valor empleado durante las pruebas fue 2; siendo suficiente para lograr la visualización de la mayoría de los modelos empleados con un rendimiento de 2 cuadros por segundo (ver imagen 3.1).
El aumento de la cantidad de niveles a 3, no provocó cambios en el rendimiento (ver tabla 3.4); en otros casos, cuando se combinó con el aumento del umbral, se detectó mejoría en cuanto al rendimiento, pasando de 2 cuadros por segundo a 4 (ver imagen 3.2). En los casos de volúmenes cuyo tamaño es muy similar a la capacidad del GPU, el aumento de dicho valor, provocó una sobrecarga en la estación de trabajo y por consecuente la finalización forzada del programa.

Tabla 3.4. Resultados obtenidos según los niveles de octree empleados.

Volumen	Dimensiones	Niveles	Cuadros por segundo
female	512x512x700	2	2
female	512x512x700	3	2
female-lg	512x512x1000	2	2
female-lg	512x512x1000	3	2
male	512x512x1024	2	2
male	512x512x1024	3	2
male-lg	512x512x1624	2	2
male-lg	512x512x1624	3	2
VisibleMale	512x512x750	2	2
VisibleMale	512x512x750	3	2

3.4.2. Resultados de eliminar los espacios en blanco

La eliminación de elementos de volumen innecesarios hace posible la representación de *datasets* que exceden la capacidad de la GPU. El criterio empleado para determinar si un nodo se encuentra vacío, es la comparación entre los valores de densidad de los vóxeles que ocupan el nodo con el valor del umbral (*isovalue*) introducido por el usuario (si es mayor se considera ocupado, en caso contrario vacío). En las pruebas realizadas el valor fue de 100; para la mayoría de los modelos empleados el valor fue acertado para la eliminación de elementos innecesarios del volumen, alcanzando un rendimiento de 2 cuadros por segundo (ver imagen 3.1). Con el objetivo de buscar mejorías en el rendimiento el valor fue elevado a 120, logrando así disminuir aun más el tamaño del modelo gracias a la eliminación de nodos sin elementos de volumen que anteriormente no eran detectados, en estos casos el rendimiento aumentó hasta 4 cuadros por segundo 3.2).
Con el fin de verificar el comportamiento del sistema, se realizaron comprobaciones con todos los volúmenes disponibles. Para ello se determinó el rendimiento al emplear valores de umbral 100 y 200. El aumento del valor, trajo consigo, en algunos casos, mejorías en el rendimiento, debido a que se disminuye la cantidad de nodos que se visualizan. Los resultados obtenidos se reflejan en la tabla 3.5.

Tabla 3.5. Resultados obtenidos según el valor del umbral(*isovalue*) empleado.

Volumen	Dimensiones	Umbral	Cuadros por segundo
female	512x512x700	100	2
female	512x512x700	200	2
female-lg	512x512x1000	100	2
female-lg	512x512x1000	200	8
male	512x512x1024	100	2
male	512x512x1024	200	2
male-lg	512x512x1624	100	2
male-lg	512x512x1624	200	4
VisibleMale	512x512x750	100	2
VisibleMale	512x512x750	200	2

3.5. Pruebas de aceptación

Las pruebas de aceptación se crean en base a las HU de cada iteración del ciclo de desarrollo. El cliente especifica uno o varios escenarios para cada HU, con el fin de verificar si ha sido correctamente implementada. Una HU no se puede considerar terminada hasta tanto no pase correctamente todas las pruebas de aceptación [34]. En la tabla 3.6 se muestran los escenarios de pruebas definidos y los resultados obtenidos en cada uno.

Tabla 3.6. Descripción de los escenarios de pruebas.

id del escenario	Escenario	Variable	Respuesta del sistema	Resultados de la prueba
1	Crear octree	nivel, *isovalue*	El sistema genera la estructura de octree sobre el volumen cargado en el momento.	El sistema visualiza el volumen cargado una vez aplicado el octree.
2	Cambiar valores	nivel, *isovalue*	El sistema permite el cambio de los valores empleados en la construcción del octree y recalcula la estructura a partir de estos nuevos valores.	Una vez modificados los valores de las variables el sistema reconstruye la estructura a partir estos.

3	Eliminar espacios en blanco	nivel, *isovalue*	El sistema elimina las secciones del volumen que no contengan elementos del modelo.	El sistema elimina los espacios en blanco del volumen permitiendo la representación de *datasets* que excedan la capacidad de la GPU.
4	Ordenar nodos hoja	nivel, *isovalue*	El sistema ordena los nodos hojas del árbol de mayor a menor con respecto a la distancia a la cámara.	El sistema ordena los nodos de forma tal que los nodos más cercanos al observador sean los últimos que se representan.

3.6. Resultados visuales

La siguiente sección muestra resultados visuales de la solución desarrollada variando la cantidad de niveles y el valor de *isovalue*.

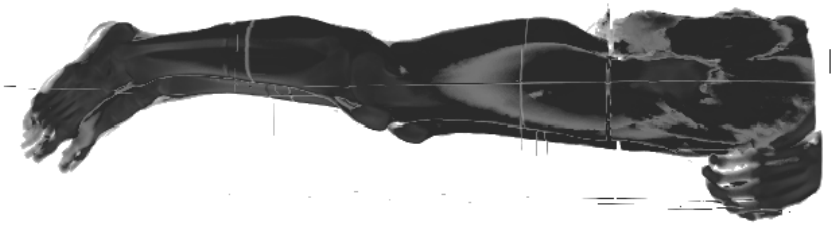

Figura 3.1. Visualización del modelo **female-lg.raw** con 2 niveles de octree e *isovalue* de 100.

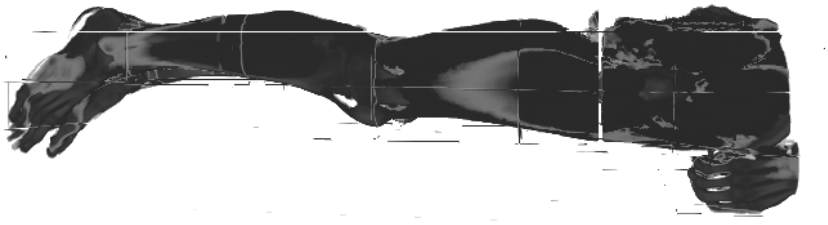

Figura 3.2. Visualización del modelo **female-lg.raw** con 3 niveles de octree e *isovalue* de 120.

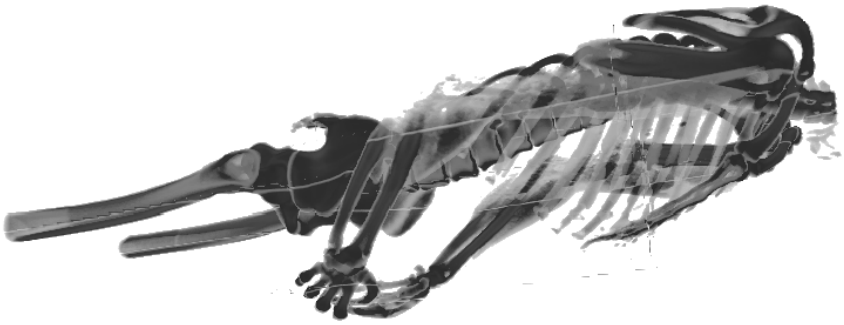

Figura 3.3. Visualización del modelo **male-lg.raw** de 406 MB de tamaño (imposible de representar anteriormente) empleando 2 niveles de octree e *isovalue* con valor 100.

3.7. Consideraciones parciales

En este capítulo se discutieron los resultados obtenidos, tanto visuales como de rendimiento. De igual forma se verificó si la solución desarrollada cumple todos los requisitos definidos. De forma general se puede resumir que:

- El constante intercambio de información entre la CPU y la GPU disminuye el rendimiento del Ray-casting, pero permite visualizar volúmenes que antes no era posible observarlos.
- El empleo de 2 niveles de octree son suficientes para la representación de la mayoría de los modelos.
- La eliminación de los espacios en blanco, aunque requiere un octree de más niveles, mejoró la cantidad de cuadros por segundo o FPS (del inglés «*frames per second*») de la visualización de volúmenes de dimensiones de 512^3 en adelante.

Conclusiones

Al término del presente trabajo se adicionó al proyecto Vismedic - Illustration un nuevo algoritmo de visualización que emplea la estructura de datos octree para representar modelos de datos que exceden la capacidad de la tarjeta gráfica disponible. Con lo que se da cumplimiento al objetivo planteado al inicio de la investigación. Además se concluye que:

- La implementación de un algoritmo de visualización basado en octree, permitió la representación de modelos que exceden la capacidad de la GPU disponible.
- El empleo de 2 niveles de octree fue suficiente para lograr representar la mayoría de los modelos; con un aumento en la cantidad se puede elevar el rendimiento y alcanzar mayor nivel de detalle.
- La eliminación de los espacios en blanco fue fundamental para lograr visualizar volúmenes mayores a la capacidad de la GPU y el empleo de un correcto valor de *isovalue* trae consigo la eliminación de la mayoría de los espacios innecesarios y, por tanto, mejorías en el rendimiento.

Recomendaciones

Aunque la estructura de datos propuesta, de conjunto con las modificaciones al algoritmo *Raycasting*, resuelven el problema planteado al inicio de la investigación, se pueden añadir mejoras que contribuyan al rendimiento y a la calidad de las representaciones finales. Dentro de las principales mejoras a considerar para trabajos futuros se encuentran las siguientes:

- Incorporación de otras técnicas de optimización, tales como Multiresolución y Oclusión (*Oclussion Culling*), con el objetivo de aumentar los cuadros por segundo, y por consiguiente, la interactividad de la aplicación.
- Extender la estructura de datos implementada y las modificaciones del algoritmo *Raycasting* al resto de los *plugins* de Vismedic Illustration, con la correspondiente adición de soporte para funciones de transferencia e iluminación.

Glosario

fragment shader Programa que calcula el color de los píxeles individuales. Controla como las texturas son aplicadas a los fragmentos. 11, 12

shader Conjunto de instrucciones capaces de ser ejecutadas por un procesador gráfico. 12, 13

vertex shader Función del procesador gráfico que manipula los valores de un vértice en un plano 3D mediante operaciones matemáticas sobre un objeto. Estas variaciones pueden ser diferentes en color, en las coordenadas de la textura, en la orientación en el espacio o en el tamaño del punto. Permite el control de las transformaciones sobre los vértices. 13

vóxel La palabra proviene del término en inglés "*volumetric pixel*", es una unidad cúbica que compone un objeto tridimensional. Constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente del píxel en objeto 2D. 5, 10

DICOM En inglés (*Digital Imaging and Comuication in Medicine*) es el estándar reconocido mundialmente para el intercambio de imágenes médicas, pensado para el manejo, almacenamiento, impresión y transmisión de imágenes médicas. 2

Porgramación Orientada a Objetos Paradigma de la programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas como herencia, polimorfismo, abstracción y encapsulamiento. 22

píxel Abreviatura de "*picture element*". Es la mínima unidad de información dentro de una imagen bidimensional. 5, 11–13

Acrónimos

CPU Unidad Central de Procesamiento. 31, 32, 36
CRC Clase, Responsabilidad y Colaboradores. 28
CT Tomografías Computarizadas. 1, 7, 8, 10

DVR Visualización Directa de Volumen. 9, 11, 14, 17, 19, 21

fMRI Imágenes por Resonancia Magnética Funcional. 8

GPU Unidad de Procesamiento Gráfico. 2–4, 11–13, 15, 16, 18–21, 31–37

HU Historias de Usuario. 25, 27, 30, 34

IVR Visualización Indirecta de Volumen. 9

MRI Imágenes por Resonancia Magnética. 1, 8

PC Computadora Personal. 2, 16, 20, 32
PET Tomografías por Emisión de Positrones. 1, 8

RAE Real Academia de la Lengua Española. 5
RAM Memoria de Acceso Aleatorio. 16, 22, 31, 32

UCI Universidad de las Ciencias Informáticas. 2

XP Programación Extrema. 25, 30

Referencias bibliográficas

[1] Miguel Ángel Verdugo Alonso, Ortíz González, María del Carmen et al., *Personas con discapacidad: perspectivas psicopedagógicas y rehabilitadoras*. 376. Siglo Veintiuno de España, 2005 (vid. pág. 1).

[2] Dani Tost, Anna Puig e Isabel Navazo. «Visualización de volumen: Aplicaciones médicas». En: *Informática gráfica* 19 (1998), pág. 179 (vid. pág. 1).

[3] Luis Guillermo Silva Rojas. «Visualización directa de volumen para endoscopias virtuales». Tesis doct. Universidad de las Ciencias Informáticas, 2011 (vid. págs. 1, 13).

[4] Eduardo Sierra-Martínez, Ricardo Cienfuegos-Monroy y Gerardo Fernández-Sobrino. «OsiriX, visor DICOM útil para procesar imágenes tomográficas de fracturas faciales». En: *Cirugia y Cirujanos* 77.2 (2009) (vid. pág. 1).

[5] Jennis Meyer-Spradow et al., «Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations». En: *IEEE Computer Graphics and Applications* 29.6 (2009), págs. 6-13 (vid. pág. 2).

[6] Nadine Richard. «InViWo agents: write once, display everywhere». En: *Proceedings of the eighth international conference on 3D Web technology*. ACM. 2003, págs. 123-127 (vid. pág. 2).

[7] Klaus Engel et al., «Real-time volume graphics». En: *ACM Siggraph 2004 Course Notes*. ACM. 2004, pág. 29 (vid. pág. 2).

[8] D Ronda, O Ferrer y NA Alvarez. «Imagis: Sistema para la transmisión de imágenes médicas multimodales». En: *Memorias del II Congreso Latinoamericano de Ingeniería Biomédica, Habana*. 2001 (vid. pág. 2).

[9] Carlos Luis Castro Márquez y Alejandro Delgado García. «Visor de imágenes médicas digitales web». En: *Revista Cubana de Informática Médica* 6.1 (2014), págs. 57-70 (vid. pág. 2).

[10] Bernhard Preim y Charl P Botha. *Visual computing for medicine: theory, algorithms, and applications*. Newnes, 2013 (vid. págs. 5-8, 10, 14).

[11] Luis Guillermo Silva Rojas. «Visualización ilustrativa de datos volumétricos». Tesis de mtría. Universidad de las Ciencias Informáticas, 2017 (vid. págs. 6, 10, 11, 13, 20, 21).

[12] Johanna Beyer. «GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery». Tesis doct. Citeseer, 2009 (vid. págs. 7-9, 11).

[13] Marius Gavrilescu. «Visualization and Graphical Processing of Volume Data». Tesis doct. Institute of Computer Graphics y Algorithms, Vienna University of Technology, 2011 (vid. pág. 9).

[14] William E Lorensen y Harvey E Cline. «Marching cubes: A high resolution 3D surface construction algorithm». En: *ACM Siggraph Computer Graphics*. 1987 (vid. pág. 10).

[15] Nelson Max. «Optical models for direct volume rendering». En: *Visualization and Computer Graphics, IEEE Transactions on* (1995) (vid. pág. 11).

[16] Henning Scharsach. «Advanced raycasting for virtual endoscopy on consumer graphics hardware». En: (2005) (vid. pág. 12).

[17] M Hadwiger et al., «Advanced illumination techniques for GPU-based volume ray-casting». En: *ACM SIGGRAPH* (2009) (vid. pág. 12).

[18] Randi J Rost et al., *OpenGL shading language*. Pearson Education, 2009 (vid. pág. 13).

[19] Martti Mäntylä. «An introduction to solid modeling». En: *Computer science press* (1988) (vid. pág. 14).

[20] Héctor Yela Reneses e Isabel Navazo Álvaro. «Visualización de grandes volúmenes vía bricking y texturas 3D». En: (2007) (vid. pág. 16).

[21] Wagner Toledo Correa. «New techniques for out-of-core visualization of large datasets». Tesis doct. Princeton University, 2004 (vid. pág. 16).

[22] Craig M Wittenbrink. «Cellfast: Interactive unstructured volume rendering». En: *HP LABORATORIES TECHNICAL REPORT HPL* 81/REV (1999) (vid. pág. 16).

[23] Rajagopalan Srinivasan, Shiaofen Fang y Su Huang. «Volume rendering by templatebased octree projection». En: *Eurographics, Visualization in Scientific Computing* (1997) (vid. pág. 17).

[24] Ian Sommerville. *Ingeniería del Software*. Pearson Educación, S.A., 2005 (vid. págs. 22, 23, 32).

[25] Thomas H Cormen. *Introduction to algorithms*. The MIT Press, 2009 (vid. pág. 24).

[26] Borja López Yolanda. «Metodología Ágil de Desarrollo de Software–XP». En: *línea]. Disponible en: http://www. runayupay. org/publicaciones/2244_555_COD_18_290814203015. pdf.[Accedido: 03-jun-2016]* (2014) (vid. pág. 25).

[27] Rafael Graña Salgado. «Metodologías de desarrollo de proyectos informáticos en entornos web». B.S. thesis. Universitat Oberta de Catalunya, 2011 (vid. pág. 26).

[28] Patricio Letelier. «Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)». En: (2006). URL: http://www.cyta.com.ar/ta0502/b_v5n2a1.htm (vid. pág. 26).

[29] Jürgen Börstler y Carsten Schulte. «Teaching object oriented modelling with CRC cards and roleplaying games». En: *Proceedings WCCE*. Vol. 5. 2005 (vid. pág. 29).

[30] The QT Company. *Qt | Cross-platform software development for embedded & desktop*. URL: https://www.qt.io/ (vid. pág. 30).

[31] Salvador Pozo. «Curso de C++». En: (2000) (vid. pág. 30).

[32] Roger S. Pressman. *Ingeniería del Software: Un enfoque práctico*. McGrallHill, 2001 (vid. pág. 32).

[33] Michael J Ackerman. «The visible human project». En: *Proceedings of the IEEE* 86.3 (1998), págs. 504-511 (vid. pág. 33).

[34] José Joskowicz. «Reglas y prácticas en eXtreme Programming». En: *Universidad de Vigo* (2008), pág. 22 (vid. pág. 36).

Generado con LaTeX: 12 de junio de 2018: 11:58am